Mémoire de recherche bibliographique L3 maths-info

Étude de graphes pour la compensation mutuelle des dettes

Arthur Rousseau
Sous la direction de Nazim Fatès

Mai 2021

1 Introduction

Ce mémoire a pour objectif de faire une documentation des méthodes aujourd'hui disponibles pour résoudre le problème de compensation mutuelle des dettes.

1.1 Contexte économique

La dette entre entreprises pose des problèmes dans une économie, car elle induit des risques liés au non-remboursement, pouvant donner lieu à de graves incidents économiques. La résolution des problèmes présentés a pour but de réduire cette dette [1, 2]. Cela peut se faire de manière directe (réduction partielle de la dette) ou avec l'aide d'un acteur extérieur (suppression intégrale de la dette). Dans ce dernier cas, l'acteur extérieur peut être une banque, auquel cas elle récupère la somme investie avec intérêt, ou un état qui dispose d'un budget pour améliorer la santé de son économie. Le principe de toute réduction est qu'aucune entreprise ne soit favorisée ou au contraire lésée : ainsi, pour chaque entreprise, sa position nette (c'est-à-dire la somme qu'elle doit moins la somme qu'on lui doit) doit rester constante.
Les dettes peuvent apparaître notamment sur de courtes échéances (typiquement 3 mois) pendant lesquelles une prestation rendue n'est pas encore payée par l'entreprise qui en a bénéficié : celle-ci est endettée envers le prestataire. On est en situation de blocage économique, car personne ne rembourse tout de suite ses dettes, chacun étant lui-même en attente de remboursements.
Dans une économie réelle peuvent apparaître aussi des dettes négatives, qui correspondent à un remboursement à effectuer, ou même des dettes d'une entreprise à elle-même. Nous simplifions la situation en considérant que toutes les dettes sont positives et entre deux acteurs différents, situation à laquelle on peut toujours se ramener en supprimant les dettes d'une entreprise à elle-même et en remplaçant les dettes négatives par des dettes positives où créditeur et débiteur sont inversés.
La représentation sous forme de graphe, qui peut sembler naturelle, fait néanmoins l'objet d'étude récentes avec des modèles comme le réseau monétaire Sardex [3], avec l'évolution des systèmes financiers. Ce système consiste à considérer la dette comme une monnaie, et ainsi à pouvoir, en ayant une position nette positive, acheter avec de cette position nette plutôt qu'avec de l'argent réel. Une partie des dettes dues est alors déplacé de l'acheteur vers le fournisseur.
Le but des méthodes présentées est de réduire la dette entre entreprises. Les échéances pour la compensation sont de l'ordre de quelques jours à un mois : autrement dit, les algorithmes présentés sont appliqués pour réduire la dette tous les mois, voire toutes les semaines ou tous les deux jours si l'acteur de la réduction (banque, état) a une grande connaissance et une grande maîtrise de l'économie.

1.2 Exemple introductif

On présente le problème étudié par un exemple. On suppose que trois entreprises A, B, C ont des dettes les unes envers les autres, suite à des prestations facturées mais encore non payées. Les factures sont les suivantes : C doit 8€ à A, A doit 5€ à B, A doit 2€ à B, A doit 3€ à C, B doit 5€ à C, C doit 1€ à A. On remarque que C a deux dettes envers A, correspondant à deux factures différentes. On modélise cette situation par un graphe (Fig. 1). Notre but est de réduire la somme que se doivent ces trois entreprises, en maintenant les positions nettes indiquées en rouge. Cette position est négative si l'entreprise est débitrice et positive sinon.
La première méthode que nous étudierons est la *réduction partielle de la dette*, qui consiste à supprimer de le dette de manière cyclique. Ainsi, chacun garde la même position nette. Dans notre exemple, on peut éliminer 5€ dans les dettes de A à B, de B à C et de C à A (Fig. 2).

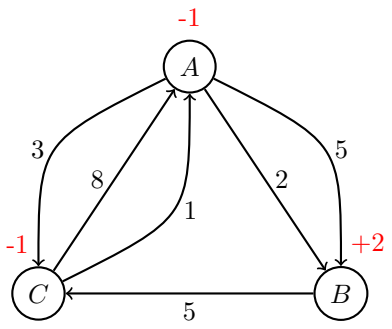


FIGURE 1 – Graphe initial

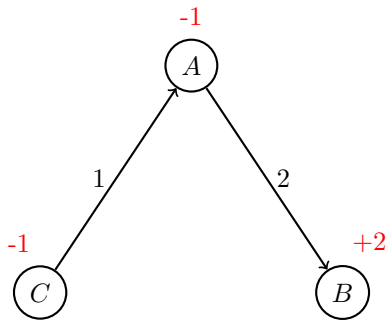


FIGURE 2 – Graphe après réduction partielle de la dette

La seconde, la *suppression intégrale de dettes*, consiste à injecter depuis un centre financier (banque, état) une somme d'argent, de manière à réduire au maximum, voire à éliminer la dette totale. Pour cela, on finance tout ou partie de la dette des entreprises de manière à éliminer un maximum de dette dans le graphe. Dans notre exemple, on peut en prêtant 1 à C (en vert) éliminer presque toute la dette du graphe. Souvent, la somme dont dispose le centre financier est limitée, ce qui nécessite de chercher les entreprises où injecter l'argent de manière à la quantité de dettes annulée. Le terme intégral correspond au fait qu'on ne peut rembourser une facture qu'entièrement : aussi, même si à la fin de notre exemple A est en bénéfice de 1€, il ne peut rembourser sa dette de 2€ à B. La balance de A reste donc de 1€, car il doit 2€ et dispose de 1€ en réserve (en bleu). On peut remarquer qu'avec les bénéfices et les dettes au centre de financement, la position nette est toujours respectée.

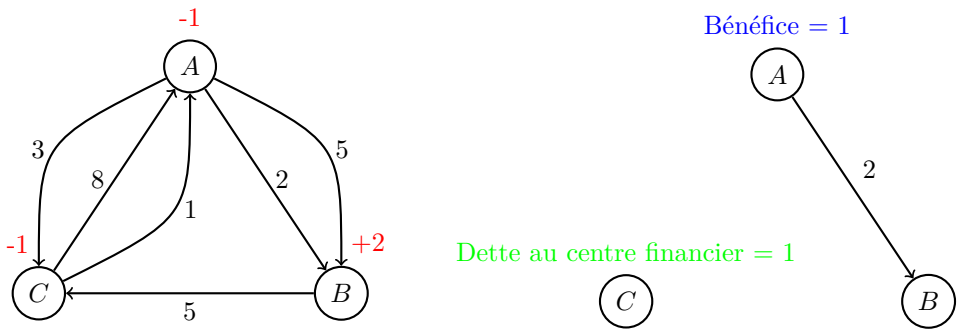


FIGURE 3 – Graphe après suppression intégrale de la dette

On remarque qu'avec cette seconde méthode, on peut éliminer toute la dette du graphe si on dispose d'une somme supérieure à la somme des position nette des entreprises globalement endettées. En effet, il suffit de financer à chaque entreprise se dette et ainsi tout peut être compensé.
En réalité, les multi-graphes étudiés ont des tailles, pour une étude sur un mois, de l'ordre de cent mille noeuds

(entreprises), vingt-huit millions d'arêtes (factures) et la somme totale circulant est de l'ordre de cent millions d'euros. Cette grande taille influe sur les algorithmes utilisés, pour que les calculs restent faisables [4]. La complexité est ainsi une composante majeure du problème de compensation de la dette, ce qui se fera ressentir lors de l'étude des algorithmes déjà disponibles.

1.3 Formalisation de la situation

Définition 1 (Multi-graphe de dettes)**.** On appelle *multi-graphe de dettes* un quadruplet $\mathcal{G} = (V, \mathcal{E}, y, c)$ avec

— V est l'ensemble des sommets, ils correspondent aux entreprises.

— $\mathcal{E}$ est un multi-ensemble de paires de V, soit un ensemble de paires de V dans lequel la même paire peut apparaître plusieurs fois. Chaque paire (i, j) correspond à une facture que i doit payer à j. L'utilisation d'un multi-ensemble s'explique par le fait que i peut avoir plusieurs factures non réglées à j.

— $y : \mathcal{E} \to \mathbb{R}_+^*$ est un fonction qui à chaque facture de $\mathcal{E}$ associe un entier positif, le montant de cette facture. On l'appelle fonction de somme.

— $c : \mathcal{E} \to \mathbb{R}_+^*$ est une fonction qui à chaque facture associe un poids positif, correspondant à l'importance qu'a cette facture. L'utilité de celui-ci peut s'expliquer par des factures n'ayant pas la même importance selon leur ancienneté, une facture ancienne devant être remboursée prioritairement par rapport à une facture plus récente.

L'ensemble des *débiteurs* de i est $V^-(i) = \{j \in V \,/\, (j, i) \in \mathcal{E}\}$, l'ensemble de ses *créanciers* est $V^+(i)\{j \in V \,/\, (i, j) \in \mathcal{E}\}$.

La *position nette* de i peut être définie par $b(i) = \sum_{(j,i)\in\mathcal{E}} y(j, i) - \sum_{(i,j)\in\mathcal{E}} y(i, j)$. Elle représente la somme algébrique totale due à i.

Par équité, $b(i)$ doit rester constante à tout instant de la compensation des dettes.

On va étudier deux modèles pour compenser les dettes, la réduction partielle de la dette (partie 2) et la suppression intégrale de la dette (partie 5), ainsi de divers outils utiles dans ces modèles, à savoir des algorithmes d'élimination des cycles dans un graphe (partie 3), de recherche des cycles dans un graphe (partie 4) et de détection de communautés dans un graphe (partie 6).

2 Premier modèle - Réduction partielle de dettes

Le modèle que nous étudions à été développé par Diamond et Dybvig en 1983 [1] puis par Allen et Gale en 2000 [5], qui ont commencés à introduire l'utilisation de graphes pour modéliser l'économie. Le formalisme de notre problème a néanmoins été introduit par Gazda, Horváth et Rešovský à partir des années 2000 [6, 7].

L'idée de base consiste à remarquer que si des entreprises $s_1, \cdots, s_p \in V$ ont des dettes sont telles que pour tout i, s_i doit $\alpha > 0$ à s_{i+1} et s_p doit α à s_1, alors cette dette α peut être annulée pour toutes les entreprises $s_1, \cdots, s_p$.

Pour modéliser la réduction partielle de la dette, on se ramène du cas d'un multi-graphe $\mathcal{G} = (V, \mathcal{E}, y, c)$ au cas d'un graphe $G = (V, E, \tilde{y}, \tilde{c})$ où :

— E est l'ensemble contenant tous les éléments du multi-ensemble $\mathcal{E}$, sans répétitions.

— Pour $e \in E$, $\tilde{y}(e) = \sum_{e' \text{ occurence de } e \text{ dans } \mathcal{E}} y(e')$ (la dette totale entre deux sommets est la somme des dettes entre ces deux sommets).

— Pour $e \in E$, $\tilde{c}(e) = \sum_{e' \text{ occurence de } e \text{ dans } \mathcal{E}} c(e')$.

y, c sont ainsi définies de manière unique pour chaque couple de sommet. On notera $y(i, j) = y_{ij}$, $c(i, j) = c_{ij}$.

On fait une autre simplification : on considère que deux entreprises ne peuvent pas avoir de dettes mutuelles.

Ainsi, si pour $i, j \in V$, on a $y_{ij} > 0$ et $y_{ji} > 0$, on remplace y_{ij} par $\max(0, y_{ij} - y_{ji})$ et y_{ji} par $\max(0, y_{ji} - y_{ij})$. Cela revient à compenser directement les dettes mutuelles entre i et j. Cette simplification réduit la somme maximale théorique pouvant être réduite, mais simplifie les algorithmes, et est en pratique efficace.

Définition 2. Le problème de réduction partielle de la dette consiste, étant donné un graphe de dettes $G = (V, E, y, c)$, à trouver un graphe $G' = (V, E, y', c)$ tel que :
- L'équité soit préservée entre G et G', autrement dit si on note b et b' les positions nettes dans G et G' respectivement, on a $b(e) = b'(e)$ pour tout $e \in E$.
- Pour tout $e \in E$, $y'(e) \leqslant y(e)$.
- $\sum_{e \in E} c(e)(y(e) - y'(e))$ (la somme réduite est maximale).

Définition 3 (Somme échangée pondérée)**.** On définit la *somme échangée pondérée* dans le graphe par $\sum_{e \in E} c(e)y(e)$.

Définition 4. On appelle *flot* (ou *circulation*) dans le graphe G une fonction $x : E \to \mathbb{R}_+^*$ telle que :
- Pour tout $e \in E$, $0 \leqslant x(e) \leqslant y(e)$.
- Pour tout $i \in V$, $\sum_{j \in V^-(i)} x_{ji} = \sum_{j \in V^+(i)} x_{ij}$ (préservation de l'équité ou conservation du flot).

Remarque. Dans la littérature, le terme de flot est le plus souvent utilisé dans une situation avec une source et un puits. Ce n'est pas le cas ici, car toutes les entreprises ont un rôle similaire.

Notre problème d'optimisation implique donc de trouver un flot pondéré maximal, c'est-à-dire tel que

$$\sum_{e \in E} c(e)x(e) = \max_{\tilde{x} \text{ flot sur } G} \sum_{e \in E} c(e)\tilde{x}(e).$$

La fonction y' est alors donnée par $y' = y - x$.

Soit maintenant $C \subset E$ un cycle orienté du graphe des dettes. On note $y(C)$ la capacité du cycle, définie par $y(C) = \min_{e \in C} y(e)$. Alors si on compense la dette des entreprises dans ce cycle de $y(C)$, l'équité reste vérifiée. Plus précisemment, si on pose

$$\begin{aligned} \tilde{y} : E &\to \mathbb{R}_+ \\ e &\mapsto y(e) - y(C) \text{ si } e \in C \\ e &\mapsto y(e) \text{ sinon.} \end{aligned}$$

on a un nouveau graphe des dettes vérifiant pour tout $i \in V$, $\tilde{b}(i) = b(i)$. On a mutualisé les dettes des entreprises dans le cycle C. Le cycle C est alors éliminé, son arête au poids le plus faible ayant maintenant un poids nul. Partant de ce résultat, on va chercher à éliminer successivement les cycles du graphe des dettes, pour réduire les flux d'argent réels. On cherche ainsi à éliminer des cycles de manière à minimiser la somme échangée pondérée. On doit donc résoudre un problème d'optimisation sur un graphe.

Définition 5. On appelle *flot cyclique* sur G un flot x tel que x est nul partout sauf sur un cycle $C = (i_1, \cdots, i_k, i_1)$ de G avec les (i_j) distincts.

Lemme 1 (Busacker & Saaty,1965)**.** *Tout flot dans un graphe $G = (V, E, y)$ peut être décomposé en une somme finie de flots cycliques. De plus, le nombre de flots cycliques dans cette somme est majoré par le nombre d'arêtes non-nulles du flot $|\{e \in E \,/\, x(e) \neq 0\}|$.*

Démonstration. On prend un flot x sur G. On travaille par récurrence sur $m = |\{e \in E \,/\, x(e) > 0\}| \in \mathbb{N}$ le nombre d'arêtes non-nulles du flot.
<u>Initialisation</u> : $m = 0$.

Alors x est nul et la somme vide de flots convient.
<u>Hérédité</u> : Soit $m \in \mathbb{N}^*$. On suppose le résultat vrai pour tout entier strictement inférieur à m.
On note $A = \{e \in E \,/\, x(e) \neq 0\}$ l'ensemble des arêtes non-nulles du flot et on suppose $|A| = m > 0$.
Montrons qu'on peut construire un cycle dans A. x est non nul donc il existe $i_1, i_2 \in V$ tel que $x(i_1 i_2) > 0$. Par conservation du flot x, il existe $i_3 \in V$ tel que $x(i_2 i_3) > 0$. Si $i_1 = i_3$, alors (i_1, i_2, i_3) est un cycle. Sinon on itère le raisonnement. Comme V est fini, il existe $k, k' \in \mathbb{N}^*$ tels que $k < k'$ et $i_k = i_{k'}$. Alors $C = (i_k, \cdots, i_{k'})$ est un cycle dans A.
Posons alors $\delta = \min\limits_{e \in C} x(e) > 0$ et $e_{\min}$ tel que $x(e_{\min}) = \delta$. On définit le flot cyclique x_C sur C par

$$x_C(e) = \begin{cases} \delta \text{ si } e \in C \\ 0 \text{ sinon.} \end{cases}$$

On note $x' = x - x_C$. Alors x' est toujours un flot sur G et $x'(e_{\min}) = 0$. Ainsi $|\{e \in E \,/\, x(e) \neq 0\}| < m$ donc par hypothèse de récurrence, il existe $p \leqslant m - 1$ et $x_1, \cdots, x_p$ des flots cycliques tels que $x' = \sum\limits_{1 \leqslant i \leqslant p} x_i$. Alors $x = x_C + \sum\limits_{1 \leqslant i \leqslant p} x_i$, et $p + 1 \leqslant m$ ce qui conclut la récurrence.

Remarque. Un flot peut ainsi être vu comme une superposition de cycles de dettes, qui peuvent être compensés les uns après les autres.

Ainsi, soustraire successivement des cycles un à un revient formellement à trouver un flot, qui sera la somme des cycles soustraits, que l'on retire au graphe de dettes. Le graphe résiduel $(V, E, y - x)$ minimise les échanges d'argent tout en maintenant l'équité.

3 Élimination des cycles

Pour éliminer les cycles dans le graphe des dettes, on va utiliser un algorithme proposé par Klein [8] en 1967. Celui-ci a été repris plus tard dans les travaux de Goldberg et Tarjan [9] (1989). Si la recherche de cycles à éliminer peut sembler proche de l'algorithme de Ford-Fulkerson [10] (1962) dont Klein déclare s'inspirer, la grande différence réside dans le fait qu'ici il n'y a ni source ni puits.
Une idée naïve pour résoudre le problème posé serait d'éliminer les cycles tant qu'il y en a, et de s'arrêter quand ce n'est plus possible. Le résultat obtenu serait cependant un maximum local sur les flots, mais pas global, ce qui ne résoudrait pas le problème. L'algorithme présenté va transformer cette résolution d'un problème de maximum local pour résoudre un problème global.

3.1 Algorithme

Pour construire un tel flot de poids maximal, on va utiliser l'algorithme d'élimination des cycles de Klein (Alg. 1). Celui-ci construit un flot de poids maximal de manière itérative en partant du flot nul et en augmentant son poids par ajout de flots cycliques.

