

Recherche opérationnelle (MAT\_1)  
L'optimisation en univers combinatoire (UA\_141)

Cours réalisé dans le cadre de l'ANR OpenMiage

François Delbot

11 janvier 2021



# Table des matières

<b>1</b>	<b>Introduction à la Recherche Opérationnelle</b>	<b>9</b>
1.1	Qu'est ce qu'un problème de décision ? . . . . .	9
1.2	Qu'est ce qu'un problème d'optimisation ? . . . . .	10
1.3	Démarche de résolution face à un problème de décision ou d'optimisation . . . . .	10
1.4	Qu'est-ce que la recherche opérationnelle . . . . .	11
1.4.1	Définition de la recherche opérationnelle . . . . .	11
1.4.2	Intérêt économique pour l'industrie . . . . .	12
1.4.3	Place de la Recherche Opérationnelle en France . . . . .	12
1.5	Exemple d'article accepté lors de la ROADEF 2019 . . . . .	13
1.6	Identifier les problématiques issues de problèmes concrets . . . . .	16
1.6.1	Un port de commerce . . . . .	16
1.6.2	Un aéroport . . . . .	17
1.6.3	Intensification du flux automobile . . . . .	18
1.6.4	Acheminement de l'électricité . . . . .	19
1.6.5	Agencement des circuits d'une carte mère . . . . .	20
1.6.6	Approvisionner des troupes . . . . .	21
<b>2</b>	<b>Théorie des graphes</b>	<b>23</b>
2.1	Origines de la théorie des graphes . . . . .	23
2.2	Les graphes, un outil de modélisation . . . . .	24
2.3	Définitions et concepts fondamentaux . . . . .	24
2.3.1	Définitions . . . . .	25
2.4	Exemple de modélisation d'un problème : Le loup, la chèvre et le chou . . . . .	31
2.5	Graphe d'intervalle . . . . .	32
2.5.1	Définition . . . . .	32
2.5.2	Déterminer si un graphe est un graphe d'intervalle . . . . .	33
2.5.3	Exemple (ludique) d'application . . . . .	35
<b>3</b>	<b>Exemples de familles de graphes</b>	<b>37</b>
3.1	Graphe complet à $n$ sommets . . . . .	37
3.2	Chemin . . . . .	38
3.3	Cycle . . . . .	38
3.4	Étoile . . . . .	39
3.5	Roue . . . . .	39
3.6	Arbre . . . . .	40
3.7	hypercubes . . . . .	40
3.8	Graphe biparti . . . . .	40
3.8.1	Graphe biparti complets . . . . .	41
3.8.2	Graphe biparti aléatoire . . . . .	41