Définition 6. Pour une arête $e = (i, j) \in E$, on note $e^{\mathcal{R}} = (j, i)$ l'arête inverse de e. On note $E^{\mathcal{R}} = \{e^{\mathcal{R}} \,/\, e \in E\}$ l'ensemble des arêtes inverses.
Pour un flot x, on définit le *graphe résiduel* de x dans G par : $G_x = (V, E_x, y_x, c_x)$, avec :

$$E_x = \{(i, j) \in E \,/\, x_{ij} < y_{ij}\} \cup \{(i, j) \in E^{\mathcal{R}} \,/\, x_{ji} > 0\}$$

$$y_x : E_x \to \mathbb{R} \qquad\qquad c_x : E_x \to \mathbb{R}$$

$$e \mapsto y(e) - x(e) > 0 \text{ si } e \in E \qquad\qquad e \mapsto c(e) \text{ si } e \in E$$

$$e \mapsto x(e^{\mathcal{R}}) > 0 \text{ si } e \in E^{\mathcal{R}}. \qquad\qquad e \mapsto -c(e^{\mathcal{R}}) \text{ si } e \in E^{\mathcal{R}}.$$

Algorithme 1 : Algorithme de Klein

Entrées : $G = (V, E, y, c)$
Résultat : Flot maximal x
Initialisation : $x = 0$, $G_x = G$;
Tant que G_x contient un cycle C de poids strictement positif **faire**
  $\delta = \min_{e \in C} x(e) > 0$;
  Pour e parcourant C **faire**
    si $e \in E$ **alors**
      $x(e) \leftarrow x(e) + \delta$
    sinon
      $x(e^{\mathcal{R}}) \leftarrow x(e^{\mathcal{R}}) - \delta$
    fin
  fin
  Mettre à jour E_x et y_x
fin

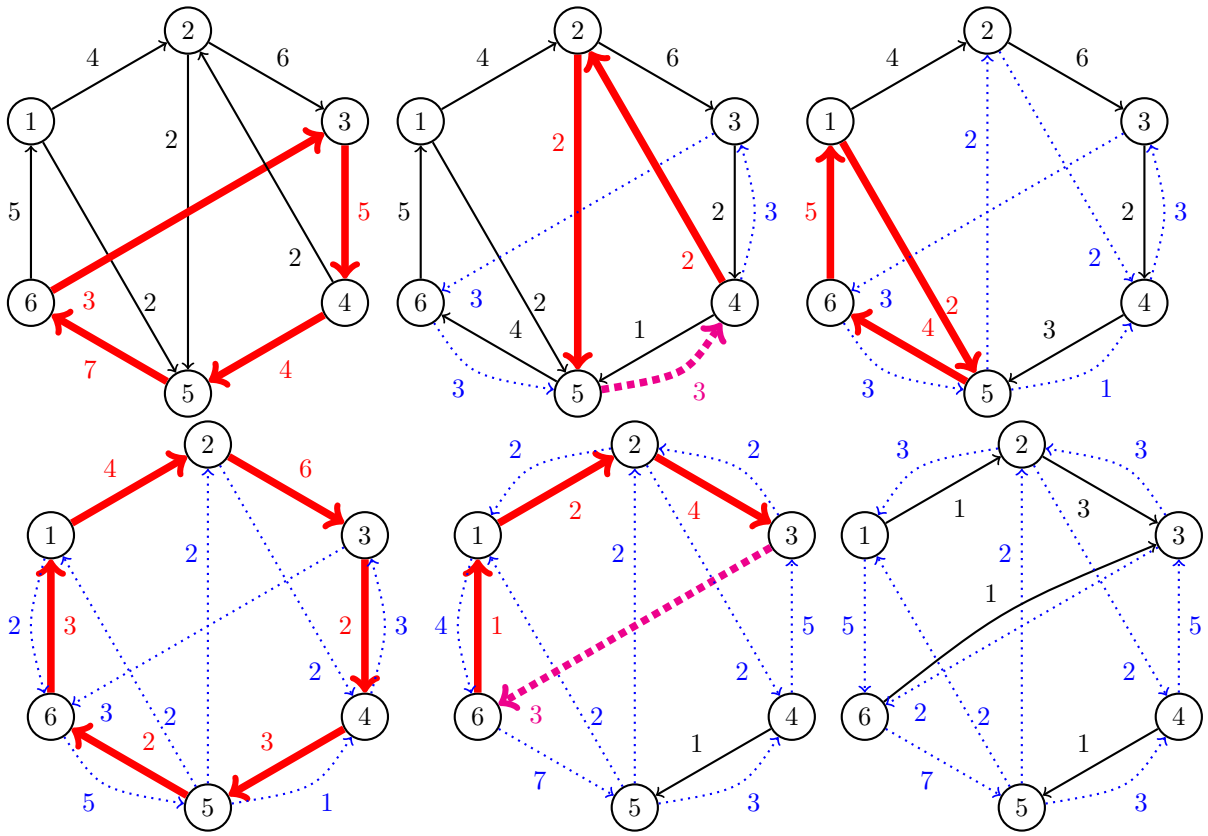


FIGURE 4 – Application de l'algorithme de Klein

E_x correspond à l'ensemble des arêtes directe qui ne sont pas complètement réduites et des arêts indirectes dont

l'arête directe a été partiellement réduite.

Comme dans l'algorithme de Ford-Fulkerson, on ajoute au graphe des arêtes inverses pour se ramener d'un problème de minimum local (plus de cycles dans le graphe) à un problème de minimum global (flot minimal). Le fait d'ajouter ces arêtes inverses, crée ainsi des cycles de poids négatif, car les arêtes inverses ont des poids négatifs. On ne faire ciculer du flot que le long des cycles de poids positif, ce qui permet d'augmenter la valeur du flot, quand bien même il n'y aurait plus de cycle dans le graphe originel. L'utilisation de ces arêtes apparaîtra clairement dans la preuve de correction. La valeur de x est destinée à changer pour construire un flot de poids maximal. Un cycle C est dit de poids strictement positif si $\sum_{e \in C} c_x(e) > 0$. La recherche de cycles fera l'objet de la partie 4.

Exemple. On applique l'algorithme d'élimination des cycles au graphe ci-dessous. Les arêtes directes sont indiquées en noir et plein, les inverses en bleu et pointillé. Le cycle sélectionné pour la réduction est indiqué en rouge gras plein pour une arête directe et en magenta gras pointillé pour une arête inverse.

Remarque. Dans l'article de Klein [8], il y a une source et un puits et l'objectif est de trouver un flot maximal de poids minimal. L'algorithme 1 est une adaptation, mais on peut se ramener à l'article de Klein de la manière suivante : on rajoute une source et un puits déconnectés du reste du graphe, et on cherche des cycles de poids positifs au lieu de négatifs de manière à maximiser le flot au lieu de le minimiser. Le flot maximal est alors nul, donc la recherche d'un flot maximal de poids maximal correspond simplement à la recherche d'un flot de poids maximal, soit notre objectif.

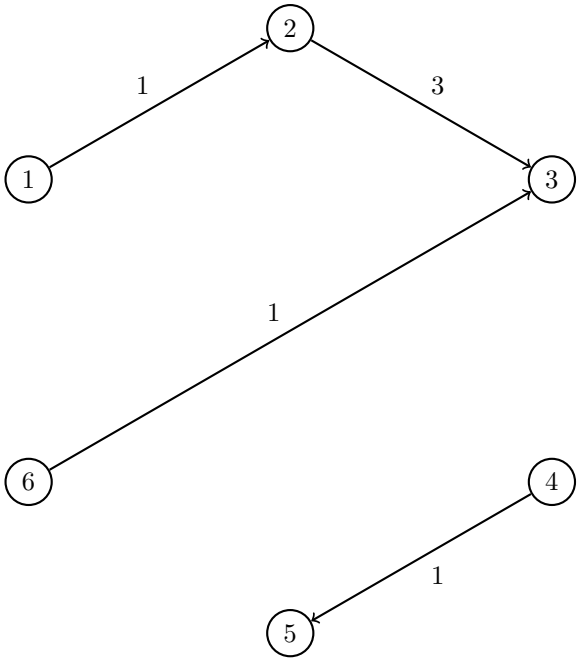


FIGURE 5 – Graphe résiduel

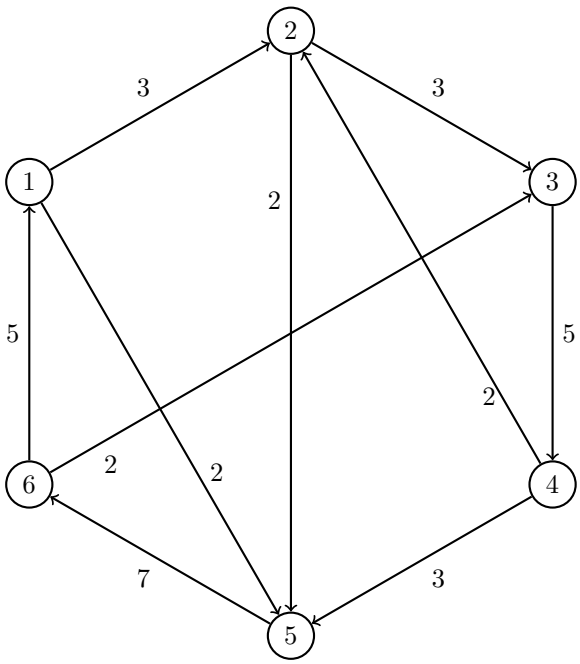


FIGURE 6 – Flot maximal

Remarque. La recherche de cycles peut se faire à partir de chaque sommet, en appliquant l'algorithme de Ford-Fulkerson avec une source et un puits identique pour chacun des sommets du graphe, jusqu'à ce que plus aucune amélioration du flot maximal ne soit possible. Lors de l'application de Ford-Fulkerson au niveau d'un sommet, il faut cependant modifier le critère lorsqu'on augmente le flot le long d'un chemin : non seulement le flot passant par le sommet étudié doit augmenter, mais aussi le flot global.

3.2 Correction de l'algorithme

Remarque. Si l'algorithme de Klein est proche de celui de Ford-Fulkerson, la démonstration ici présentée est très différente car elle ne s'appuie pas sur un problème dual, comme le problème MIN-CUT dans le cas de Ford-Fulkerson. La recherche d'un problème dual à celui du flot maximal pourra faire l'objet de futures recherches.

Remarque. Comme un cycle a même valeur de y en toutes les arêtes qui le composent, le signe du poids d'un cycle est strictement le même que celui de la somme des poids des arêtes qui le compose.

Remarque. Si $x = \sum_{1 \leqslant i \leqslant p} x_i$ alors $c(x) = \sum_{1 \leqslant i \leqslant p} c(x_i)$.

Théorème 1 (Busacker & Saaty,1965)**.** *Un flot x sur un graphe $G = (V, E, y)$ est de poids maximal si et seulement s'il n'existe aucun cycle de poids strictement positif dans G_x.*

Démonstration. $\boxed{\Rightarrow}$ Par contraposée, on suppose qu'il existe un cycle C de poids strictement positif dans G_x.
Alors on pose $\delta = \min_{e \in C} x_x(e) > 0$ et

$$\begin{aligned} x' : E &\to \mathbb{R}_+ \\ e &\mapsto x(e) + \delta \text{ si } e \in C \\ e &\mapsto x(e) - \delta \text{ si } e \in C^{\mathcal{R}} \\ e &\mapsto x(e) \text{ si } e \notin C \cap C^{\mathcal{R}}. \end{aligned}$$

On a

$$\begin{aligned} c(x') &= \sum_{e \in E} x'(e)c(e) \\ &= \sum_{e \in E \cap C} (x(e) + \delta)c(e) + \sum_{e \in E \cap C^{\mathcal{R}}} (x(e) - \delta)c(e) + \sum_{e \in E \setminus (C \cap C^{\mathcal{R}})} x(e)c(e) \\ &= c(x) + \delta \sum_{e \in C} c_x(e) \ > \ c(x). \end{aligned}$$

Ainsi x n'est pas maximal.
$\boxed{\Leftarrow}$ Par contraposée, on suppose que x n'est pas de poids maximal. Soit x^* un flot de poids maximal.
On définit $A \subset E_x$ par $A_x = A_E \cup A_{E^{\mathcal{R}}}$ avec $A_E = \{e \in E \cap E_x \,/\, x^*(e) > x(e)\}$ et $A_{E^{\mathcal{R}}} = \{e \in E^{\mathcal{R}} \cap E_x \,/\, x^*(e^{\mathcal{R}}) < x(e^{\mathcal{R}})\}$.
On définit

$$\begin{aligned} x' : A &\to \mathbb{R}_+^* \\ e &\mapsto x^*(e) - x(e) \text{ si } e \in A_E \\ e &\mapsto x(e^{\mathcal{R}}) - x^*(e^{\mathcal{R}}) \text{ si } e \in A_{E^{\mathcal{R}}}. \end{aligned}$$

On remarque que

$$\begin{aligned} A_E \cup A_{E^{\mathcal{R}}}^{\mathcal{R}} &= \{e \in E \,/\, x(e) < y(e) \text{ et } x(e) < x^*(e)\} \cup \{e \in E^{\mathcal{R}} \,/\, x(e^{\mathcal{R}}) > 0 \text{ et } x(e^{\mathcal{R}}) > x^*(e^{\mathcal{R}})\}^{\mathcal{R}} \\ &= \{e \in E \,/\, x(e) < x^*(e)\} \cup \{e \in E^{\mathcal{R}} \,/\, x(e^{\mathcal{R}}) > x^*(e^{\mathcal{R}})\}^{\mathcal{R}} \\ &= \{e \in E \,/\, x(e) < x^*(e)\} \cup \{e \in E \,/\, x(e) > x^*(e)\} \\ &= \{e \in E \,/\, x(e) \neq x^*(e)\}. \end{aligned}$$

Pour $i \in V$, on a donc :

$$\begin{aligned}
\sum_{(j,i)\in A} x'_{ji} - \sum_{(i,j)\in A} x'_{ij} &= \sum_{(j,i)\in A_E} (x^*_{ji} - x_{ji}) + \sum_{(j,i)\in A_{E^{\mathcal{R}}}} (x_{ij} - x^*_{ij}) - \sum_{(i,j)\in A_E} (x^*_{ij} - x_{ij}) - \sum_{(i,j)\in A_{E^{\mathcal{R}}}} (x_{ji} - x^*_{ji}) \\
&= \sum_{(j,i)\in A_E} (x^*_{ji} - x_{ji}) + \sum_{(j,i)\in A^{\mathcal{R}}_{E^{\mathcal{R}}}} (x_{ij} - x^*_{ij}) - \sum_{(i,j)\in A_E} (x^*_{ij} - x_{ij}) - \sum_{(j,i)\in A^{\mathcal{R}}_{E^{\mathcal{R}}}} (x_{ji} - x^*_{ji}) \\
&= \sum_{(j,i)\in A_E \cup A^{\mathcal{R}}_{E^{\mathcal{R}}}} (x^*_{ji} - x_{ji}) - \sum_{(i,j)\in A_E \cup A^{\mathcal{R}}_{E^{\mathcal{R}}}} (x^*_{ij} - x_{ij}) \\
&= \sum_{(j,i)\in E} (x^*_{ji} - x_{ji}) - \sum_{(i,j)\in E} (x^*_{ij} - x_{ij}) \qquad \text{(On ajoute les termes manquants, qui sont nuls)} \\
&= 0 \qquad \text{par conservation des flots } x \text{ et } x^*.
\end{aligned}$$

Ainsi x' est un flot sur (V, A) (la capacité n'a pas d'importance ici).
Il se décompose donc en une somme de cycles $x' = \sum_{1\leqslant i\leqslant p} c_i$. Ainsi $c(x') = \sum_{1\leqslant i\leqslant p} c(c_i)$.
On a de plus

$$\begin{aligned}
c(x') &= \sum_{e\in A} x'(e)c_x(e) \\
&= \sum_{e\in A_E} (x^*(e) - x(e))c(e) + \sum_{e\in A_{E^{\mathcal{R}}}} \left(x(e^{\mathcal{R}}) - x^*(e^{\mathcal{R}})\right)\left(-c(e^{\mathcal{R}})\right) \\
&= \sum_{e\in A_E \cup A^{\mathcal{R}}_{E^{\mathcal{R}}}} (x^*(e) - x(e))c(e) \\
&= \sum_{e\in E} (x^*(e) - x(e))c(e) \\
&= c(x^*) - c(x) > 0.
\end{aligned}$$

Ainsi il existe $i \in [\![1, p]\!]$ tel que $c(c_i) > 0$, et le cycle c_i est de poids strictement positif.

Théorème 2 (Klein,1967)**.** *L'algorithme d'élimination des cycles de Klein termine et si on note x^* la valeur de x à la fin de l'algorithme, alors :*

$$e \in E \mapsto x^*(e) \text{ est un flot pondéré maximal sur } G.$$

Démonstration. Le flot renvoyé par l'algorithme est tel que son graphe résiduel ne contient pas de cycle de poids strictement positif, donc il est maximal d'après le théorème précédent.

Remarque. L'algorithme présenté est polynomial en la taille du graphe, ce qui pourrait le rendre accessible. Néanmoins, la grande taille des graphes rend impossible son utilisation tel quel, et les simplifications du modèle présentées pour appliquer les algorithmes peuvent ne pas être satisfaisante, ce qui explique la recherche d'autres pistes pour la compensation mutuelles des dettes et la tentative de découper le graphe en sous-graphes (voir partie 6 sur le détection de communautés).

4 Recherche de cycles

L'utilisation de l'algorithme de Klein nécessite de chercher des cycles dans le graphe. Nous allons donc étudier quelques algorithmes de recherche de cycles. Plusieurs algorithmes existent pour rechercher les cycles d'un graphe, de manière exhaustive ou non. L'un des plus anciens est celui de Bellman [11] (1958), qui permet en cherchant les plus courts chemins dans un graphe orienté pondéré de détecter si un cycle de poids positif existe. Plusieurs algorithmes de recherche exhaustive des cycles ont été développés dans les années 1970, par

Algorithme 2 : Algorithme de Bellman-Ford adapté

Entrées : $G = (V, E, c)$ ($E \subset V \times V$) un graphe muni d'une fonction de poids c.
Résultat : Un cycle dans G de poids positif s'il existe, rien sinon
Initialisation : $D_0(s) = 0$ et pour tout $v \in V \backslash \{s\}$ $D_0(v) = -\infty$. Pour tout $v \in V$, initialement $P_k(v)$=`nil`;
Pour k allant de 1 à n **faire**
  $\forall v \in V$, $D_k(v) \leftarrow \max \begin{cases} D_{k-1}(v) \\ \max\limits_{(u,v)\in E} (D_{k-1}(u) + c(u,v)) \end{cases}$;
  si $D_k(v) = D_{k-1}(v)$ **alors**
    $P_k(v) \leftarrow P_{k-1}(v)$
  sinon
    Soit $u \in V$ tel que $D_k(v) = D_{k-1}(u) + c(u,v)$;
    $P_k(v) \leftarrow u$
  fin
fin
si Il existe $v \in V$ tel que $D_n(v) > D_{n-1}(v)$ **alors**
  Poser $v_n := v$;
  Pour k allant de $n-1$ à 0 **faire**
    Poser $v_k := P_n(v_{k+1})$;
    Poser $j = \min\{k \in [\![1, n]\!] \,/\, v_j = v_0\}$;
    Renvoyer le cycle $(v_0, \cdots, v_j)$
  fin
sinon
  Renvoyer `nil` \\ Il n'existe pas de cycles de poids positif dans G
fin

Tiernan [12] (1970), Weinblatt [13] (1972), Tarjan [14, 15] (1972-1973) et enfin Johnson [16] (1975), avec des améliorations progressives de la complexité temporelle. En général, le nombre de cycles d'un graphe explose avec le nombre d'arêtes, ce qui rend la recherche exhaustive de cycles trop coûteuse sur de grands graphes, mais celle-ci peut cependant être utile dans des graphes de petite taille.

4.1 Algorithme de Bellman-Ford

L'algorithme de Bellman-Ford (Alg. 2) cherche le chemin de poids le plus important entre un sommet s fixé et tous les autres sommets du graphe. Dans notre cas, le choix de s importe peu, puisqu'on va se servir de Bellman-Ford pour rechercher un cycle de poids strictement positif. Pour ce faire, on note que l'algorithme de Bellmen-Ford classique dans un graphe termine en n itérations dans un graphe à n sommets sans cycles de poids positif, et ne termine pas s'il y a un cycle de poids positif. Ainsi, si le résultat renvoyé change entre les itérations n et $n+1$, alors il y a un cycle de poids positif, et cette condition est nécessaire.
Pour $k \in \mathbb{N}$, $v \in V$, on note ainsi $D_k(v)$ le poids maximum d'un chemin de s à v contenant au plus k arêtes. S'il n'existe pas de chemin de s à v avec au plus k arêtes, on note $D_k(v) = -\infty$. On note de plus $P_k(v)$ le prédécesseur de v dans le chemin qui relie s à v de poids minimal avec au plus k arêtes (cette valeur n'est pas définie si un tel chemin n'existe pas).

Théorème 3 (Bellman,1958)**.** *L'algorithme de Bellman-Ford renvoie un cycle de poids strictement positif s'il existe, et indique qu'il n'en existe pas sinon. Sa complexité temporelle est $\mathcal{O}(|E||V|)$ et sa complexité spatiale est $\mathcal{O}(|V|)$ (en ne gardant que les valeurs les plus récentes dans les tables) (Bellman, 1958 [11]).*

Algorithme 3 : énumération_cycle (algorithme de Johnson)

Données : $A_K = (V_K, E_K)$ composante connexe étudiée (graphe orienté), $B = (V_B, E_B)$ graphe orienté marquant le cycle en cours de visite, *blocked* tableau booléen de taille n, $s \in V_K$, *pile* une pile de sommets, L_C une liste de cycles du graphe

Entrées : $v \in V_K$ à partir duquel on cherche les cycles

Résultat : Booléen f indiquant si un cycle a été trouvé

Initialisation : $f \leftarrow$`false`;

Empiler v;

blocked$[v] \leftarrow$`true`;

Pour (v, w) parcourant E_K **faire**

  si $w = s$ **alors**

    On a trouvé un cycle;

    Ajouter à L_C le cycle composé de *pile* suivi de s \\ Effet de bord sur L_C;

    $f \leftarrow$`true`

  sinon

    On applique récursivement l'algorithme;

    si $\neg$*blocked*$[w]$ et *cycle*(w) **alors** $f \leftarrow$`true`;

  fin

fin

si f **alors**

  débloquer(w)

sinon

  Pour $(v, w) \in E_K$ **faire**

    si $(w, v) \notin E_B$ **alors** ajouter (w, v) à E_B;

  fin

fin

Dépiler v

4.2 Algorithme de Johnson

L'algorithme de Johnson (Alg. 3-4-5) fonctionne sur un principe simple : il consiste à parcourir chaque sommet et à noter les cycles passant par ce sommet. Cependant, lors de la recherche de cycles à partir d'un sommet, on ne s'intéresse qu'à la composante fortement connexe du graphe induit par les sommets non visités. Ainsi, comme on a déjà trouvé tous les cycles passant par les sommets visités, on réduit le nombre de répétitions. Pour trouver les composantes fortement connexes, on peut utiliser des algorithmes comme ceux de Tarjan ou Kosaraju. De plus, pour éviter les répétitions, on marque les sommets du cycle où l'on ne doit plus passer dans un tableau booléen *blocked* et on retient le cycle en cours de parcours dans B. Ainsi, on évite de passer deux fois par un même cycle lors de la visite d'un même sommet, ce qui crée un grand gain de temps par rapport aux algorithmes précédents.

Théorème 4 (Johnson,1975)**.** *L'algorithme de Johnson renvoie tous les cycles du graphe passé en argument. Pour un graphe $G = (V, E)$ on note n le nombre de sommets, e le nombre d'arêtes et C le nombre de cycles dans G. La complexité temporelle de l'algorithme de Johnson est en $\mathcal{O}((n+e)(C+1))$ et sa complexité spatiale supplémentaire est en $\mathcal{O}(n+e)$.*

Remarque. On parle de complexité spatiale supplémentaire pour ne considérer que les données intermédiaires nécessaires au calcul : la prise en compte de la taille des entrées et des sorties serait peu significative dans l'étude de la complexité d'un algorithme, car celles-ci sont indépendantes de l'algorithme.

Remarque. Bien que l'un des algorithmes les plus efficaces à ce jour, dans le cas de graphes de dettes l'algorithme

Algorithme 4 : débloquer (algorithme de Johnson)

Données : $B = (V_B, E_B)$ graphe orienté, *blocked* tableau booléen de taille n indiquant les sommets bloqués

Entrées : $u \in V_B$

Résultat : Débloque récursivement tous les sommets accessibles depuis u

blocked$[u]$ ←`false`;

Pour (u, w) parcourant E_B **faire**

  Enlever (u, w) de E_B;

  si *blocked*$[w]$ **alors** débloquer(w);

fin

Algorithme 5 : Algorithme de Johnson

Données : $G = (V, E)$ graphe orienté, $n = |V|$

Résultat : L_C liste de tous les cycles de G

Initialisation : *pile* $\leftarrow \emptyset$, $s \leftarrow 1$;

Tant que $s < n$ **faire**

  $A_K \leftarrow$ composante fortement connexe de s dans le graphe induit par $\{s, s+1, \cdots, n\}$ dans G.;

  si $V_K \neq \{s\}$ **alors**

    Pour i parcourant V_K **faire**

      blocked$[i]$ ←`false`

    fin

    $B \leftarrow$ graphe vide d'ensemble de sommets V_K;

    énumération_cycle(s)

  fin

  $s \leftarrow s + 1$

fin

de Johnson reste inutilisable : les temps de calcul seraient trop élévés. De plus, l'élimination des cycles d'un graphe comme décrit précédemment ne nécessite pas de connaître tous les cycles d'un graphe, juste certains (ceux qu'on élimine). Pour résoudre ces problèmes de complexité, on peut envisager plusieurs solutions. On peut tout d'abord rechercher un nombre fixé de cycles grâce à cet algorithme, et s'arrêter lorsqu'on atteint ce nombre, de manière à ne pas faire trop de calculs inutiles. On peut aussi découper le graphe en plus petits sous-graphes, auxquels on appliquerait l'algorithme. C'est ce qui sera fait dans la partie 6 sur la détection de communautés.

Exemple. Voir Annexe.

5 Second modèle - Suppression intégrale de dettes

La suppression intégrale de dette est une manière nouvelle d'étudier le problème des dettes entre entreprises, et il y a aujourd'hui peu de documentation sur le sujet. On peut citer un article de Pătcaş [17] en 2011, qui ne correspond pas exactement au problème que nous étudions mais qui s'en rapproche. En effet, là où Pătcaş s'autorise de rajouter des arêtes entre des sommets qui n'étaient pas reliés au départ, de manière à simplifier le graphe de dettes, nous nous interdisons cette simplification pour des motifs économiques : en effet, il est impossible de demander à une entreprise de rembourser une autre si les deux n'ont jamais eu de contact entre elles.

Définition 7. Le problème de suppression intégrale de dettes consiste, étant donné un multi-graphe de dettes pondéré, à financer des sommets endettés (dont la balance est négative) de manière à supprimer une partie de la dette du graphe. Contrairement au problème de réduction de la dette, *on ne peut supprimer une arête du graphe qu'entièrement*, d'où la dénomination *intégrale*. Ainsi le but est de trouver où injecter une somme d'argent, souvent limitée, de manière à maximiser la dette pondérée supprimée. On remarque que dans cette situation, il est important de conserver un multi-graphe.

5.1 Formalisation du problème

Définition 8 (SID)**.** On se donne un multi-graphe pondéré $G = (V, E, y, c)$ et une somme maximale à injecter $Q > 0$. On cherche à trouver un ensemble d'arêtes à supprimer $S \subset E$ vérifiant :

— Si on note $I(v) = \sum_{(v,w)\in S} y(v,w) - \sum_{(w,v)\in S} y(w,v)$ la somme algébrique injectée en v (celle-ci est positive si on doit financer v et négative sinon), on doit avoir $\sum_{v\in V} I(v)^+ \leqslant Q$, où x^+ désigne la partie positive de x (on doit injecter au plus Q).

— $\sum_{e\in S} c(e)y(e)$ (la quantité pondérée de dette supprimée) est maximale.

5.2 Complexité

Le problème SID a été démontré comme NP-dur par Güntzer en 1998 [18] puis en 2017 par 3 chercheurs de l'université de Zurich [19]. Une manière élégante de le montrer est de réduire le problème KNAPSACK, qui est NP-complet, au problème SID.

Définition 9 (KNAPSACK)**.** Une instance du problème KNAPSACK est définie par $(K, (m_i)_{1\leqslant i\leqslant n}, (v_i)_{1\leqslant i\leqslant n}$ du problème KNAPSACK, où $K \geqslant 0$ est la capacité du sac, et où les items 1 à n sont de masse $m_i \geqslant 0$ et de valeur $v_i \geqslant 0$. On cherche à trouver un sous-ensemble $J \subset \{1, \cdots, n\}$ tel que $\sum_{j\in J} m_j \leqslant K$ et tel que $\sum_{j\in J} v_j$ soit maximale.

Théorème 5. *Le problème* KNAPSACK *est NP-complet.*

Corollaire 1. *Le problème* SID *est NP-complet.*

Démonstration. On se donne une instance $(K, (m_i)_{1\leqslant i\leqslant n}, (v_i)_{1\leqslant i\leqslant n})$ du problème KNAPSACK, où K est la capacité du sac, et où les items 1 à n sont de masse m_i et de valeur v_i. Quitte à ajouter les éléments de poids nul à la fin, on suppose que tous les poids sont strictement positifs. On définit le graphe de dettes $G = (V, E, y, c)$ où

— $V = \{A, B\}$ (le graphe a deux sommets),

— $E = \{(A, B)_i \backslash 1 \leqslant i \leqslant n\}$ (il y a n factures de A à B),

— $y((A, B)_i) = m_i$,

— $c((A, B)_i) = \dfrac{v_i}{m_i}$.

On pose de plus $Q = K$.

Alors résoudre le problème SID sur (G, Q) revient à trouver $J \subset \{1, \cdots, n\}$ tel que $\sum_{j\in J} y((A, B)_i) \leqslant Q$ et $\sum_{j\in J} c((A, B)_i)y((A, B)_i)$ soit maximale, soit $\sum_{j\in J} m_i \leqslant K$ et $\sum_{j\in J} v_i$ maximale. On a ainsi ramené le problème KNAPSACK au problème SID, ce qui montre que ce dernier est NP-complet.

Remarque. Cette complexité implique la recherche d'heuristiques pour résoudre le problème, générales ou particulières, de manière à approximer une solution.

Remarque. Le problème SID fera l'objet du stage lors de l'été 2021.

Algorithme 6 : choisir_sommets (algorithme de Stoer-Wagner)

Entrées : $G = (V, E, c)$, $|V| > 1$, $a \in V$
Résultat : $s, t \in V$, ω le poids de la coupe $(V \backslash \{t\}, \{t\})$
Initialisation : $S_1 = \{a\}$;
si $|V| = 2$ **alors**
  $s = a$, $t \in V \backslash \{a\}$, $\omega = c_{st}$
sinon
  Tant que $|S_1| < |S| - 2$ **faire**
    $b^* \in \operatorname{argmax}_{b \in S \backslash S_1} \sum_{i \in V_1,\, (b,i) \in E} c_{bi}$ \\ on prend le sommet qui est le plus connecté à l'ensemble des sommets déjà choisis;
    $S_1 \leftarrow S_1 \cup \{b\}$
  fin
  $s \in \operatorname{argmax}_{b \in S_1} \sum_{i \in V_1,\, (b,i) \in E} c_{bi}$;
  $t \in S_1 \backslash \{b\}$;
  $\omega = \sum_{i \in V_1,\, (t,i) \in E} c_{ti}$ \\ la valeur de la coupe est le poids des arêtes entre le dernier sommet restant et les autres sommets choisis
fin

6 Détection de communautés

Les algorithmes exposés précédemment peuvent être coûteux en temps et en espace, car les graphes étudiés peuvent être très grands. Aussi, une idée naturelle est de réduire la taille des graphes étudiés, en étudiant des sous-graphes du graphe principal, qui aurait des propriétés permettant, sinon de maximiser, de supprimer une partie significative de la dette par rapport à la dette maximale supprimable. On risque de perdre en optimalité au recollement, car réunir les traitements individuels de manière optimale reviendrait à résoudre le problème général, mais on gagne, on va le voir, énormément en temps de calcul.

6.1 Algorithme de Stoer-Wagner

La recherche d'une coupe minimale est une manière de créer deux communautés, une de chaque côté de la coupe. Historiquement, le premier algorithme qui trouve cette coupe minimale est celui de Ford et Fulkerson, avec le théorème de MAX-FLOW-MIN-CUT [20]. En 1992, Hao et Orlin ont amélioré cet algorithme pour obtenir une meilleure complexité en 1992 [21]. On peut aussi citer un algorithme stochastique de Karger en 1999 [22].
L'algorithme de Stoer-Wagner [23] (1997) trouve une coupe minimale dans un graphe pondéré non-orienté (Alg. 6-7-8) avec un algorithme qui n'utilise pas de flots, ce qui le rend plus efficace. Soit $G = (V, E, c)$ un graphe non-orienté avec $c : E \to \mathbb{R}_+$ une fonction de poids. L'algorithme de Stoer-Wagner calcule une partition $V = V_1 \cup V_2$ de V qui minimise le poids des arêts coupantes $\sum_{i \in V_1,\, j \in V_2,\, (i,j) \in E} c_{ij}$.
Pour cela, on itère une fonction qui choisit deux sommets et les fusionne à partir d'un sommet fixé au départ.

Exemple. Voir Annexe.

Théorème 6 (Stoer-Wagner,1997)**.** *La coupe renvoyée par l'algorithme de Stoer-Wagner est minimale.*
La complexité temporelle de l'algorithme de Stoer-Wagner est $\mathcal{O}(|V|^2|E|)$. Sa complexité spatiale supplémentaire

Algorithme 7 : fusionner_sommets (algorithme de Stoer-Wagner)

Entrées : $G = (V, E, c)$, $|V| > 1$, $s \neq t \in V$
$V \leftarrow V \cup \{s \cup t\} \backslash \{s, t\}$;
Pour a parcourant $V \backslash \{s \cup t\}$ **faire**
  si $(a, s) \in E$ *ou* $(a, t) \in E$ **alors**
    $E \leftarrow E \cup \{(a, s \cup t)\} \backslash \{(a, s), (a, t)\}$ \\ les deux derniers sommets choisis sont fusionnés en un seul;
    $c_{a,s\cup t} = c_{as} + c_{at}$
  sinon
    $E \leftarrow E \backslash \{(a, s), (a, t)\}$
  fin
fin

Algorithme 8 : Algorithme de Stoer-Wagner

Entrées : $G = (V, E, c)$, $a \in V$
Résultat : V_1, V_2 une coupe minimale de G, ω_m son poids
Initialisation : $\omega = 0$, $V_1 = V$, $V_2 = \emptyset$, $V' = \{\{i\}/i \in V\}$, $E' = \{(\{i\}, \{j\})/(i, j) \in E\}$, $c'_{\{i\}\{j\}} = c_{ij}$, $G' = (V', E', c')$;
Tant que $|V'| > 1$ **faire**
  $((s, t), \omega) =$ choisir_sommets$(G', \{a\})$;
  si $\omega < \omega_m$ **alors**
    $\omega_m \leftarrow \omega$;
    $V_1 \leftarrow t$
  sinon
  fin
  fusionner_sommets$(G', (s, t))$
fin
$V_2 \leftarrow V \backslash V_1$

est $\mathcal{O}(|V|)$.

Dans le cas qui nous intéresse, celui d'un graphe de dettes $G = (V, E, y, c)$, quelques changements s'imposent :

— Les graphes de dettes étant orientés, il faut les rendre non-orientés pour appliquer cet algorithme. Cela peut donner lieu à une coupe (A, B) minimale, mais telle qu'une grande majorité des arêtes aillent de A vers B (ou l'inverse). Ceci pose une première limite à cette approche, car la minimisation des échanges entre différents groupes est alors peu efficace.

— Le choix de la fonction de poids est crucial. Dans le cas où les poids dans le graphe de dettes sont constants égaux à 1 ($c = 1$), prendre pour fonction de poids la capacité y est une bonne solution. Sinon, on peut s'intéresser à un monôme de y et c de la forme $y^\alpha c^\beta$ pour la fonction de poids utilisée dans Stoer-Wagner. Le choix d'un monôme est lié à la simplicité de calcul. On peut ainsi en choisissant $\alpha \ll \beta$ optimiser la réduction de dettes le long des arêtes de poids élevé, ou à l'inverse favoriser la réduction des dettes importantes en fixant $\alpha \gg \beta$.

Un grand inconvénient de l'algorithme de Stoer-Wagner est qu'il cherche une coupe minimale de manière absolue, sans pondérer par le nombre de sommets dans cette coupe. Son utilisation donnerait ainsi une coupe déséquilibrée dans notre contexte, et n'aiderait pas à la résolution du problème posé.

6.2 Dépliage rapide des communautés

L'algorithme suivant [24] a été proposé en 2008 par Blondel, Guillaume, Lambiotte et Lefebvre. Il repose sur l'utilisation d'une mesure de partition en communautés appelée modularité.

Définition 10. Dans un graphe orienté $G = (V, E, c)$, on définit la *modularité* $Q = \frac{1}{2n} \sum_{i,j \in V} \left(c_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j)$ avec :

— $n = |V|$
— c_{ij} le poids de l'arête (i, j)
— $k_i = \sum_{j \in V} c_{ij}$
— C_i est la communauté du sommet i
— δ est la fonction de Kronecker $\delta(C_i, C_i) = \begin{cases} 1 \text{ si } C_i = C_j \\ 0 \text{ sinon} \end{cases}$
— $m = \frac{1}{2} \sum_{i,j \in V} c_{ij}$

Le terme $\frac{k_i k_j}{2m}$ correspond à une moyenne des poids des arêtes adjacentes aux sommets i et j. Ainsi, si cette moyenne est plus grande que le poids de l'arête entre i et j, mettre les sommets dans la même communauté peut paraître étrange car ils sont relativement distants. C'est pour cela que le terme ajouté à la modularité est négatif, et positif dans le cas de sommets fortement reliés. Ce terme n'apparaît que pour des sommets dans la même communauté (on ne s'intéresse pas aux distances entre sommets de communautés différentes). On cherche ainsi à mettre dans la même communauté les termes qui sont reliés par les arêtes les plus lourdes par rapport à la moyenne locale des poids des arêtes. Localement, on crée donc des communautés, ce qui se transmet à l'échelle globale.

L'algorithme procède en deux parties, itérées l'une après l'autre. La première crée une communauté pour chaque noeud (la plupart sont destinées à diparaître), puis pour chaque sommet $i \in V$, pour chaque voisin j de i, on regarde si la modularité augmente en enlevant i de sa communauté et en l'ajoutant à celle de j. Si c'est le cas, on ajoute i à la communauté qui maximise le gain de modularité. Puis on itère tant que des augmentations de modularité sont possibles. La seconde phase construit un nouveau graphe dont les sommets sont les communautés du graphe précédent, et dont les arêtes sont les sommes des arêtes entre deux communautés du graphe précédent. On peut alors réappliquer la première phase à ce graphe.
Cet algorithme permet de traiter des graphes avec 100 millions de noeuds, ce qui est une grande amélioration par rapport aux précédents algorithmes, qui dépassaient difficilement 5 millions de noeuds [24].

6.3 Graphes petit monde

L'étude des graphes de dettes peut amener à se demander quelles sont les particularités de ces graphes, et si on peut les exploiter. Les *graphes petit monde* sont une catégorie de graphes qui semble bien s'apparenter aux graphes de dettes. L'étude de ces graphes a commencée dans les années 2000 [25, 26], avec l'explosion du nombre de données à traiter, notamment sur Internet.

Définition 11 (Distance dans un graphe)**.** Soit un graphe orienté $G = (V, E)$. Pour $u, v \in V$, on définit la *distance* entre u et v dans G par $d(u, v) = \inf\{n \in \mathbb{N} / \exists u = s_0, \cdots, s_n = v, (s_i, s_{i+1}) \in E \text{ ou } (s_{i+1}, s_i) \in E\}$. Autrement dit, c'est le nombre minimum d'arcs séparant u et v dans G.

Définition 12. Un graphe petit monde est un graphe dans lequel la distance moyenne L entre deux sommets est de l'ordre du logarithme du nombre de sommets dans le graphe. On a ainsi $L = \Theta(\log |V|)$. La constante doit bien sûr être petite pour que la définition ait un intérêt. En pratique, ce sont des graphes dans lequel le rapport

entre le nombre de voisins d'un sommet et la distance moyenne entre deux sommets est faible. La définition implique que le graphe soit connexe, mais on peut se ramener à l'étude des composantes connexes dans le cas contraire.

Les graphes de dettes semblent être des graphes petit monde : en effet, si une entreprise est en relation avec peu d'autres entreprises, on doit pouvoir facilement passer d'une entreprise à l'autre en peu de factures. L'étude de nos graphes de dettes reviendrait alors à l'étude d'un graphe petit monde, et les communautés correspondraient aux sommets qui sont fortement interconnectés, qui formeraient ainsi des groupes de petit effectif. Une étude rigoureuse du fait que les graphes de dettes sont des graphe petit monde sera un des points étudiés lors du stage.

L'algorithme de dépliage rapide des communautés, basé sur la modularité, fonctionne bien sur les graphes petit monde, et leur structure (peu de voisins pour chaque sommet) donne des garanties sur les temps de calcul [25, 26]. Les calculs sont ainsi notablement plus rapides dans le cas d'un graphe petit monde, ce qui justifie l'utilisation de cet algorithme dans le cas qui nous intéresse. Le fait que les graphes étudiés soient des graphes petit monde donne aussi des garanties sur les communautés, qui ne seront pas trop grandes de manière à pouvoir appliquer nos algorithmes.

6.4 Algorithme multiniveau basé sur le noyau (*Kernel-Based Multilevel Algorithm*)

On a vu un algorithme de dépliage rapide des communautés qui permettait de traiter de grands graphes. Une autre méthode pour ce faire est de réduire la taille du graphe étudié avant de chercher les communautés, par des méthodes à noyau [27] (2005). On construit ainsi un algorithme à plusieurs niveaux, où chaque niveau correspond à un graphe plus petit que le niveau précédent. On procède en trois phases :

— Épaissification (Alg. 9) : Lors de cette phase, on diminue la taille du graphe en marquant les sommets. On applique plusieurs fois cette étape jusqu'à obtenir un graphe permettant l'utilisation d'algorithmes de détection de communautés classique. On se donne pour cette phase une fonction de poids sur les sommets k, qui peut varier selon le type de communauté recherché. Dans notre contexte, on prendrait $k(x) = \sum_{y \in V} c_{xy}$. La fusion de deux noeuds se fait en remplaçant ces deux noeuds par un seul, et en sommant les arêtes sortant des deux noeuds initiaux pour obtenir les arêtes sortant du noeud nouvellement formé.