3.9	Grille . . . . .	41
3.10	Graphe aléatoire de Erdös-Renyi . . . . .	42
3.11	Graphe petit monde . . . . .	43
<b>4</b>	<b>Résolution exacte et approchée, ainsi que quelques problèmes fameux</b>	<b>45</b>
4.1	Le problème du vertex cover . . . . .	45
4.1.1	Résolution exacte . . . . .	46
4.1.2	Résolution approchée . . . . .	54
4.2	Le problème de l'ensemble indépendant maximum(stable) . . . . .	56
4.3	Le problème de la clique maximum . . . . .	56
4.4	Le problème du voyageur de commerce . . . . .	56
4.5	Le problème des 4 couleurs . . . . .	57
<b>5</b>	<b>Parcours dans les graphes</b>	<b>59</b>
5.1	Atteignabilité, algorithme de Roy-Warshall . . . . .	59
5.1.1	Exemple de déroulement de l'algorithme. . . . .	60
5.1.2	Parcours en largeur . . . . .	61
5.1.3	Parcours en profondeur . . . . .	65
5.1.4	Exemple d'application de l'algorithme . . . . .	66
5.2	Plus courts chemins . . . . .	72
5.2.1	Un problème, trois versions . . . . .	72
5.2.2	Algorithme de Dijkstra . . . . .	73
5.2.3	Explications succinctes . . . . .	74
5.2.4	Exemple d'application de l'algorithme . . . . .	74
5.2.5	Tableau résumé de l'algorithme . . . . .	79
5.2.6	Algorithme de Bellman-Ford . . . . .	80
5.2.7	Explications . . . . .	81
5.2.8	Déroulement de l'algorithme de Bellman-Ford . . . . .	81
5.3	Comparaison des deux algorithmes . . . . .	87
5.3.1	Le temps d'exécution. . . . .	87
5.3.2	Graphes admissibles. . . . .	87
5.3.3	Conclusion. . . . .	88
<b>6</b>	<b>Problème de transports et flots</b>	<b>89</b>
6.1	Réseaux de transport et flots . . . . .	89
6.2	Coupe minimum . . . . .	91
6.3	Flot maximum . . . . .	93
6.3.1	Graphe résiduel . . . . .	93
6.3.2	Chemin augmentant . . . . .	93
6.3.3	Algorithme de Ford-Fulkerson . . . . .	93
6.3.4	Déroulement de l'algorithme . . . . .	93
6.4	Exemples d'applications des flots . . . . .	98
6.4.1	Déterminer des chemins indépendants . . . . .	98
6.4.2	Couplage maximum dans un graphe biparti . . . . .	99
<b>7</b>	<b>Programmation linéaire</b>	<b>101</b>
7.1	Exemple introductif . . . . .	101
7.1.1	Un problème de production . . . . .	101
7.1.2	Modélisation . . . . .	101
7.1.3	Évolution du modèle . . . . .	102

<b>7.2</b>	Modélisation par un programme linéaire . . . . .	102
7.2.1	Les variables . . . . .	102
7.2.2	La fonction économique . . . . .	103
7.2.3	Les contraintes . . . . .	103
7.2.4	Écriture générale d'un programme linéaire . . . . .	103
<b>7.3</b>	Linéarité et non-linéarité . . . . .	103
7.3.1	Exemples de programmes linéaires . . . . .	104
7.3.2	Exemples de programmes non linéaires . . . . .	104
<b>7.4</b>	Résolution graphique . . . . .	104
7.4.1	problème infaisable . . . . .	110
7.4.2	Solution non-bornée . . . . .	110
7.4.3	Solutions multiples . . . . .	111
7.4.4	Sommets et solutions. . . . .	112
<b>7.5</b>	Une évolution naïve de la résolution graphique . . . . .	112
7.5.1	Principe de l'algorithme . . . . .	112
<b>7.6</b>	Forme canonique . . . . .	113
7.6.1	Les règles de transformation . . . . .	113
7.6.2	Un exemple complet . . . . .	117
<b>7.7</b>	Forme normale . . . . .	120
7.7.1	Règles de transformation de la forme canonique vers la forme standard . . . . .	120
7.7.2	Équivalence entre les deux formes . . . . .	122
<b>7.8</b>	L'algorithme du simplexe . . . . .	123
7.8.1	Présentation de l'algorithme . . . . .	123
7.8.2	Méthode du dictionnaire . . . . .	123
7.8.3	Méthode des tableaux . . . . .	127
7.8.4	Exemple de déroulement de l'algorithme . . . . .	127
<b>8</b>	<b>Les outils de la recherche opérationnelle</b>	<b>131</b>
8.1	Pour la théorie des graphes . . . . .	131
8.2	Pour la programmation linéaire . . . . .	131
<b>9</b>	<b>Propositions de sujets de présentation</b>	<b>133</b>

## UA141 Optimiser en univers combinatoire

Ce cours a été réalisé dans le cadre de l'ANR OpenMiage. Il corresponds au module de recherche opérationnelle : optimisation en univers combinatoire. En particulier, il recouvre totalement l'UA 141 et en partie les deux UAs 142 et 143. Voici la liste des LOs couverts (avec l'UA à laquelle ils correspondent) :

1. [UA 141] **LO468** : *Identifier* les types de problèmes énumérables et leur complexité
2. [UA 141] **LO470** : *Comparer* différentes méthodes de résolution de la théorie des graphes et/ou de la programmation linéaire sur un problème donné
3. [UA 141] **LO471** : *Choisir* une méthode de résolution adaptée pour résoudre un problème énumérable
4. [UA 142] **N\_LO471\_1** : *Modéliser* Un problème énumérable par un graphe et/ou un programme linéaire
5. [UA 143] **N\_LO471\_2** : *Résoudre* Un problème énumérable simple par un algorithme de la théorie des graphes et/ou la résolution graphique et/ou l'algorithme du simplex
6. [UA 141] **LO472** : *Implanter* une méthode donnée pour résoudre un problème énumérable simple
7. [UA 141] **LO474** : *Utiliser* un outil/bibliothèque donné pour résoudre un problème énumérable donné concret
8. [UA 142] **N\_LO474\_1** : *Argumenter* de l'intérêt de l'utilisation d'une méthode de résolution approchées et/ou méta-heuristiques sur un problème donné

## Matériel pédagogique

Ce cours contient :

1. Un polycopié reprenant l'intégralité des notions abordées, des exemples de résolution ou de déroulement d'algorithmes.
2. Les supports de présentation orale. Chaque support correspond à un enseignement contextuel indispensable ou à une compétence particulière.
3. Des activités. Chaque activité peut être réalisée durant le cours ou à distance par l'étudiant seul. Dans tous les cas, l'étudiant doit être capable de réaliser à nouveau ces différentes activités de manière autonome.
4. Des évaluations formatives. Il s'agit de tests de positionnement que l'étudiant peut réaliser seul. Le corrigé est mis à sa disposition par la suite pour s'auto-évaluer.
5. Des évaluations sommatives. Il s'agit de l'évaluation permettant de valider l'enseignement.

# Déroulement de l'enseignement

Ce cours est conçu pour pouvoir être réalisé indifféremment sous la supervision d'un enseignant ou à distance.

## Objectifs

La recherche opérationnelle est un ensemble de techniques mathématiques qui, associées à l'informatique, permettent de formaliser, de modéliser et d'analyser les problèmes de décision complexes issus de l'industrie. On peut citer, par exemple, les problèmes de gestion et d'optimisation de la production, la réalisation d'emploi du temps, la localisation du lieu d'implantation d'une centrale énergétique ou l'approvisionnement d'une armée en territoire hostile. Dans de très nombreux cas, cela revient à étudier et résoudre des problèmes d'optimisation de nature combinatoire.

Ce cours a pour objectif de vous montrer comment faire preuve d'abstraction pour identifier les types de problèmes dits combinatoires, d'identifier leur complexité de résolution et de les modéliser. Puis, partant d'une modélisation, vous allez également apprendre à choisir le bon algorithme pour trouver une solution. Pour cela, nous allons étudier divers outils, comme la théorie des graphes ou la programmation linéaire, ainsi que les différents algorithmes qui leur sont propres.

## Modalités d'évaluation des apprentissages

1. **[Option 1 (présentiel)]** Présentation orale devant la promotion (durée 10 minutes,  $\frac{1}{3}$  de la note de contrôle continu)
2. **[Option 2 (non présentiel)]** Projet (théorique ou pratique,  $\frac{1}{3}$  de la note de contrôle continu)
3. Devoirs surveillés, tests en ligne, devoirs maison ( $\frac{2}{3}$  de la note de contrôle continu)
4. Examen final (durée 2h, 50% de la note finale)

## Activités d'enseignement-apprentissage

1. 36 heures en présentiel réparties de la manière suivante :
  - 18 heures de cours
  - 18 heures de travaux dirigés, dont 3 heures consacrées aux présentations.
2. Le projet, à réaliser seul et en non présentiel nécessite environ 10 heures de travail individuel.
3. La présentation orale, à préparer en binôme et en non présentiel, nécessite environ 5 heures de travail individuel.