Algorithme 9 : Épaissification

Données : $G_i = (V, E, c)$ graphe orienté pondéré
Résultat : G_{i+1} graphe orienté pondéré obtenu par épaissification de G_i
Initialisation : Tous les noeuds sont non marqués;
Tant que Un noeud est non marqué **faire**
  Choisir aléatoirement un noeud x non marqué;
  si Tous les voisins de x sont marqués **alors**
    Marquer x
  sinon
    Soit y le voisin de x non marqué qui maximise $\frac{c_{xy}}{k(x)} + \frac{c_{xy}}{k(y)}$;
    Fusionner x et y et marquer le nouveau noeud ainsi formé
  fin
fin
Renvoyer le nouveau graphe obtenu

— Première séparation : Une fois le graphe suffisament réduit, appliquer un algorithme de détection de communautés classique (par exemple la détection de communautés spectrale (*Spectral community detection*) [27]).
— Raffinement : À partir d'un graphe réduit G_{i+1}, former des communautés dans le graphe G_i suivant la règle suivante : deux noeuds fusionnés dans G_{i+1} sont dans la même communauté dans G_i.

On remarque que cet algorithme donne accès à une classification hiérarchique, comme lors du dépliage rapide des communautés.

7 Conclusion

Dans ce mémoire a été présenté le problème de réduction de la dette, de manière générale puis avec deux modèles précis. Le premier, la réduction partielle de la dette, est un problème accessible et résolu en temps quasi-polynomial en la taille des entrées (polynomial si on fixe une somme maximum pour les factures). Cependant il peut paraître un peu simpliste dans un contexte économique, et reste lourd sur de grands graphes. Le second modèle, la suppression intégrale de dettes, est plus réaliste, mais il est NP-complet. Ce problème sera une composante importante du stage qui suivra ce mémoire, avec la modélisation à partir de données réelles.
Des outils pour l'analyse des graphes utiles dans le contexte de la réduction des dettes ont aussi été présentés, à savoir la recherche de cycles et la détection de communautés. Cette dernière permet de se ramener à des temps de calculs raisonnables, en perdant un peu en optimalité des résultats obtenus.

Remerciements

Un grand merci à mon encadrant de mémoire (et de stage) Nazim Fatès pour l'orientation des recherches et ses conseils en LaTeX, sans lequel ce mémoire aurait été beaucoup moins esthétique et beaucoup moins bien fourni.

Bibliographie

[1] D. W. Diamond and P. H. Dybvig. Bank runs, deposit insurance, and liquidity. *Journal of political economy*, 91(3) :401–419, 1983.

[2] T. Fleischman, P. Dini, and G. Littera. Liquidity-saving through obligation-clearing and mutual credit : an effective monetary innovation for smes in times of crisis. *Journal of Risk and Financial Management*, 13(12) :295, 2020.

[3] G. Iosifidis, Y. Charette, E. M. Airoldi, G. Littera, L. Tassiulas, and N. A. Christakis. Cyclic motifs in the Sardex monetary network. *Nature human behaviour*, 2(11) :822–829, 2018.

[4] Y. Kao, V. Ragupathy, K. Vela Velupillai, and S. Zambelliet. Noncomputability, unpredictability, undecidability, and unsolvability in economic and finance theories. *Complexity*, 18(1) :51–55, 2012.

[5] F. Allen and D. Gale. Financial contagion. *Journal of political economy*, 109(1) :1–33, 2000.

[6] V. Gazda, D. Horváth, and M. Rešovský. An application of graph theory in the process of mutual debt compensation. *Acta Polytechnica Hungarica*, 12(3) :7–24, 2015.

[7] V. Gazda. Mutual debts compensation as graph theory application. *Challenges for Business Administrators in the New Millennium*, pages 793–811, 2000.

[8] M. Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14(3) :205–220, 1967.

[9] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36(4) :873–886, 1989.

[10] L. R. Ford and D. R. Fulkerson. Flows in networks. *Princeton university press*, 54, 1962.

[11] R. Bellman. On a routing problem. *Quarterly of applied mathematics*, 11(1) :87–90, 1958.

[12] J. C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Communications of the ACM*, 13(12) :722–726, 1970.

[13] H. Weinblatt. A new search algorithm for finding the simple cycles of a finite directed graph. *Journal of the ACM*, 19(1) :43–56, 1972.

[14] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2) :146–160, 1972.

[15] R. Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM journal on computing*, 2(3) :211–216, 1973.

[16] D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1) :77–84, 1975.

[17] C. Pătcaş. The debts' clearing problem : a new approach. *Acta Univ. Sapientiae*, 3(2) :192–204, 2011.

[18] M. M. Guntzer, D. Jungnickel, and M. Leclerc. Efficient algorithms for the clearing of interbank payments. *European Journal of Operational Research*, 106(1) :212–219, 1998.

[19] S. Schuldenzucker, S. Seuken, and S. Battiston. The computational complexity of clearing financial networks with credit default swaps. *arXiv*, 1710(01578), 2017.

[20] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8 :399–404, 1956.

[21] J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, page 165–174, 1992.

[22] D. R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2) :383–413, 1999.

[23] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4) :585–591, 1997.

[24] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics : theory and experiment*, 2008(10) :10008, 2008.

[25] D. Auber, Y. Chiricota, F. Jourdan, and G. Melançon. Multiscale visualization of small world networks. *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No. 03TH8714)*, 2003.

[26] D. A. Bader and K. Madduri. SNAP, small-world network analysis and partitioning : An open-source parallel graph framework for the exploration of large-scale networks. *2008 IEEE international symposium on parallel and distributed processing*, 2008.

[27] I. Dhillon, Y. Guan, and B. Kulis. A fast kernel-based multilevel algorithm for graph clustering. *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005.

Mémoire Maths-info

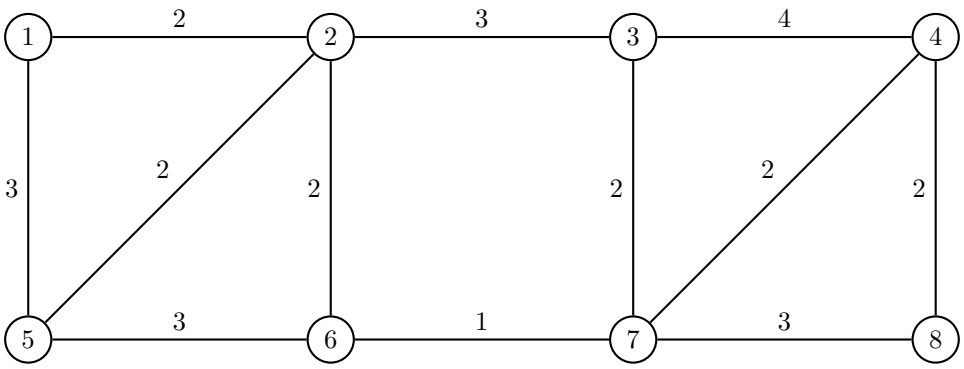
Annexe

Arthur Rousseau

Dans cette annexe sont présentés des exemples pour certains algorithmes du mémoire, trop volumineux pour y être intégrés. Sont aussi présentées quelques pistes de recherche sur le problème SID, qui n'avaient pas leur place dans un mémoire bibliographique.

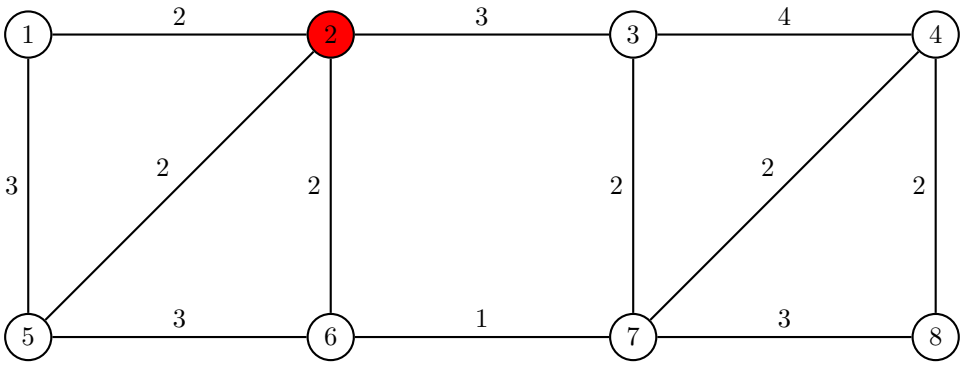
1 Algorithme de Stoer-Wagner

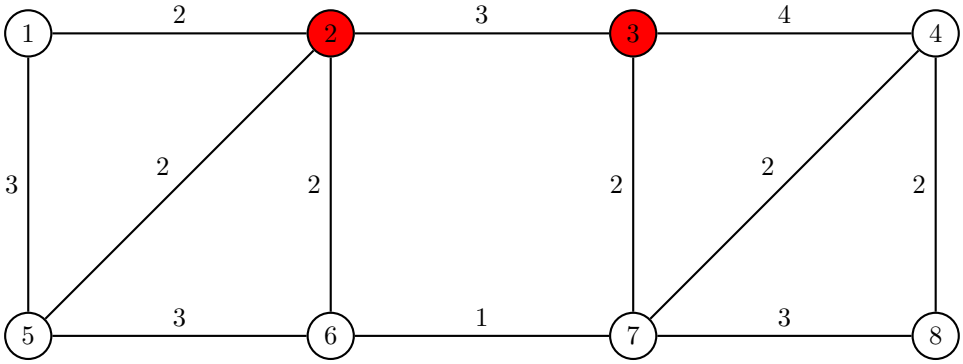
On montre un exemple d'application de l'algorithme de Stoer-Wagner au graphe suivant.

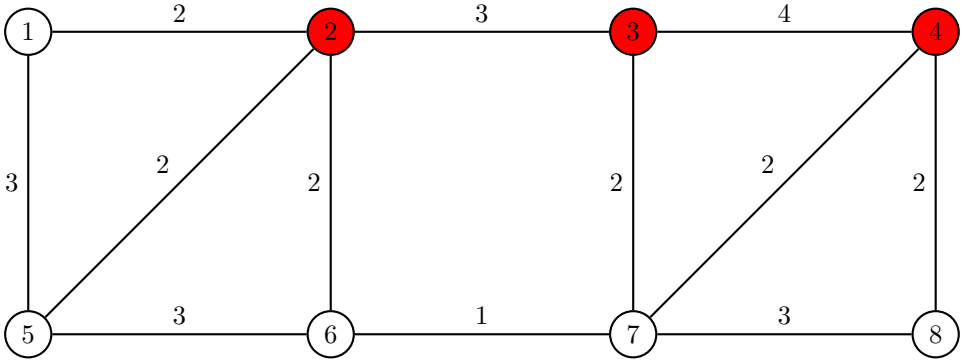


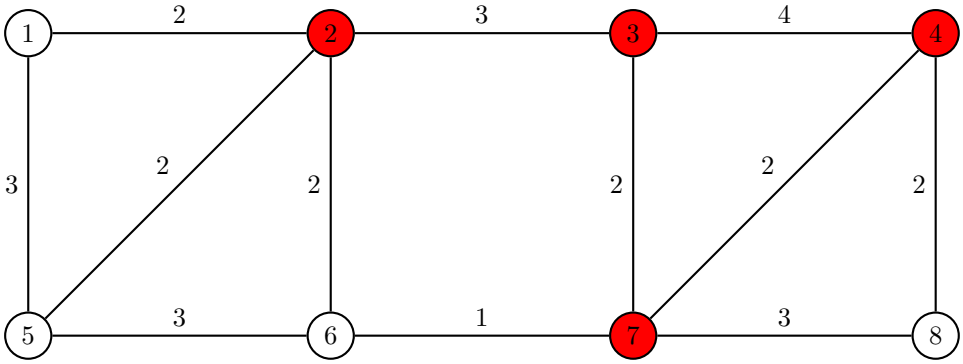
On note en rouge le noeuds choisis lors de choisir_sommets, et en orange et en jaune respectivement l'avant-dernier et le dernier des noeuds choisis. Ceux-ci sont fusionnés à la fin d'un tour.

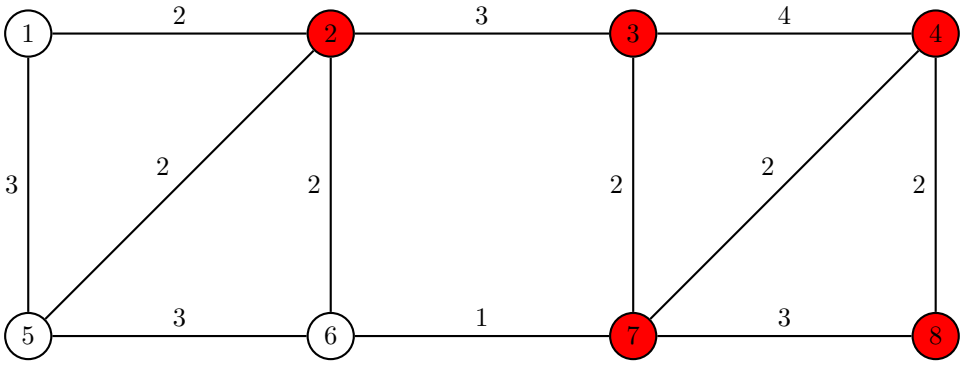
Tour 1

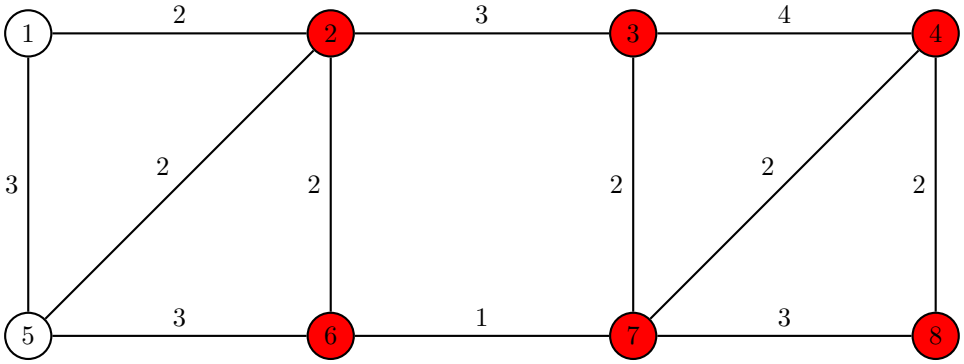


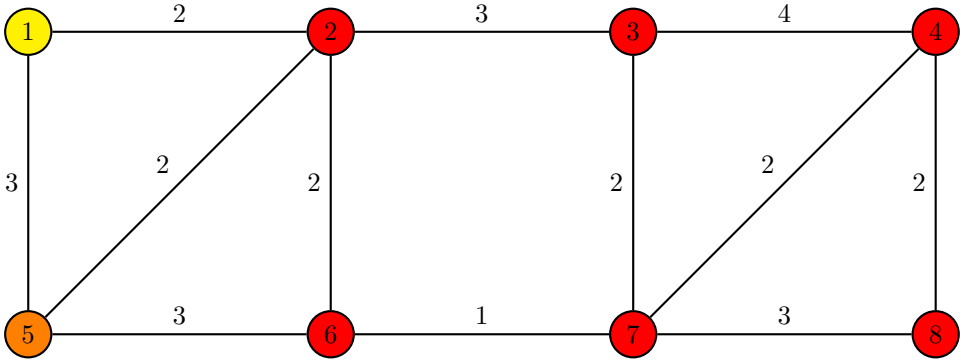

1 2 3 4
5 6 7 8
2 3 4
3 2 2 2 2
3 1 3


1 2 3 4
5 6 7 8
2 3 4
3 2 2 2 2
3 1 3


1 2 3 4
5 6 7 8
2 3 4
3 2 2 2 2
3 1 3

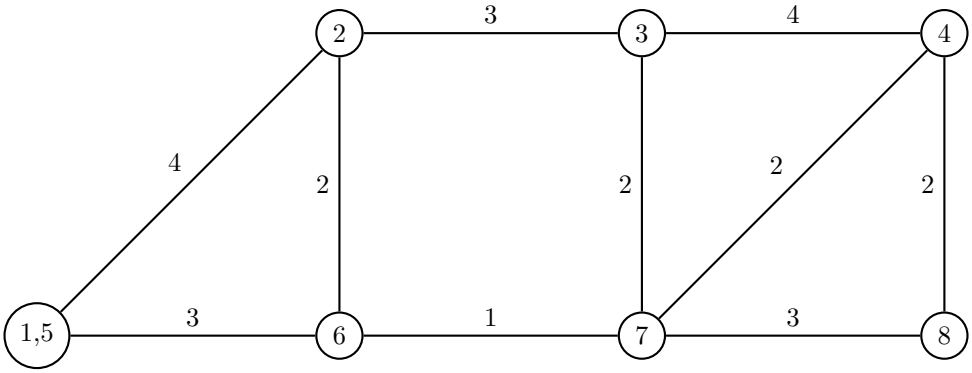

1 2 3 4
5 6 7 8
2 3 4
3 2 2 2 2
3 1 3

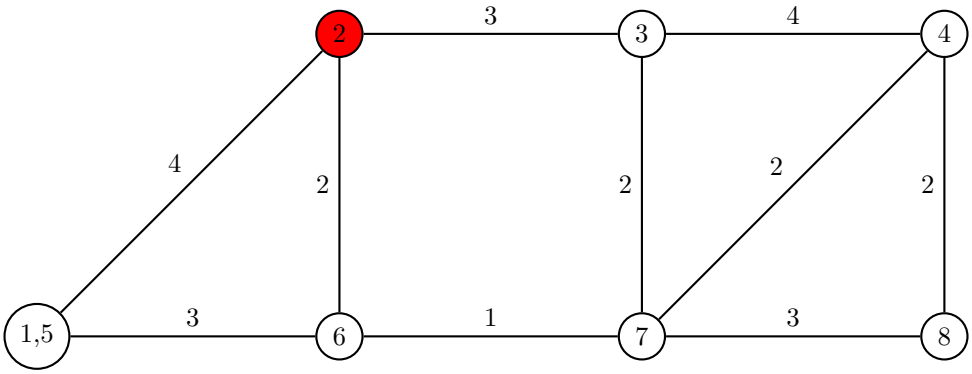


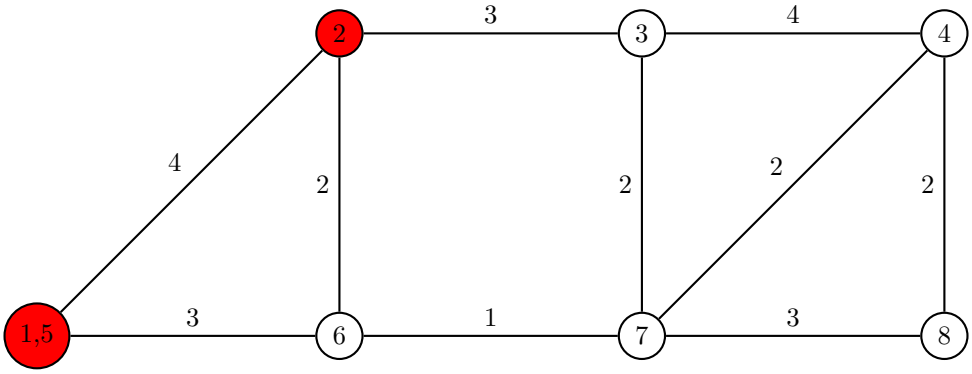


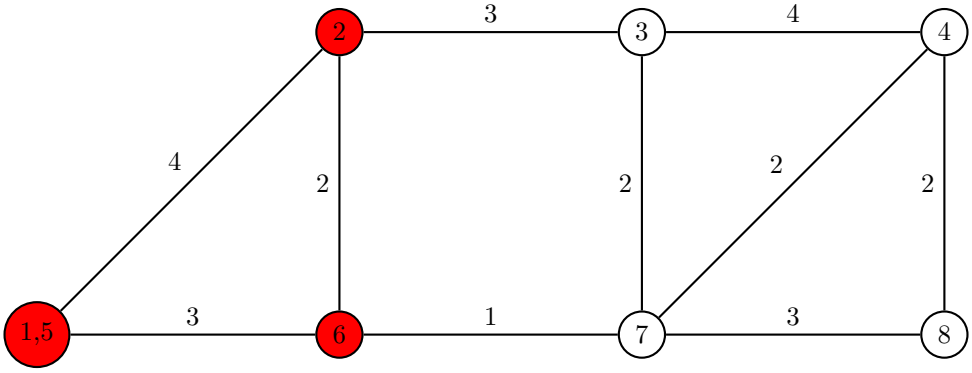
Pour ce tour, on a $s = \{5\}$ (avant-dernier sommet choisi), $t = \{1\}$ (dernier sommet choisi) et $\omega = 5$ (distance entre le dernier sommet choisi et le reste du graphe).