## **Logistique nécessaire**

- Compilateur C ou similaire pour la théorie des graphes
- Python (2.x ou 3.x) pour l'utilisation d'une bibliothèque de modélisation et de résolution de programmes linéaires (PuLp).
- Un logiciel de conceptions de présentation (PowerPoint par exemple)

# Chapitre 1

## Introduction à la Recherche Opérationnelle

La Recherche Opérationnelle (RO) est une science née du besoin d'optimiser des problèmes liés aux organisation du monde réel (armée, industrie) faisant intervenir un grand nombre de variables et contraintes. Les techniques utilisées par la RO sont très diverses et, de ce fait, on peut dire que la RO est très ancienne. On peut citer par exemple Leonhard Euler (1707-1783) à qui l'on doit les bases de la théorie des graphes, ou Gaspard Monge (1746-1818) qui réalisa son mémoire sur la théorie des déblais et des remblais en 1781. Cependant, la communauté scientifique semble s'accorder sur le fait que les véritables débuts de la RO en tant que discipline scientifique ont eu lieu au début de la seconde guerre mondiale. D'ailleurs, le mot « Opérationnelle » provient du fait que les premières applications étaient liées aux opérations militaires. Parmi celles-ci, on peut citer par exemple le travail de Patrick Blackett (1897-1974), prix Nobel de physique en 1948, qui permit de diminuer de 20000 à 4000 (en moyenne) le nombre de balles nécessaires pour abattre un avion ennemi grâce aux défenses anti-aériennes [6]. Cette diminution reflétait le gain en efficacité des défenses obtenues par un meilleur positionnement des radars et des batteries de défense. Il serait très délicat de ne pas citer George Dantzig (1914-2005) à qui l'on doit l'algorithme du simplexe. Il s'agit d'un algorithme de résolution des problèmes d'optimisation linéaire. C'est probablement le premier algorithme permettant de minimiser une fonction sur un ensemble défini par des inégalités.

De nos jours, la recherche opérationnelle s'intéresse, par exemple, à l'organisation des lignes de production, à l'optimisation des télécommunications, aux problèmes de transport, à la production et au transport d'énergie, à la planification de projets/plannings, à l'ingénierie financière etc...

La recherche opérationnelle fait intervenir de nombreux outils mathématiques et informatiques tels que la programmation mathématique, la théorie des graphes, l'optimisation combinatoire et l'algorithme. Elle s'attache à la modélisation et à la découverte de solutions concrètes, pouvant être mises en pratique. La recherche opérationnelle étant un important levier pour réaliser des économies substantielles et maintenir la compétitivité, elle se retrouve de plus en plus impliquée (éventuellement de manière cachée) dans la prise de décision stratégique.

### 1.1 Qu'est ce qu'un problème de décision ?

Un problème de décision est un problème qui peut être posé comme une question pour laquelle la réponse, déterminée en fonction des données du problème, est soit « oui » soit « non ».

Un exemple de problème de décision consiste à décider si un nombre naturel donné est premier.

**Une méthode de résolution** d'un problème de décision, présentée sous la forme d'un **algorithme**, est appelée **procédure de décision** pour ce problème. Une procédure de décision doit donner les différentes étapes pour déterminer la réponse.

**Types de problèmes de décision.** Un problème de décision qui peut être résolu par un algorithme est dit « **décidable** », dans le cas contraire il est dit « **non décidable** ». Le domaine de la complexité classe les problèmes de décision décidables selon leur difficulté à résoudre. Cette difficulté, est décrite en termes de ressources de calcul (le nombre d'opérations élémentaires) nécessaires à l'algorithme le plus efficace pour un problème donné.