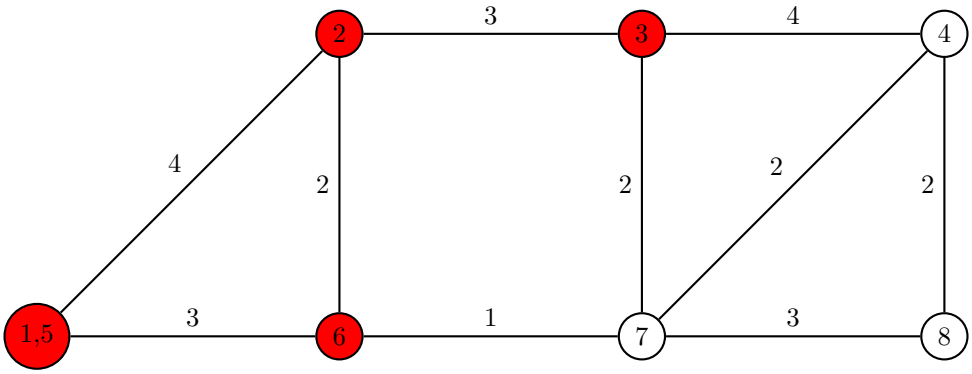
Tour 2

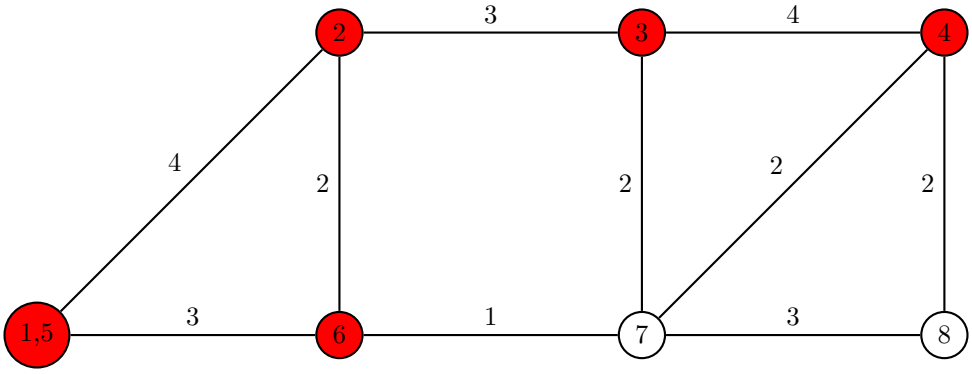


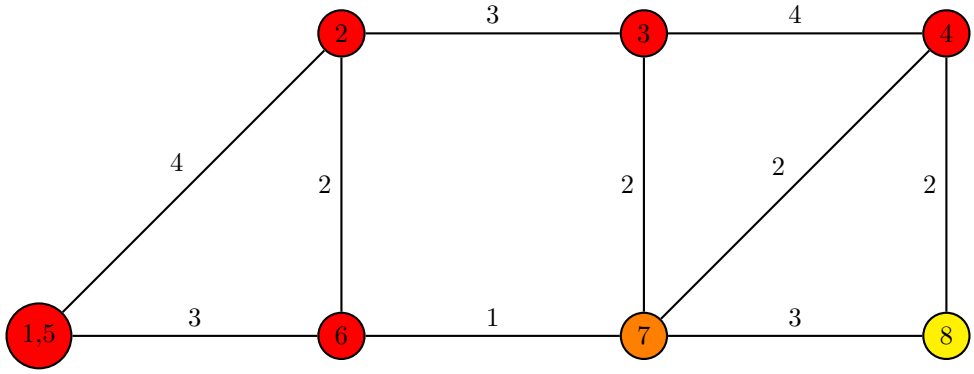



2
3
4
3
4
4
2
2
2
2
1,5
3
6
1
7
3
8


2
3
4
3
4
4
2
2
2
2
1,5
3
6
1
7
3
8

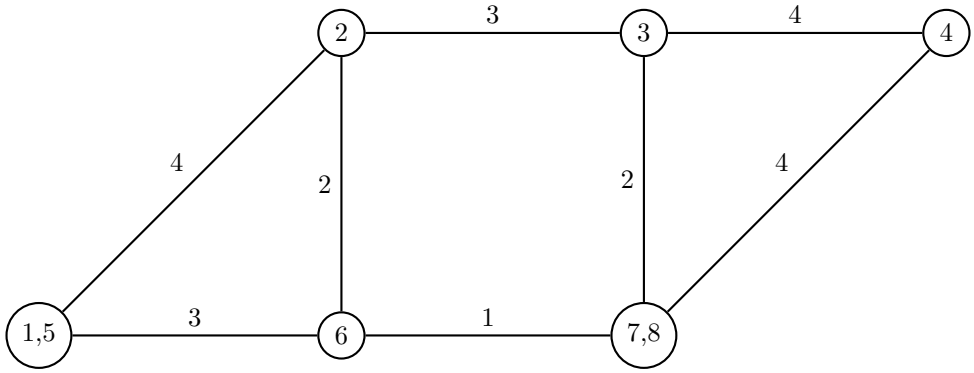

2
3
3
4
4
2
2
2
2
1,5
3
6
1
7
3
8

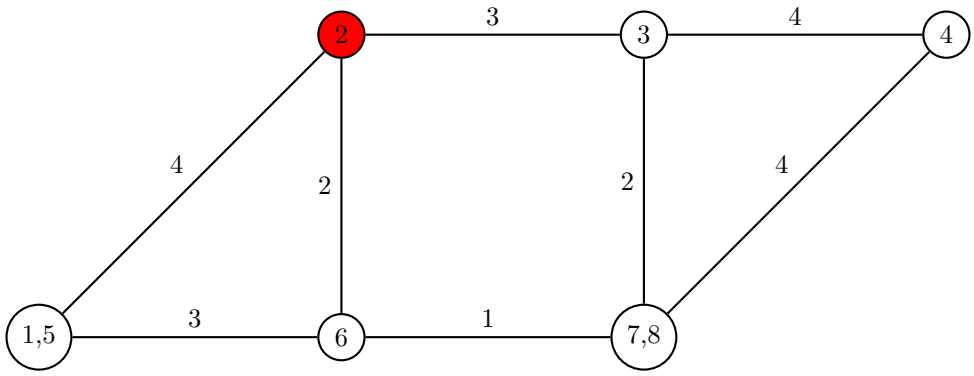

2
3
3
4
4
2
2
2
2
1,5
3
6
1
7
3
8

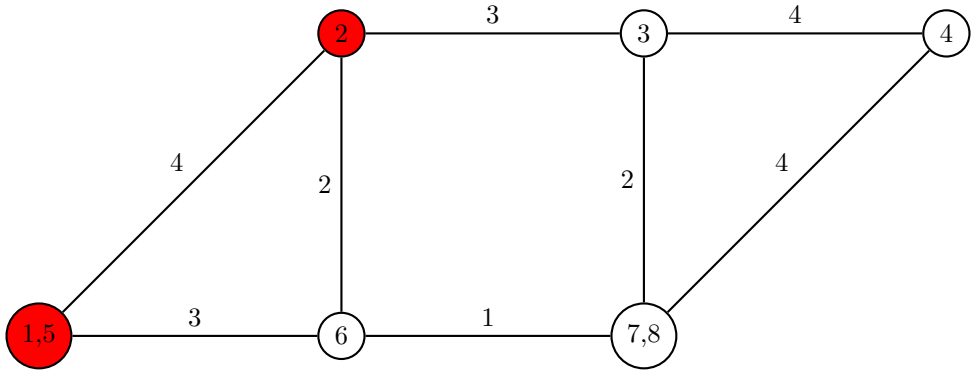


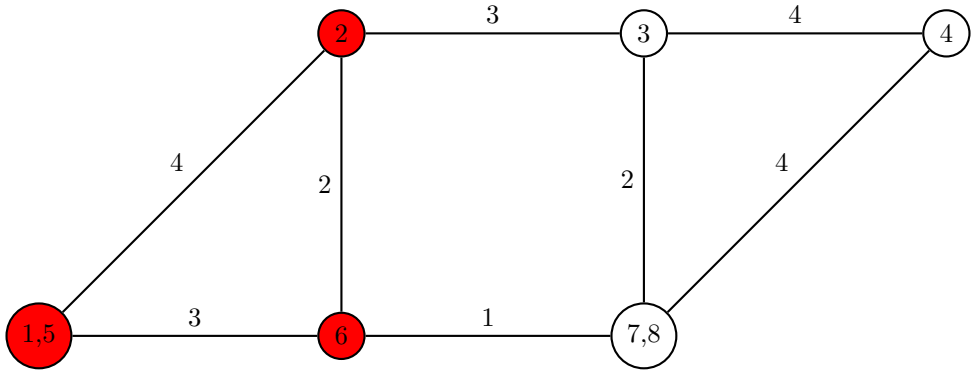
Pour ce tour, on a $s = \{7\}$, $t = \{8\}$ et $\omega = 5$.

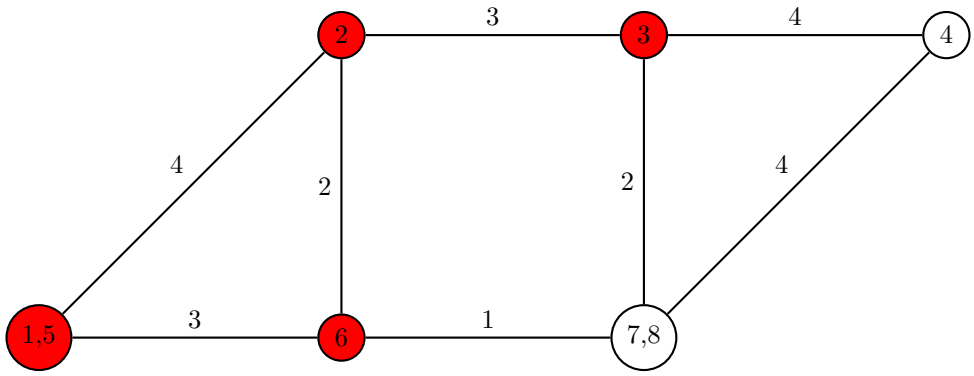
Tour 3

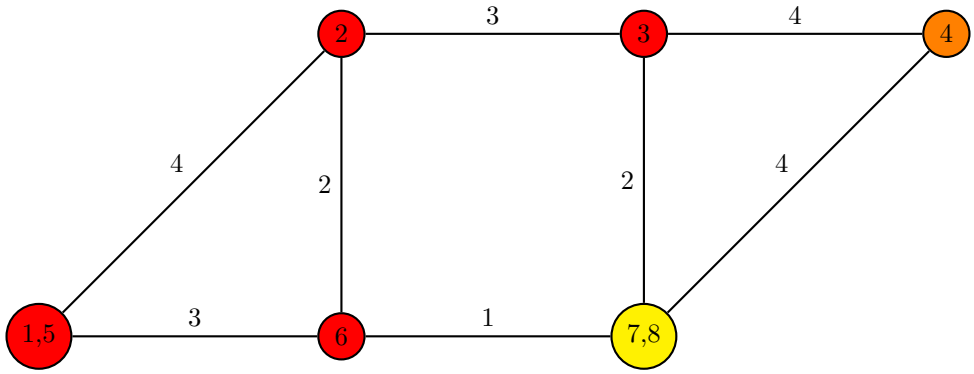






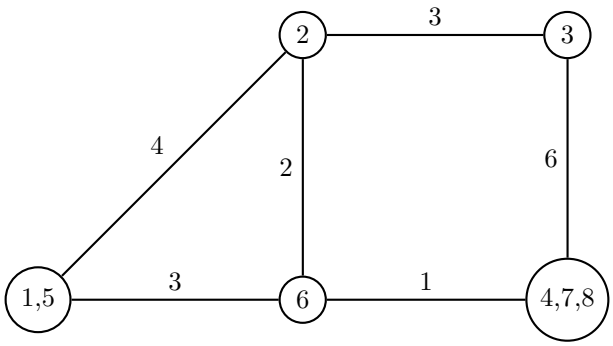


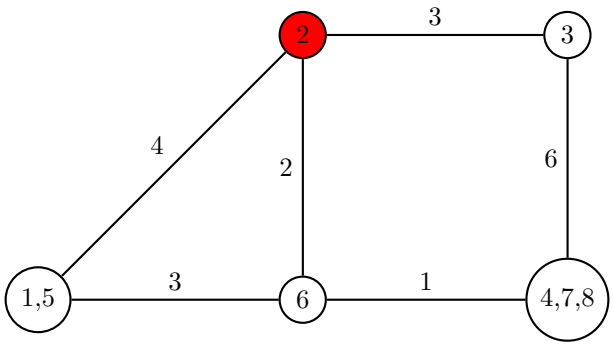


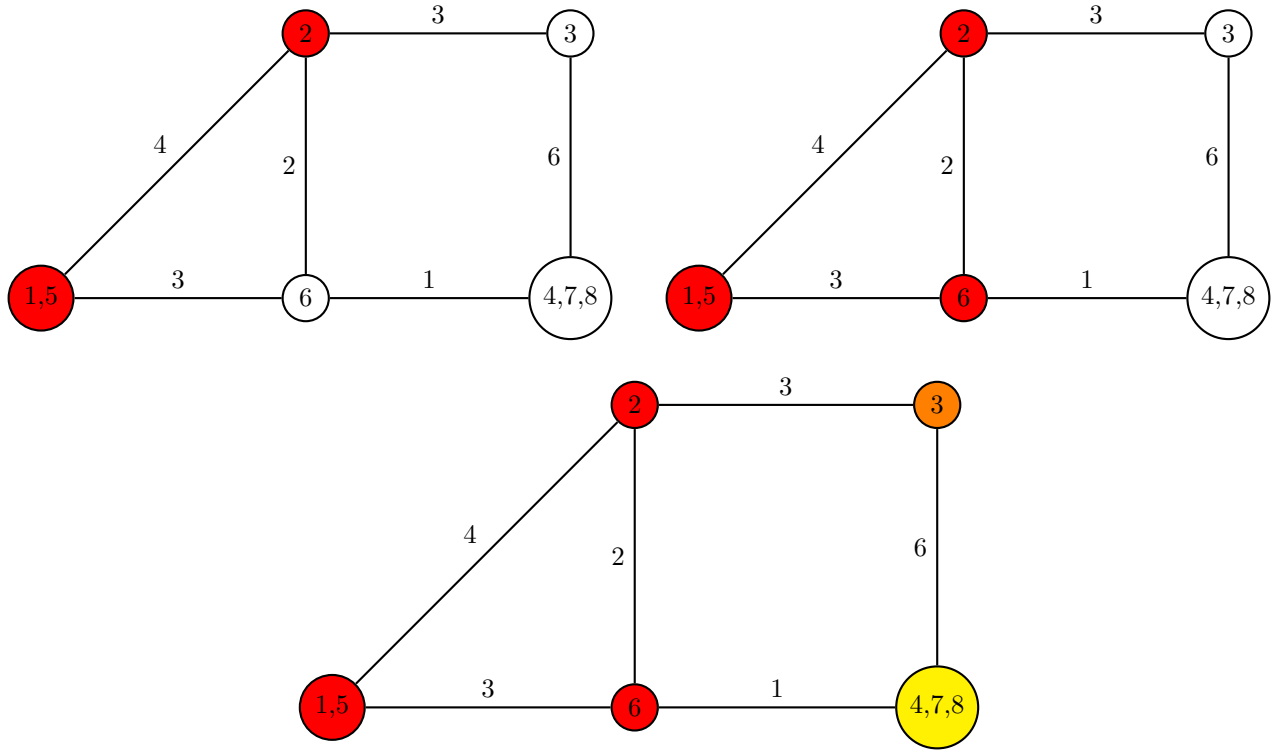


Pour ce tour, on a $s = \{4\}$, $t = \{7, 8\}$ et $\omega = 7$.

Tour 4

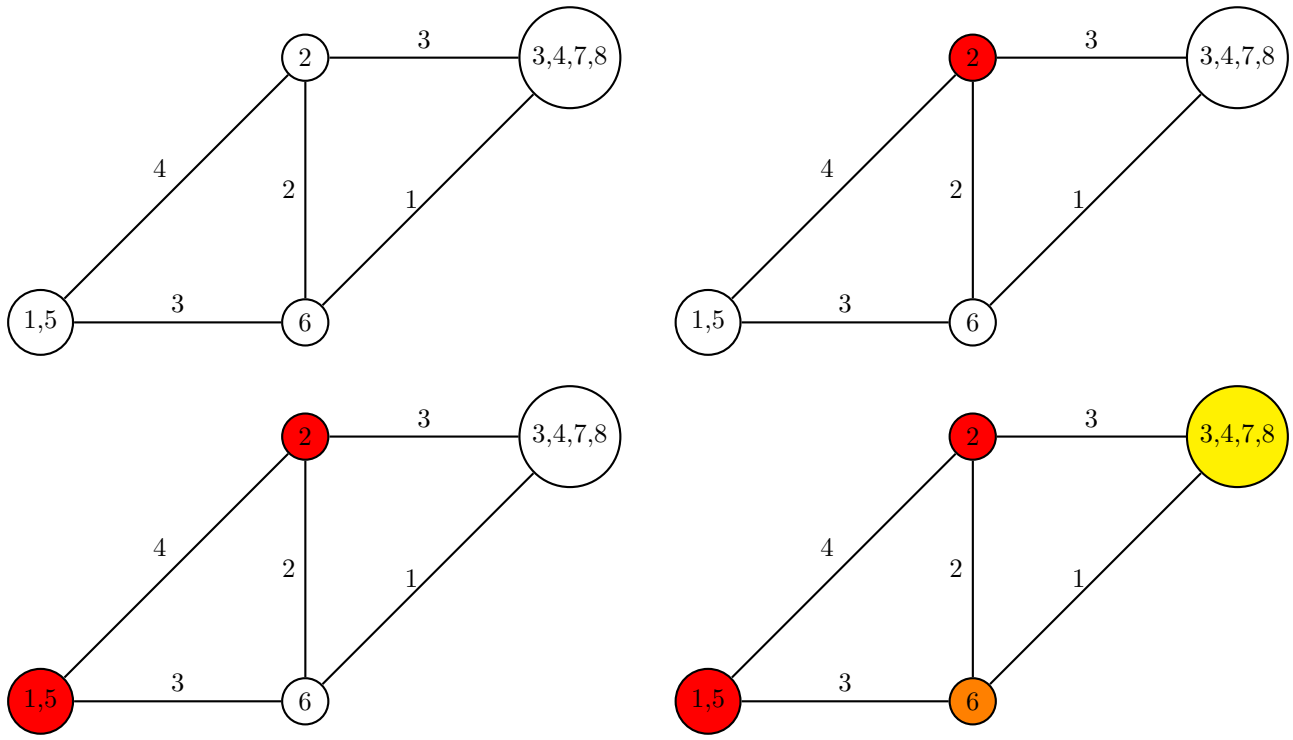






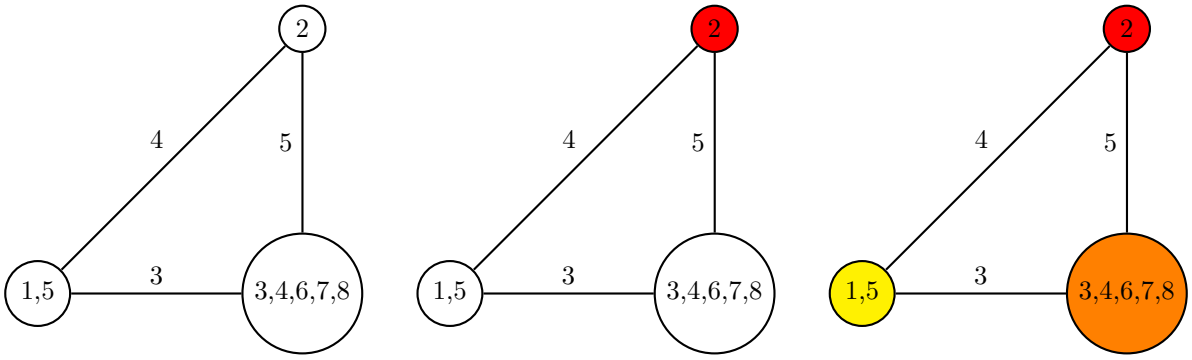
Pour ce tour, on a $s = \{3\}$, $t = \{4, 7, 8\}$ et $\omega = 7$.

Tour 5



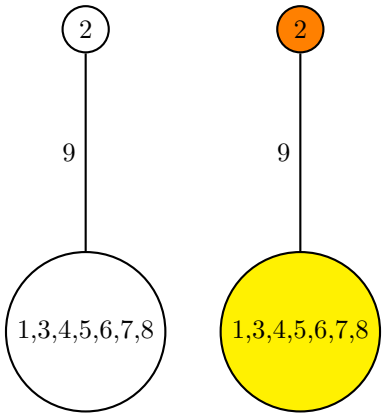
Pour ce tour, on a $s = \{6\}$, $t = \{3, 4, 7, 8\}$ et $\omega = 4$.

Tour 6



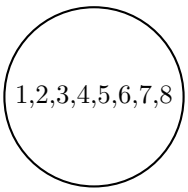
Pour ce tour, on a $s = \{3, 4, 6, 7, 8\}$, $t = \{1, 5\}$ et $\omega = 7$.

Tour 7



Pour ce tour, on a $s = \{2\}$, $t = \{1, 3, 4, 5, 6, 7, 8\}$ et $\omega = 9$.

Tour 8



Fin de l'algorithme.

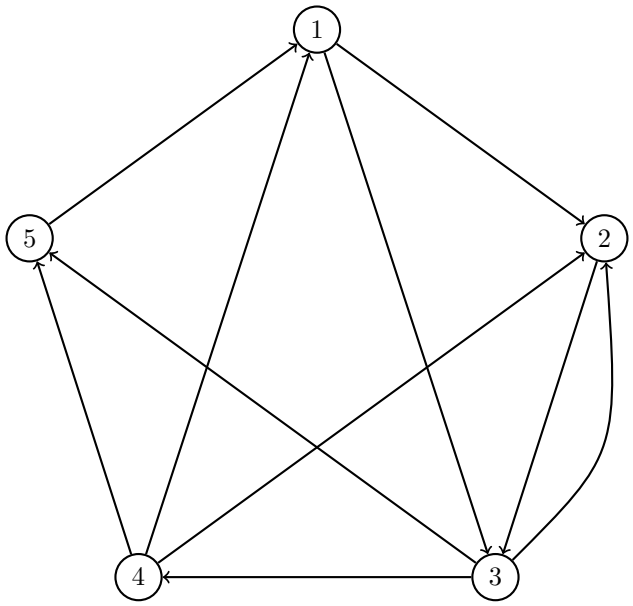
Tableau récapitulatif

Coupe	Poids de la coupe
$\{1\}, \{2,3,4,5,6,7,8\}$	5
$\{8\}, \{1,2,3,4,5,6,7\}$	5
$\{7,8\}, \{1,2,3,4,5,6\}$	7
$\{4,7,8\}, \{1,2,3,5,6\}$	7
$\{3,4,7,8\}, \{1,2,5,6\}$	4
$\{1,5\}, \{2,3,4,6,7,8\}$	7
$\{2\}, \{1,3,4,5,6,7,8\}$	9

Ainsi, d'après l'algorithme de Stoer-Wagner, la coupe minimale dans le graphe est la coupe $\{3,4,7,8\}, \{1,2,5,6\}$.

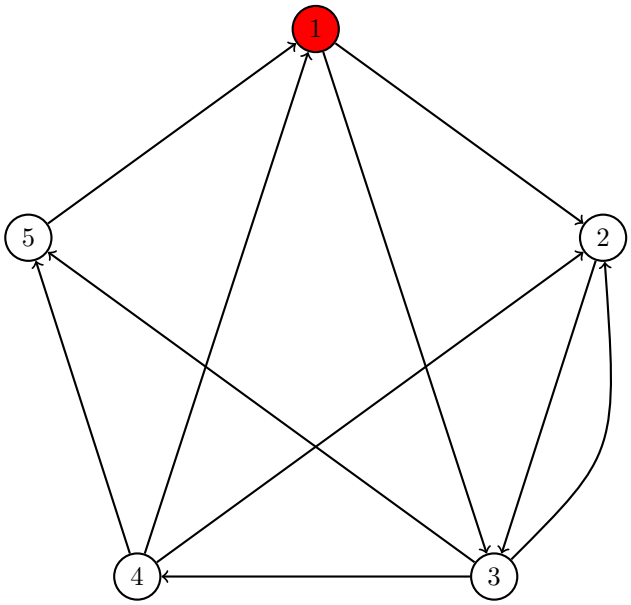
2 Algorithme de Johnson

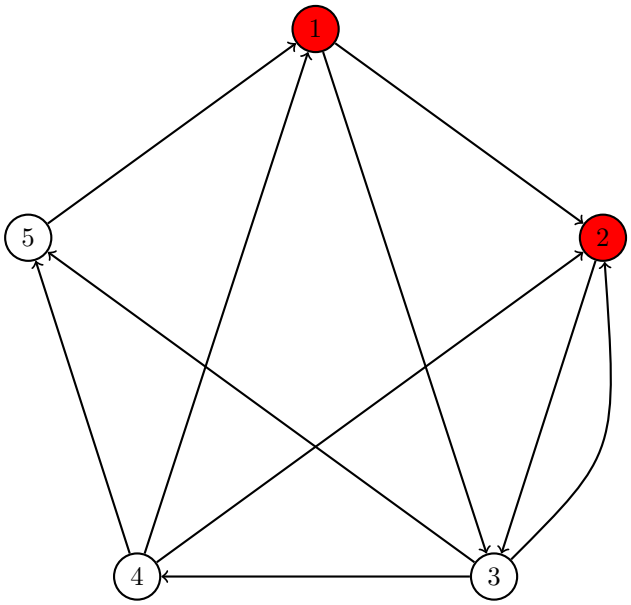
On montre un exemple d'application de l'algorithme de Johnson au graphe suivant.

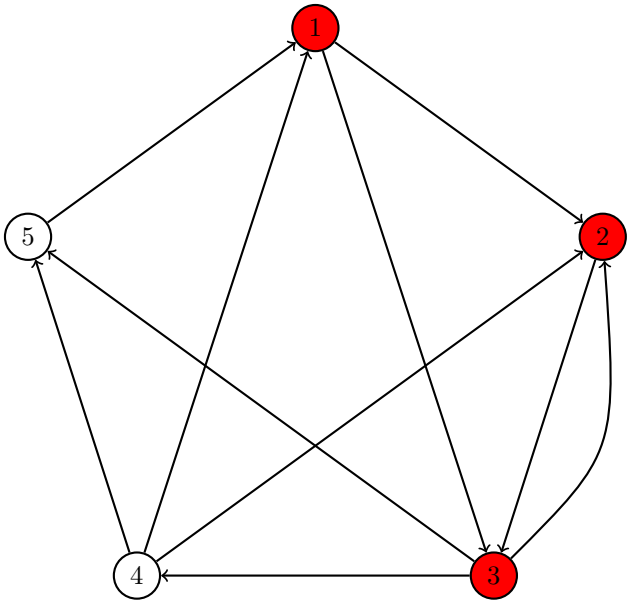


Initialement G_B est le graphe avec 5 sommets sans arêtes. Les sommets rouges sont les sommets marqués (ie bloqué) et faisant partie du cycle en cours de parcours, les sommets jaunes sont bloqués mais pas dans le cycle en cours de parcours.

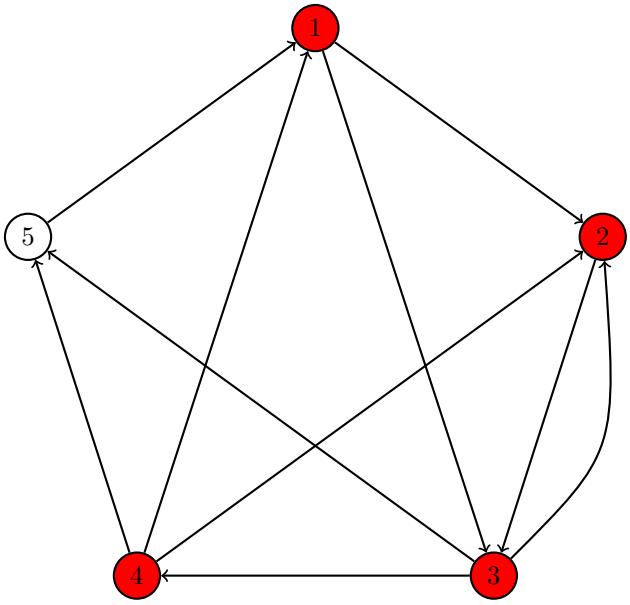
Composante connexe contenant 1 induite par $\{1,2,3,4,5\}$



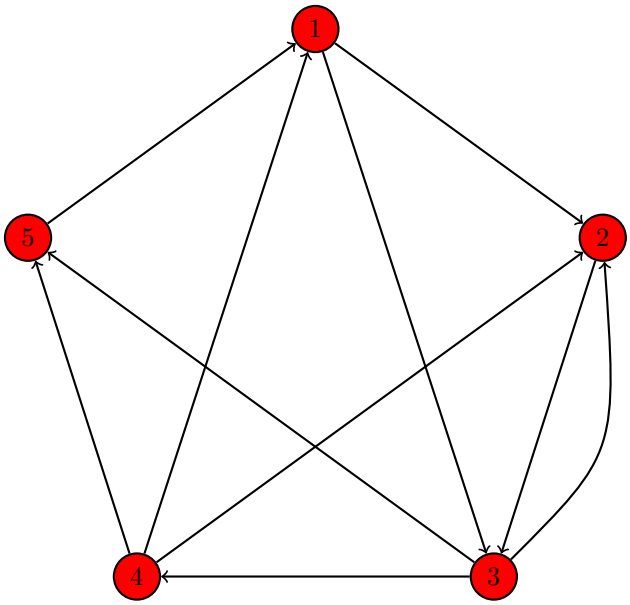




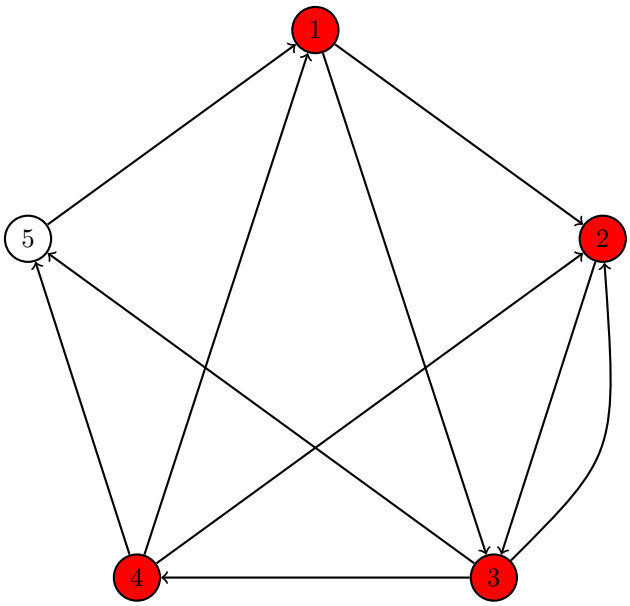
2 est déjà marqué donc ne donne lieu à rien.

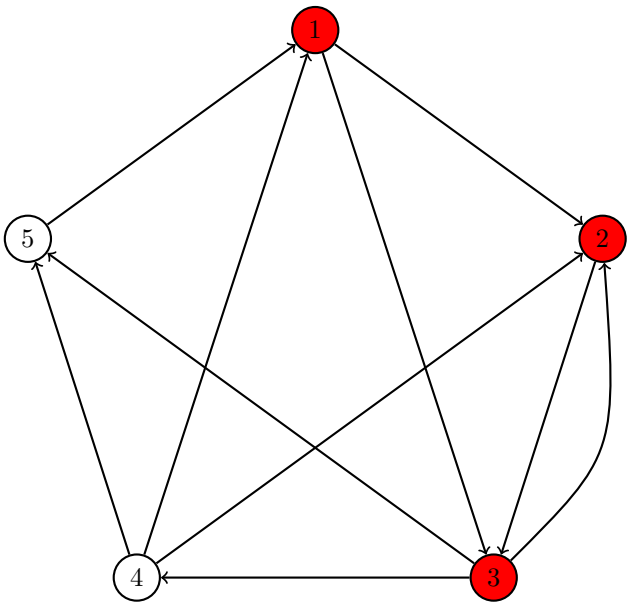


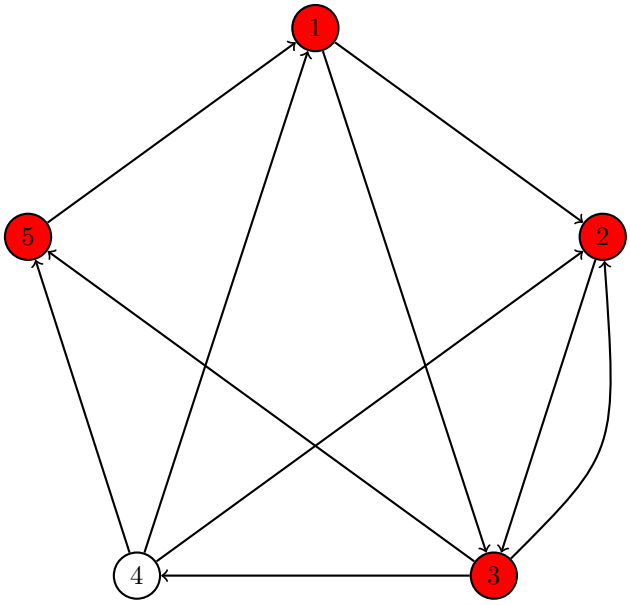
On repère le cycle (1,2,3,4).



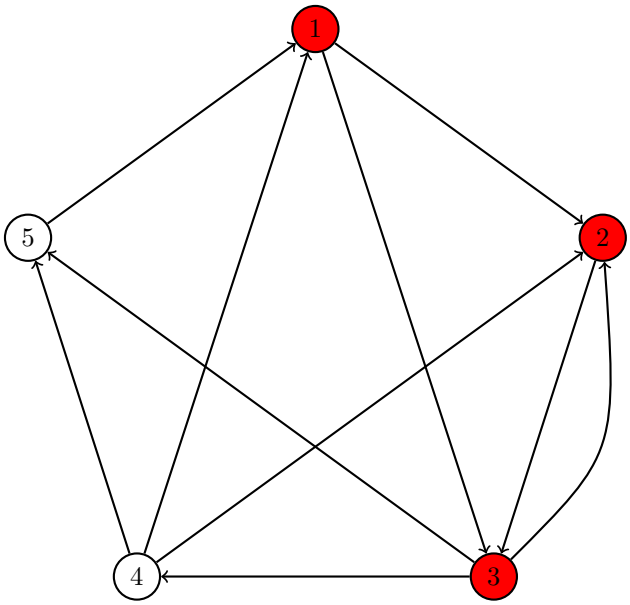
On repère le cycle (1,2,3,4,5).

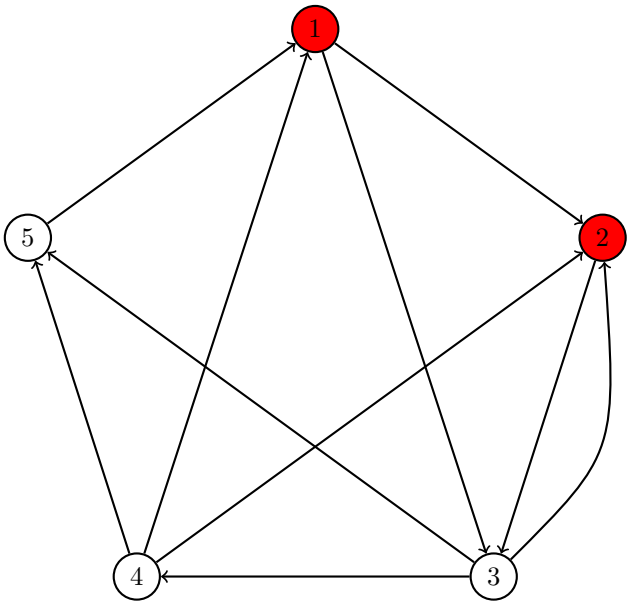


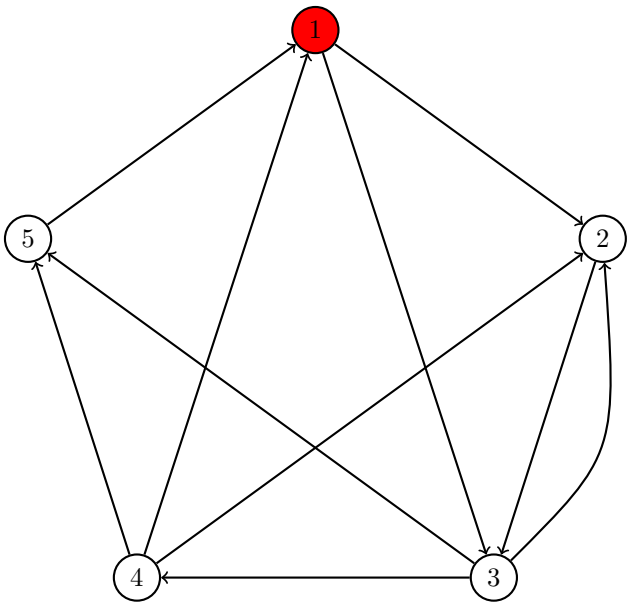


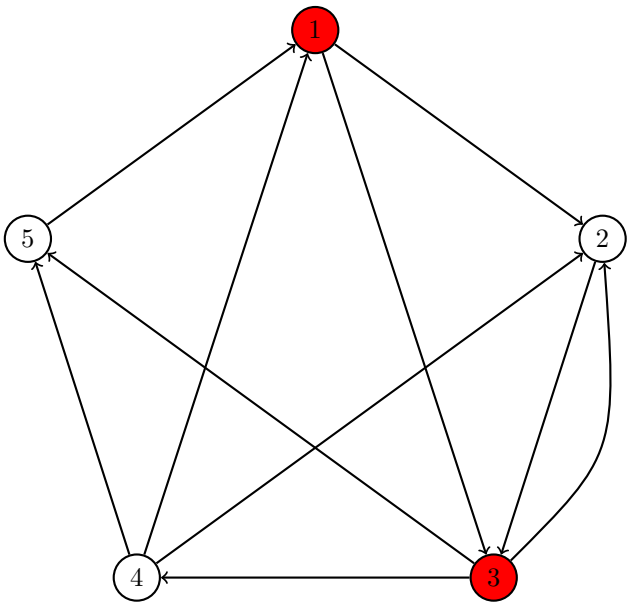


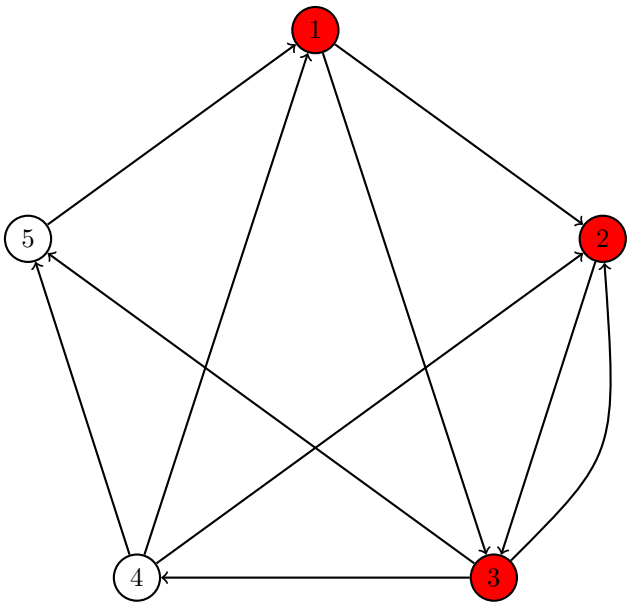
On repère le cycle (1,2,3,5).