## 1.2 Qu'est ce qu'un problème d'optimisation ?

Un problème d'optimisation est une variante d'un problème de décision. Il consiste à trouver la meilleure solution parmi toutes les solutions possibles (ou l'une des meilleures si il y en a plusieurs). Pour un problème d'optimisation discrète (avec des variables discrètes), on recherche généralement un objet tel qu'un entier, une permutation, un sous ensemble, ou un graphe déterminé à partir des données du problème. Généralement, les problèmes d'optimisation se présentent de la manière suivante :

1. **Un énoncé.** Il s'agit d'un texte en langage naturel au sein duquel le problème est clairement exprimé, de manière non-ambiguë. Il indique généralement où trouver les données.
2. **Un ensemble de critères.** Différents critères, déduits de l'énoncé, permettent de comparer les solutions deux à deux.
3. **Un objectif.** Généralement exprimé mathématiquement, sous la forme d'une fonction (pouvant inclure différents critères). Dans un problème d'optimisation, il est nécessaire de préciser si on souhaite maximiser ou minimiser la valeur de cette fonction.
4. **Un ensemble de possibilités (solutions).** Pour un problème d'optimisation, la difficulté ne consiste pas à trouver une solution, mais à déterminer la meilleure (ou l'une des meilleures) solution suivant le critère d'optimisation. Car bien que toutes ces solutions puissent être déduites de l'énoncé, le nombre de ces solutions est tellement important qu'il n'est pas raisonnable de toutes les énumérer.

**Un exemple de problème d'optimisation** consiste à déterminer le plus court chemin (en temps), permettant de se rendre en voiture d'une ville *A* à une autre ville *B* tout en respectant les limitations de vitesse.

## 1.3 Démarche de résolution face à un problème de décision ou d'optimisation

Les problèmes rencontrés dans l'industrie ne sont pas directement de nature mathématiques. Cependant les outils scientifiques utilisent les mathématiques. Il est donc nécessaire, étant donné un problème, de le traduire dans un cadre mathématique, le résoudre, puis traduire la solution mathématique obtenue vers le réel. Pour résoudre un problème de décision ou d'optimisation, la démarche est très souvent la même :

- Exprimer le problème.** Cette étape peut sembler triviale, mais il arrive fréquemment que l'on exprime mal un problème, ou de manière imprécise. Cela rend sa résolution complexe, voire impossible. « Ce qui se conçoit bien s'énonce clairement, et les mots pour le dire arrivent aisément. »
- Récupération des données.** Pour calculer, il faut des données. Cette phase de récupération, qui peut sembler triviale elle aussi peut s'avérer extrêmement complexe. Pour de nombreuses entreprises, les données constituent une ressource précieuse qu'elles conservent jalousement.
- Modélisation du problème.** Un modèle est une simplification de la réalité. Son but est de permettre une correspondance entre la réalité et les mathématiques. Ainsi, résoudre un problème à partir d'un modèle permettra d'obtenir une solution concrète, encrée dans le réel. Il existe de nombreuses manières de modéliser un problème, et trouver la bonne manière (c'est à dire celle permettant de résoudre le problème le plus facilement) est une combinaison de logique, d'expérience et d'intuition. En particulier, si le problème est mal exprimé, la modélisation est délicate voire impossible.
- Choix ou conception de l'algorithme à appliquer.** Une fois le problème modélisé, il convient de trouver la bonne méthode de calcul, qui permettra de trouver la solution attendue. Si une telle méthode n'existe pas, alors il convient de la définir. Ces méthodes de calculs, aussi appelées « algorithmes » constituent une vaste boîte à outils que l'on peut adapter à la plupart des situations rencontrées. Parfois, plusieurs algorithmes peuvent fonctionner. Le choix de l'algorithme le plus adéquat doit se faire intelligemment, en fonction du contexte et des données.
- Analyse des résultats.** Les algorithmes permettent d'obtenir des résultats. Cependant, ces résultats ne sont pas toujours directement exploitables et peuvent nécessiter un travail supplémentaire pour faire correspondre les résultats mathématiques au réel. De plus, certains algorithmes permettent d'obtenir plusieurs solutions. Il faut donc analyser chacune de ces différentes solutions afin de choisir celle qui sera la plus adaptée au contexte.
- Décision.** Obtenir des solutions pour un problème est une chose. Mais il faut toujours garder à l'esprit que la décision qui sera prise ne sera pas obligatoirement en accord avec les résultats. La dimension politique n'est jamais loin et elle ne doit pas être négligée ...

## 1.4 Qu'est-ce que la recherche opérationnelle

Définir la recherche opérationnelle n'est pas une chose simple. Cela provient sûrement du nombre de disciplines scientifiques impliquées, du nombre important de communautés scientifiques disséminées dans le monde entier et du nombre d'outils utilisés dans l'industrie.

### 1.4.1 Définition de la recherche opérationnelle

Une vieille plaisanterie consiste, pour chaque communauté nationale, à définir la recherche opérationnelle de la manière suivante :

**Définition 1 (Recherche opérationnelle, pour rire)** *La recherche opérationnelle est ce qui est fait par les membres de la société de recherche opérationnelle et ses méthodes sont celles préconisées par le journal de cette société.*

Bien entendu, il est possible de trouver une définition sérieuse, valable et plus précise dans un bon dictionnaire :

**Définition 2 (Recherche opérationnelle, Larousse)** *Ensemble de techniques rationnelles d'analyse et de résolution de problèmes concernant notamment l'activité économique, visant à élaborer les décisions les meilleures possibles (au sens d'un ou de plusieurs critères) tout en respectant les contraintes inhérentes à ces problèmes.*

### 1.4.2 Intérêt économique pour l'industrie

Les entreprises sont nombreuses à utiliser la recherche opérationnelle. Cependant, on peut distinguer des usages très divers :

- Certaines grandes entreprises (EDF, Orange, l'armée française, SNCF ...) possèdent des entités spécifiques qui ont pour objectif l'amélioration de l'efficacité de la production ou de la qualité du service rendu aux clients. Ces entités sont, bien souvent, en contact avec le monde universitaire (séminaires, conférences, intégration de personnel dans des laboratoires publics, thèses CIFRE etc... ).
- Des entreprises de taille plus modeste n'auront pas nécessairement la capacité à conserver un tel service de manière permanente. Elles feront appel à des SSII spécialisées (Eurodécision), à des laboratoires universitaires, ou encore vont utiliser des logiciels développés par des éditeurs informatiques (ilog).

Résoudre un problème d'optimisation concret, issu de l'industrie, peut nécessiter l'utilisation de techniques parfois très complexes. Cela nécessite de réaliser un transfert de connaissances théoriques et d'adapter ces connaissances au réel. Cette problématique – tout à fait essentielle dans le monde industriel – concerne la gestion du compromis entre la qualité de solution requise, le temps nécessaire pour obtenir cette solution et la complexité des méthodes mises en œuvre.

### 1.4.3 Place de la Recherche Opérationnelle en France

La Recherche Opérationnelle a connu récemment un essor important dans notre pays. Elle s'est considérablement développée au cours des dernières années, beaucoup de postes universitaires ayant été mis au concours sur le profil R.O, au CNRS, à l'INRIA et dans les universités et écoles. L'ensemble de ces chercheurs se tiennent informés grâce à l'adhésion au Groupe De Recherche de Recherche Opérationnelle (GDR RO). Pour plus d'informations, vous pouvez consulter la pages suivante : <http://gdrro.lip6.fr/?q=node/6>

#### 1.4.3.1 La ROADEF

La communauté française de Recherche Opérationnelle est composée d'universitaires et d'industriels. Une partie de cette communauté fait partie de la ROADEF, qui est une association loi de 1901. Elle a pour objet de promouvoir l'enseignement, la recherche, la formation, l'application et la création de connaissances dans le domaine de la recherche opérationnelle et de l'aide à la décision (RO-AD). Vous trouverez des informations sur cette association ici :

1. <https://www.facebook.com/roadef/>
2. <https://www.roadef.org/societe-francaise-recherche-operationnelle-aide-decision>

En particulier, la ROADEF a publié un livre blanc présentant la place de la Recherche Opérationnelle en France. Vous êtes fortement invités à le consulter : <https://www.roadef.org/pdf/LivreBlancRO.pdf>