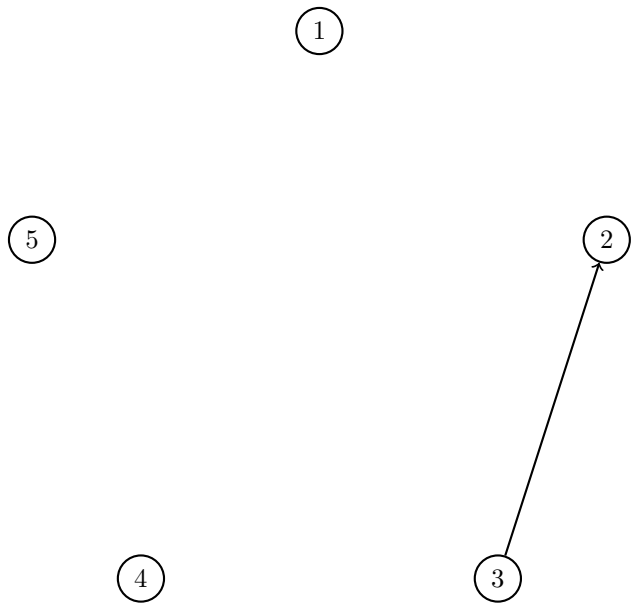




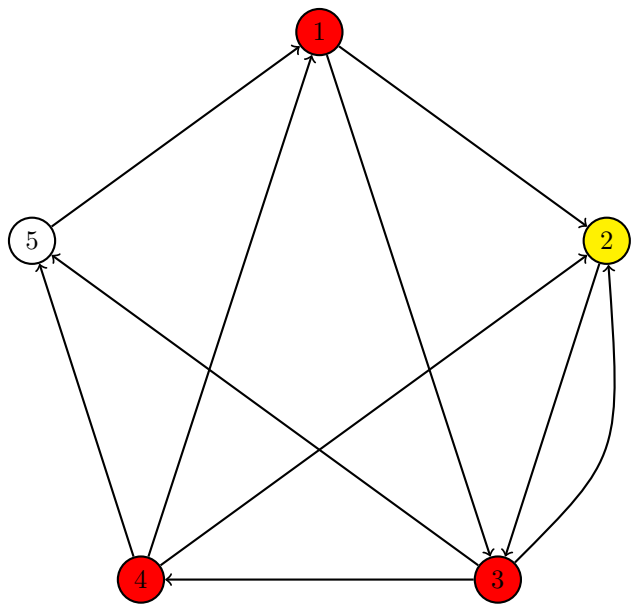




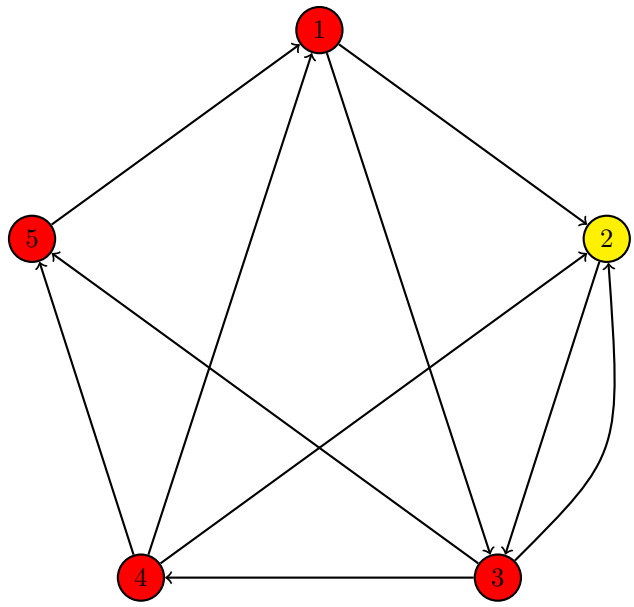
On ne trouve pas d'arête sortante à partir de 2, donc $(3, 2)$ est ajouté à E_B. 2 est marqué mais n'est plus dans la pile indiquant le cycle en cours de recherche.



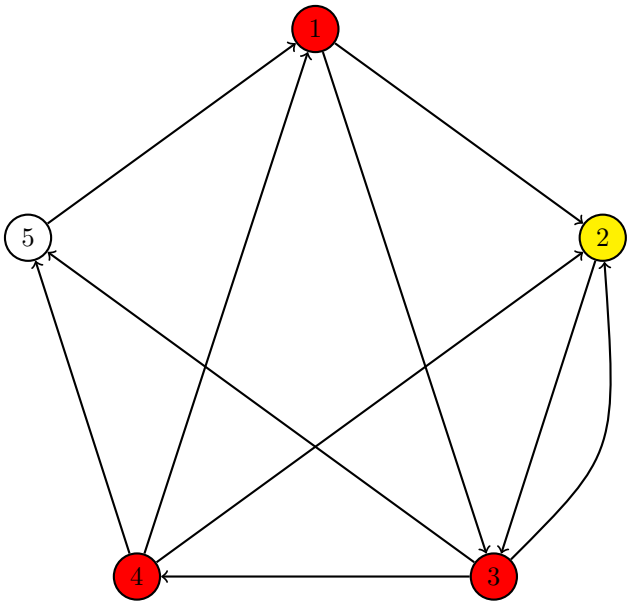
G_B

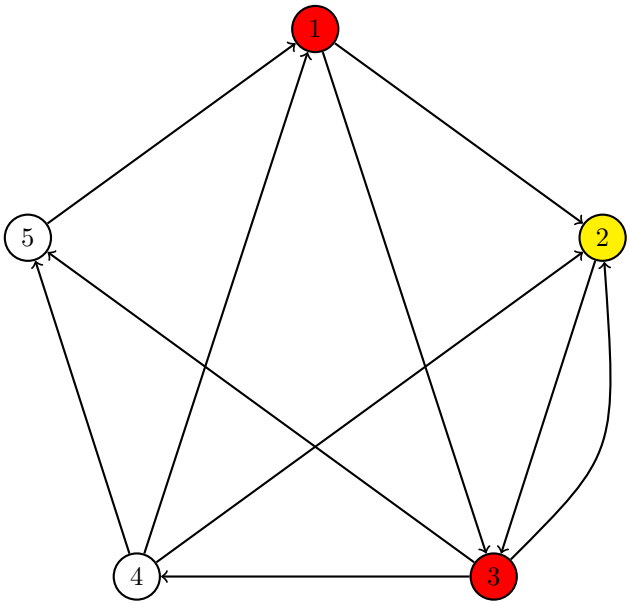


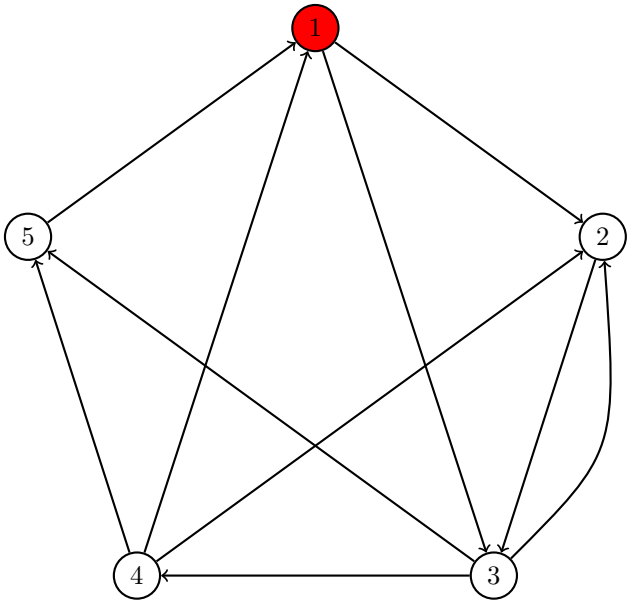
On trouve le cycle (1,3,4).



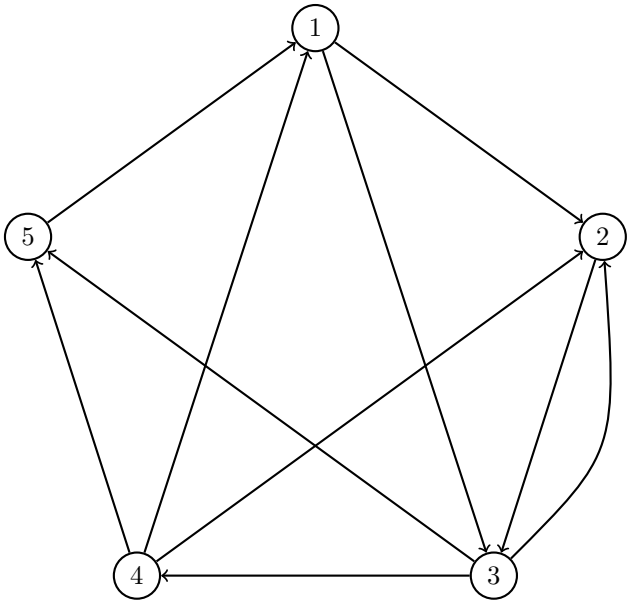
On trouve le cycle (1,3,4,5).





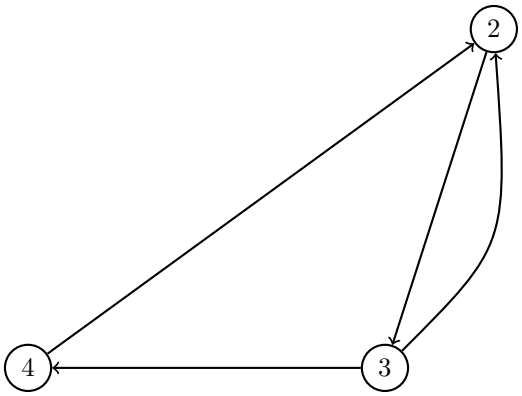


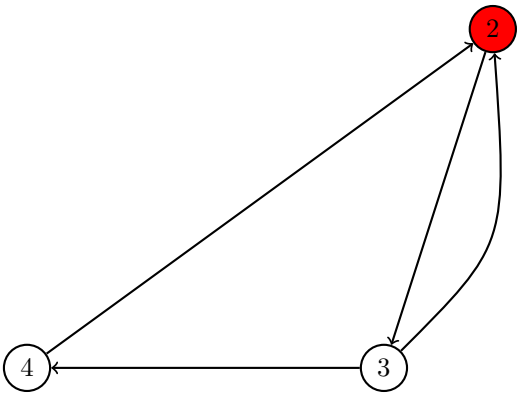
Lorsqu'on débloque 3, on regarde les arêtes partant de 3 dans G_B, ce qui fait qu'on débloque aussi 2. L'arête (3,2) est retirée de E_B.

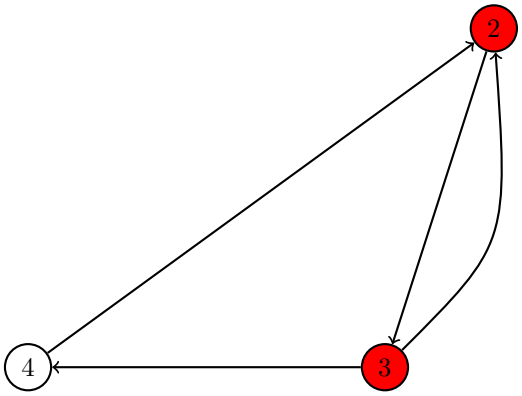


On a fini la composante fortement connexe contenant 1 induite par $\{1, 2, 3, 4, 5\}$, on s'intéresse à la composante fortement connexe contenant 2 dans le graphe induit par $\{2, 3, 4, 5\}$.

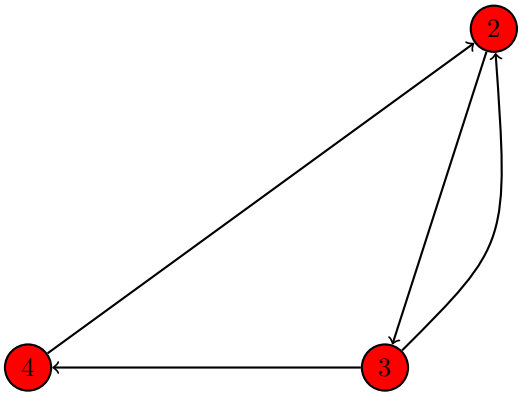
Composante connexe contenant 2 induite par $\{2, 3, 4, 5\}$



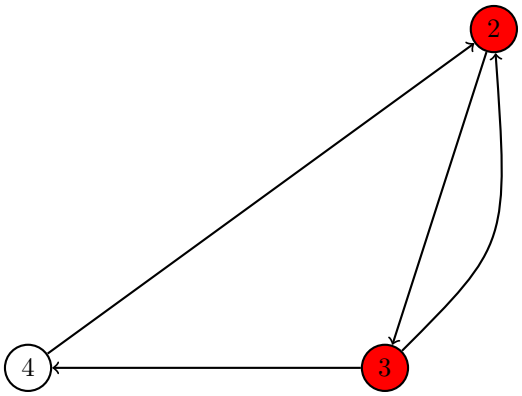


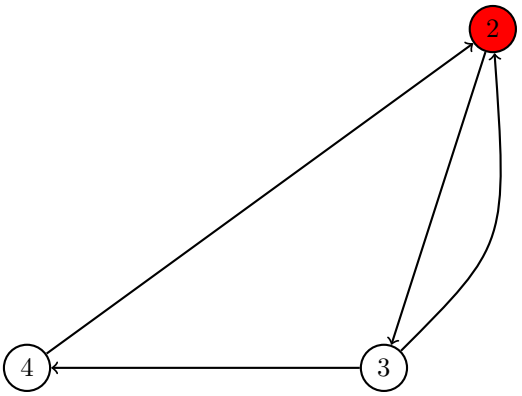


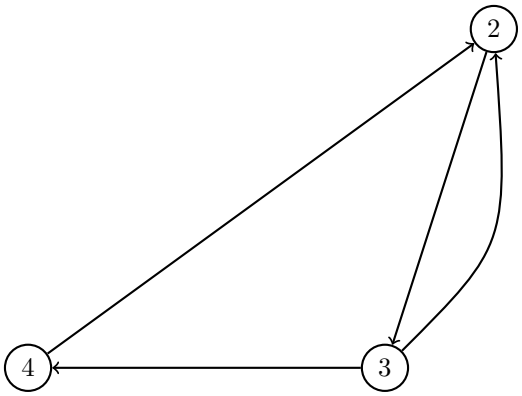
On trouve le cycle (2,3).



On trouve le cycle (2,3,4).







On a fini la composante connexe contenant 2 induite par $\{2, 3, 4, 5\}$. Les composantes connexes du graphe induit par $\{3, 4, 5\}$ sont toutes à un seul sommet, donc on ne trouvera plus de cycle dans le graphe.

Bilan

Les cycles du graphe sont (1,2,3,4),(1,2,3,4,5),(1,3,4),(1,3,4,5),(2,3),(2,3,4).

3 Suppression intégrale de dettes

On détaille ici des pistes d'algorithmes pour la suppression intégrale de dettes qui n'avaient pas leur place dans le mémoire.

3.1 Algorithme glouton

Une idée simple consiste à chercher l'arête dont le rapport entre la capacité pondérée et la somme nécessaire à la rembourser est maximal, à la supprimer, puis à recommencer (Alg. 1-2). On considère que l'origine des injections est une banque.

Algorithme 1 : Algorithme glouton de suppression intégrale de la dette

Données : $G = (V, E, y, c)$ multi-graphe de dettes pondéré
Résultat : $S \subset E$
Initialisation : $V : \varnothing, \forall v \in V,\ B(v) = I(v) = 0$ (B correspond à la somme totale reçue par un sommet, par injection ou remboursement, et I correspond à la somme injectée au niveau d'un sommet);
Tant que Il existe une arête $(v, w) \in E \backslash S$ telle que $y(v, w) - B(v) \leqslant Q$ **faire**
  si Il existe une arête $(v, w) \in E \backslash S$ telle que $y(v, w) \leqslant B(v)$ **alors**
    Soit $(v, w) \in E \backslash S$ telle que $y(v, w) \leqslant B(v)$ et $c(v, w)$ maximale;
    payer$(w, y(v, w))$;
    $B(v) \leftarrow B(v) - y(v, w)$ (v utilise une partie de son surplus d'argent)
  sinon
    Soit $(v, w) \in E \backslash S$ telle que $y(v, w) - B(v) \leqslant Q$ et $\dfrac{c(v, w)y(v, w)}{y(v, w) - B(v)}$ maximale;
    payer$(w, y(v, w))$;
    $Q \leftarrow Q - y(v, w) + B(v)$ (La somme à injecter diminue);
    $B(v) \leftarrow 0$ (v consomme tout son surplus d'argent);
    $I(v) \leftarrow I(v) + y(v, w) - B(v)$ (La dette de v à la banque augmente)
  fin
  Ajouter (v, w) à S
fin

Algorithme 2 : payer(w, S)

Données : $G = (V, E, y, c)$ multi-graphe de dettes pondéré, $I, B : V \to \mathbb{R}_+$ dettes à la banque et sommes totales reçues pour chaque sommet

Entrées : $w \in V$ sommet à payer, $S > 0$ somme à payer

Résultat : $I : V \to \mathbb{R}_+, S \subset E$

si $I(w) > 0$ **alors**

  $\delta = \min(I(w), y(v, w))$;

  $Q \leftarrow Q + \delta$ (l'excès arrivant en w sert à compenser la somme qui y a été injectée, si nécessaire);

  $I(w) \leftarrow I(w) - \delta$ (La dette de w à la banque diminue);

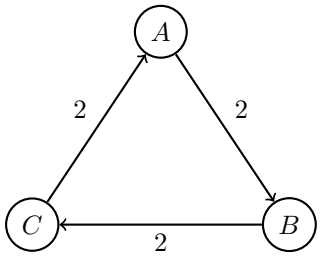
  $B(w) \leftarrow y(v, w) - \delta$ (S'il reste de l'argent après remboursement de la banque, w s'enrichit)

sinon

  $B(w) \leftarrow B(w) + y(v, w)$ (w gagne de l'argent)

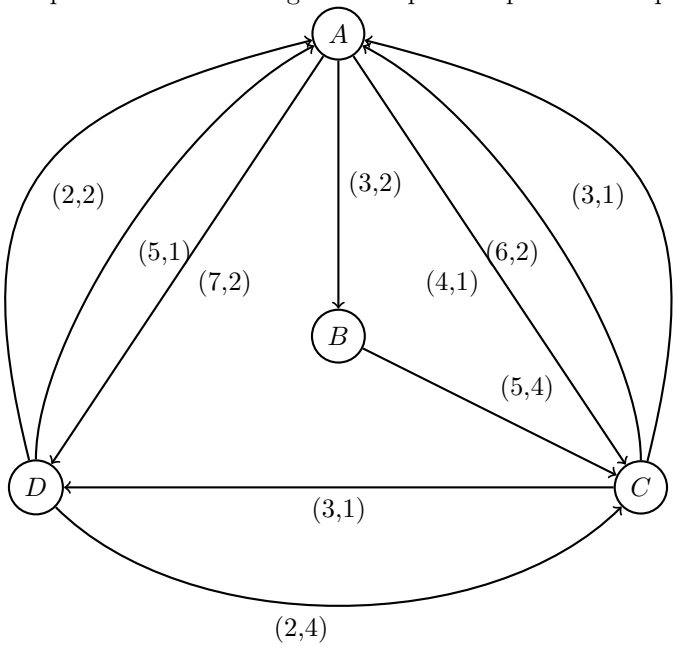
fin

Remarque. L'algorithme 1 n'est pas du tout optimal, tant sur le plan de la complexité que sur l'optimalité du graphe obtenu. En effet, sur le graphe suivant, en prenant tous les poids égaux à 1 et $Q = 1$, l'algorithme n'élimine aucune dette alors qu'il est possible de tout supprimer. Cela signale un gros problème : on ne peut élminer une dette supérieure à la somme injectée, même si celle-ci est dans un cycle qui permettrait de compenser l'excès.



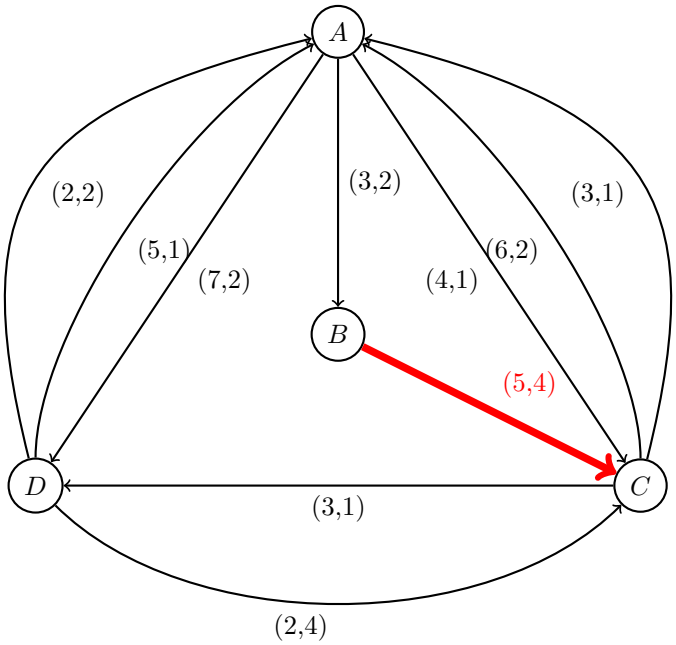
3.2 Application de l'algorithme glouton

On applique l'algorithme glouton au graphe de dettes suivant, en prenant $Q = 5$. On remarque qu'en injectant une telle somme, on pourrait rembourser toute la dette du graphe en injectant au niveau de chaque sommet dont la position nette est négative de quoi compenser cette position nette.



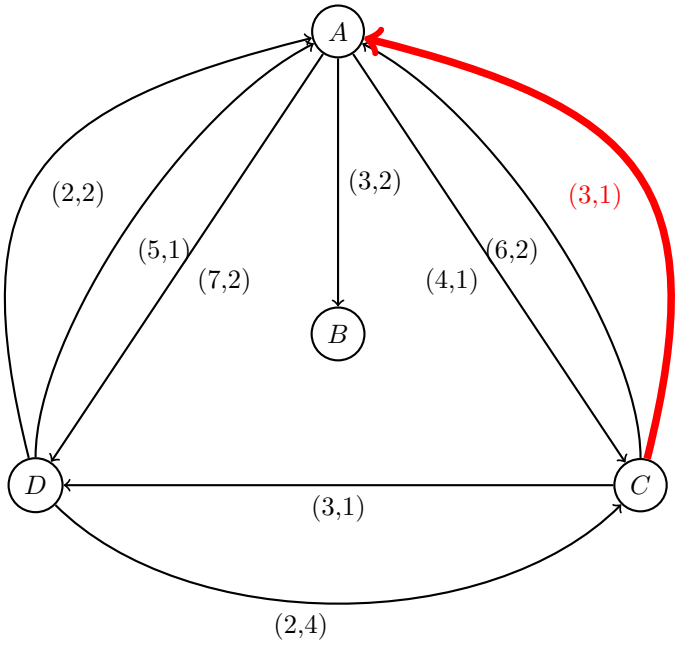
	A	B	C	D
I	0	0	0	0
B	0	0	0	0

$Q = 5$



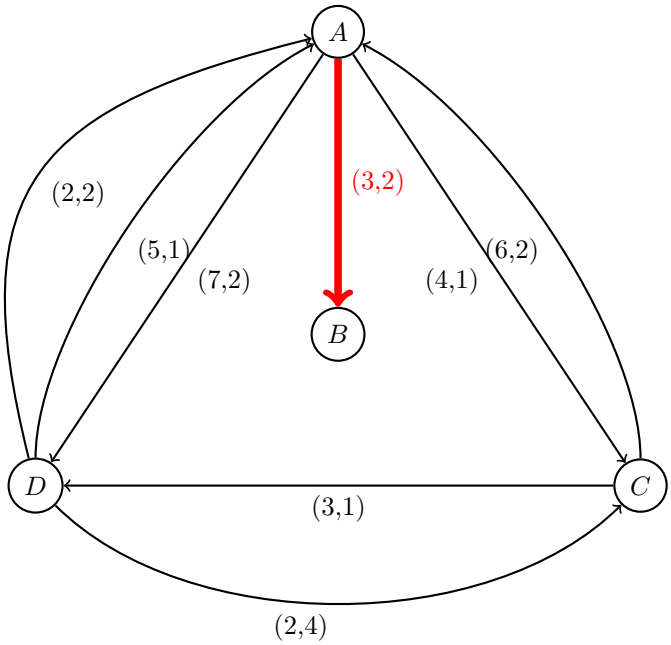
	A	B	C	D
I	0	5	0	0
B	0	0	5	0

$Q = 0$



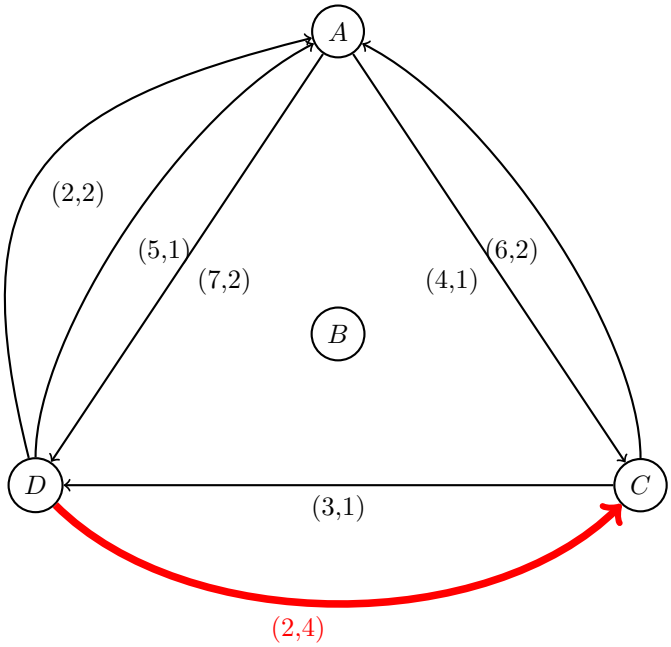
	A	B	C	D
I	0	5	0	0
B	3	0	2	0

$Q = 0$



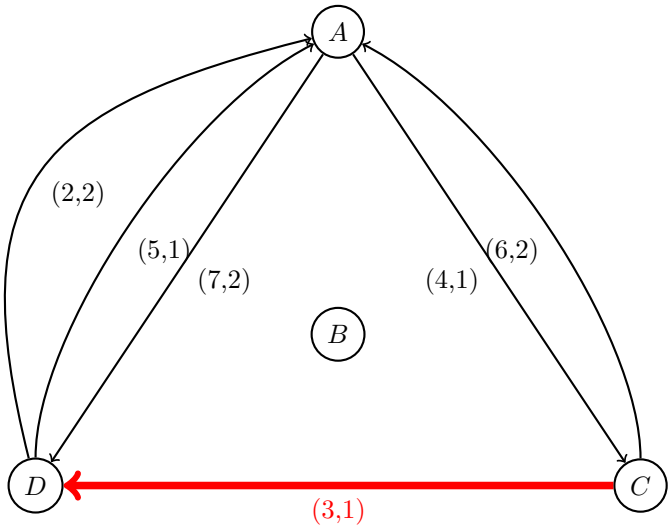
	A	B	C	D
I	0	2	0	0
B	0	0	2	0

$Q = 3$



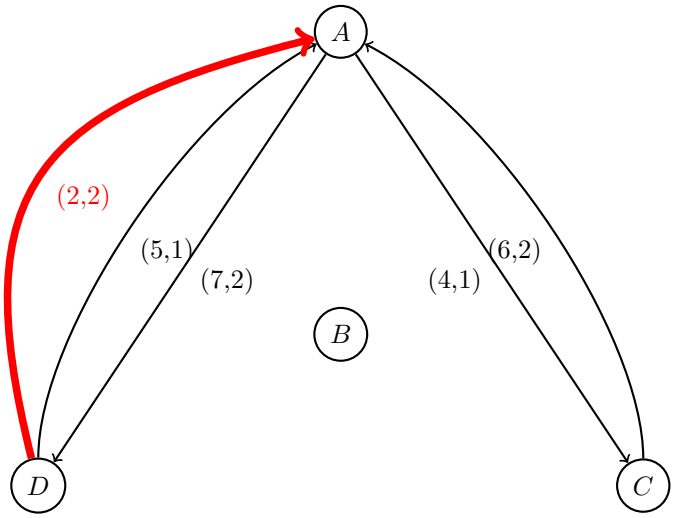
	A	B	C	D
I	0	2	0	2
B	0	0	4	0

$Q = 1$



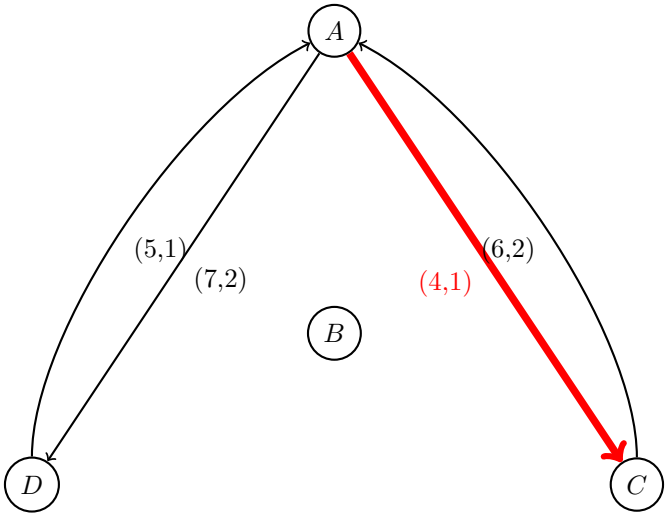
	A	B	C	D
I	0	2	0	0
B	0	0	1	1

$Q = 3$



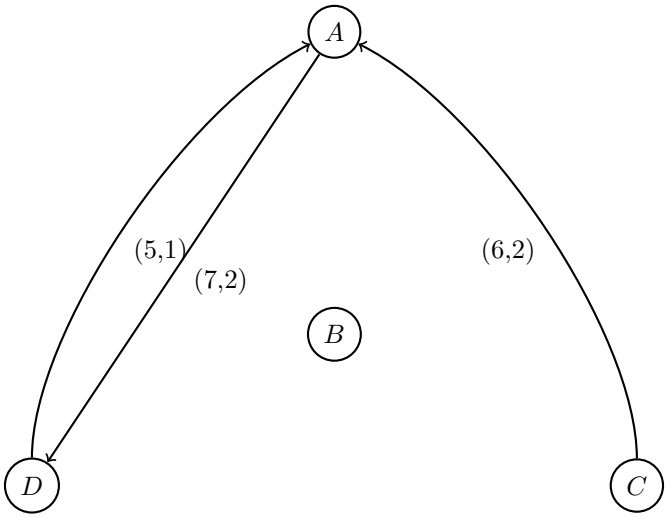
	A	B	C	D
I	0	2	0	1
B	2	0	1	0

$Q = 2$



	A	B	C	D
I	2	2	0	1
B	0	0	5	0

$Q = 0$



	A	B	C	D
I	2	2	0	1
B	0	0	5	0

$Q = 0$

On voit bien que l'algorithme glouton n'est pas optimal.

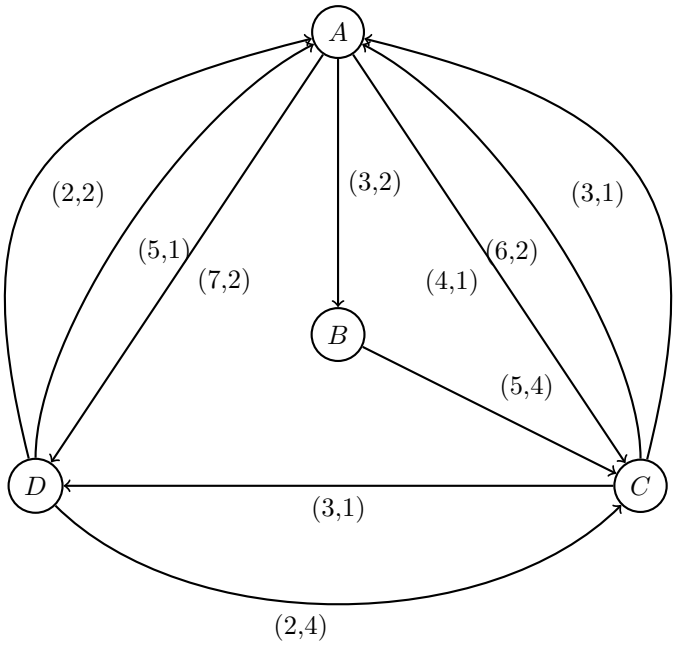
3.3 Améliorations et difficultés soulevées

Pour améliorer l'algorithme glouton, on pourrait cherche des chemins dans le graphe tels que le rapport entre la dette pondérée supprimée et la somme à injecter pour éliminer la dette le long du chemin est maximal, mais

cette recherche est exponentielle en le nombre d'arêtes. Une recherche de cycles dans le même but est déjà plus réaliste, même si elle reste inaccessible au vu de la taille des graphes étudiés.

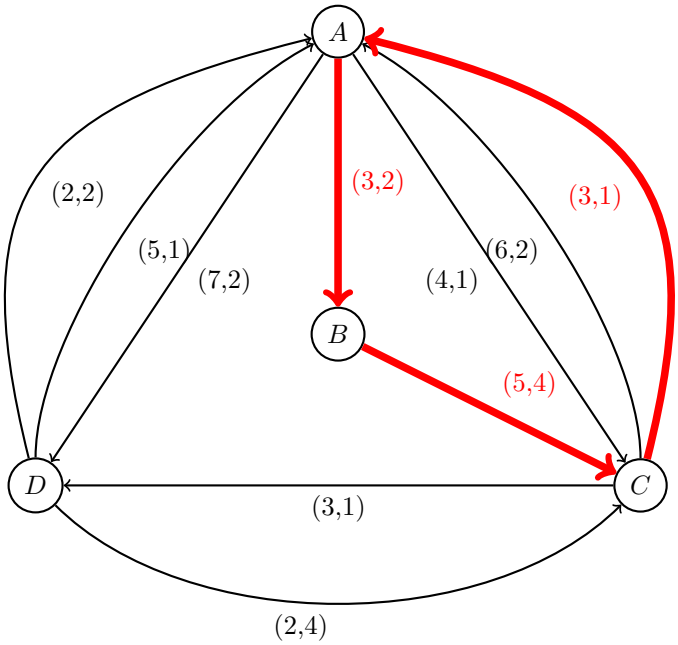
3.4 Application de l'algorithme glouton amélioré

On se propose d'appliquer l'algorithme amélioré au même graphe que précédemment. Ainsi, on cherche le chemin optimal est en rouge. On prend cette fois $Q = 2$, et on va avoir de bien meilleures performances que précédemment. On ne peut par ailleurs plus compenser toute la dette du graphe.



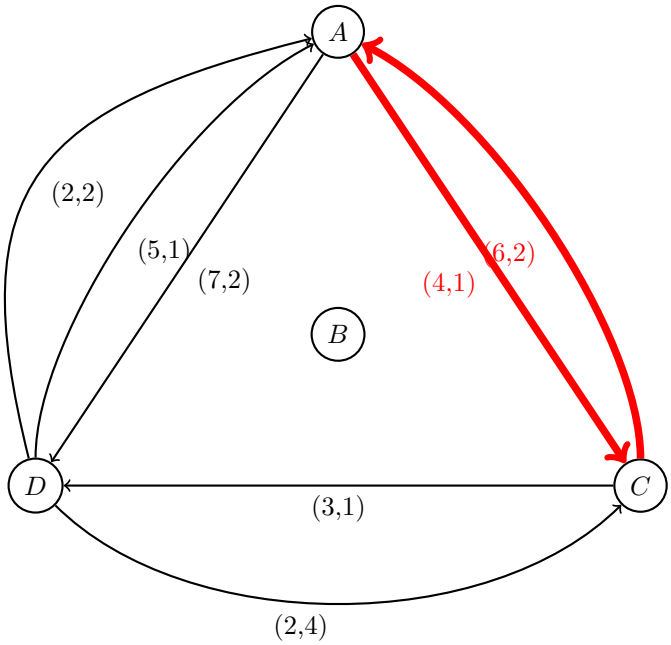
	A	B	C	D
I	0	0	0	0
B	0	0	0	0

$Q = 2$



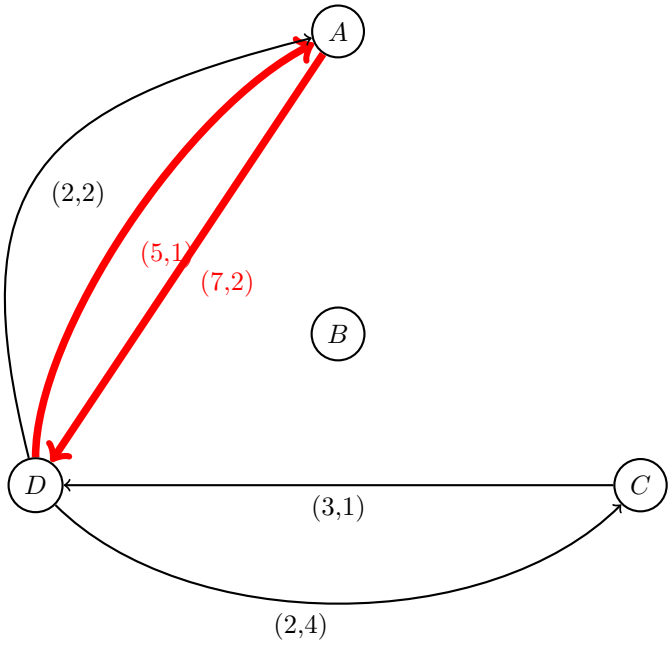
	A	B	C	D
I	0	2	0	0
B	0	0	2	0

$Q = 0$



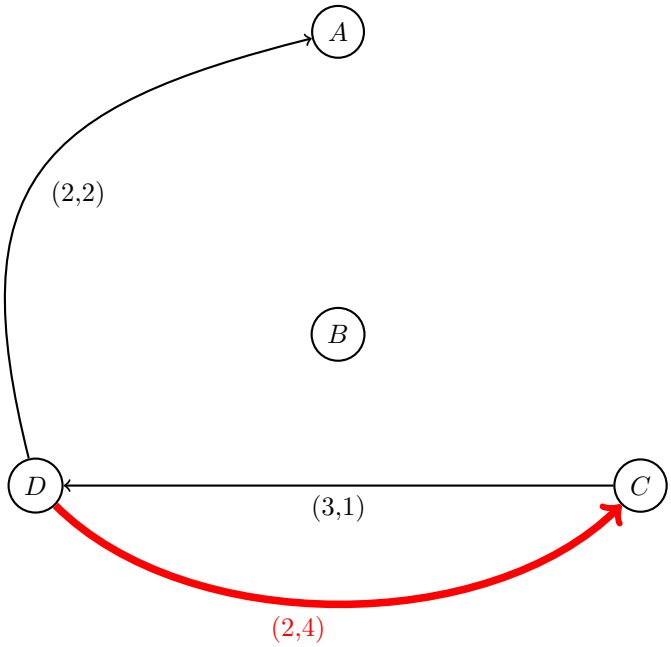
	A	B	C	D
I	0	2	0	0
B	2	0	0	0

$Q = 0$



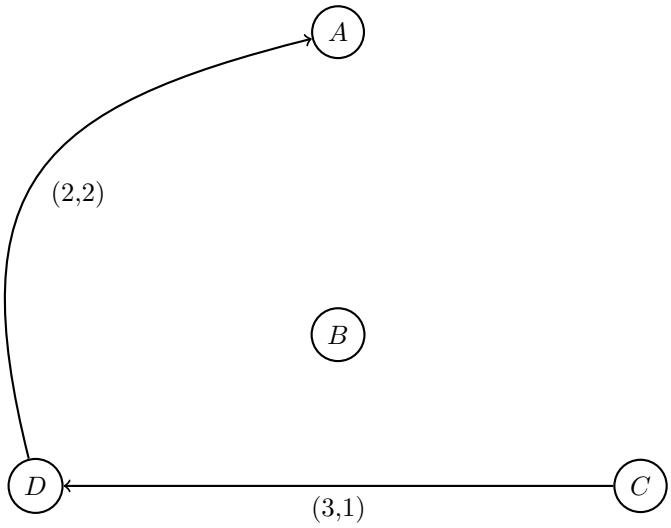
	A	B	C	D
I	0	2	0	0
B	0	0	0	2

$Q = 0$



	A	B	C	D
I	0	2	0	0
B	0	0	2	0

$Q = 0$



	A	B	C	D
I	0	2	0	0
B	0	0	2	0

$Q = 0$

On remarque que cette fois le résultat est optimal. Par NP-difficulté du problème SID, ce n'est pas toujours le cas, mais on a néanmoins trouvé un algorithme plutôt efficace. Ces pistes continueront à être étudiée lors du stage.