#### **1.4.3.2 Journées Franciliennes de Recherche Opérationnelle**

Ces journées ont pour objectif de permettre de rassembler la communauté de recherche opérationnelle de Paris et d'Ile de France (laboratoires de recherche et industriels) autour des thématiques émergentes tant au niveau théorique qu'à travers les applications. Compte tenu de la diversité et de la richesse des sujets relevant de la recherche opérationnelle, ainsi que du nombre d'équipes de recherche dans ce domaine, ces journées constituent un cadre idéal pour permettre à l'ensemble des participants de bénéficier des avancées et résultats récents en optimisation ainsi que de favoriser une synergie entre les équipes par la création d'un espace de communication et de discussion. L'objectif est double : pédagogique et scientifique. Dans ce cadre, chaque séance comporte une partie purement pédagogique où le conférencier expose les prérequis, fondamentaux et état de l'art du domaine sur lequel porte son exposé de recherche. La seconde partie consiste en la présentation des résultats.

1. Sous l'égide de la ROADEF.
2. <http://www.lamsade.dauphine.fr/~jfro/>
3. Organisée par de jeunes chercheurs du domaine.

L'inscription est gratuite. N'hésitez pas à vous y rendre, afin de rencontrer les participants qui peuvent vous conseiller sur l'orientation de vos études, et peut-être vous guider vers une thèse pour les plus passionnés d'entre vous.

### **1.5 Exemple d'article accepté lors de la ROADEF 2019**

Voici un exemple d'article [1] accepté lors du 20ème congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision. Cet article est le fruit d'une collaboration scientifique entre l'entreprise FRS consulting, le laboratoire EconomiX et le LIP6. Cette collaboration est concrétisée par une thèse CIFRE attribuée à Kymble CHRISTOPHE. Cet article traite d'un problème d'ordre économique, et est analysé du point de vue des relations entre les différents acteurs (modélisées par un graphe). La modélisation et les calculs nécessitaient l'intervention des informaticiens.

# Les perspectives du Brexit évaluées par les ensembles dominants

Valentin Bouquet<sup>1</sup>      Kymble Christophe<sup>2,3</sup>      François Delbot<sup>1</sup>  
Gaétan Le Chat<sup>3</sup>      Jean-François Pradat-Peyre<sup>1</sup>

<sup>1</sup> Université Paris-Nanterre / LIP6 UMR 7606

{valentin.bouquet,francois.delbot,jean-francois.pradat-peyre}@parisnanterre.fr

<sup>2</sup> Université Paris-Nanterre / Economix UMR 7235

<sup>3</sup> FRS consulting {kymble.christophe,gaetan.lechat}@frsconsulting.fr

**Mots-clés :** Ensemble dominant, réseaux collaboratifs, politique de financement européenne

**Introduction.** Le Programme Cadre pluriannuel (PC) créé en 1984 est rapidement devenu le principal instrument utilisé par la Communauté européenne (CE) pour réglementer, coordonner et soutenir la recherche et l'innovation (R&I) en Europe. L'idée sous-jacente est que la coopération et la collaboration en réseau sont des facteurs d'amélioration socio-économique. Le PC a pour objectif de réduire les écarts scientifiques et économiques entre les états membres, tout en favorisant la compétitivité des entreprises de l'Union européenne (UE)[1]. Avec le PC, plusieurs hypothèses sont formulées :

1. La recherche collaborative est plus efficace que la recherche isolée ;
2. les projets impliquant des entités hétérogènes (universités, instituts de recherche, entreprises, associations ou administrations publiques) ont plus de chances de réussir ;
3. idem pour les projets impliquant des acteurs situés dans différentes régions ou pays ;
4. multi-, pluri- et inter- disciplinarités sont essentielles pour réaliser des projets de recherche de portée internationale.

Ces hypothèses ont été prises en compte pour le 7ème Programme Cadre (FP7) et le suivant (H2020) dans les critères d'éligibilités pour tenter d'obtenir l'aide de l'UE. Les candidats doivent créer des consortiums intégrants des organisations privées et publiques d'au moins trois nationalités et statuts différents. Ces partenariats sont supposés apporter un environnement sécurisant, propice à l'innovation [2], permettant aux informations et connaissances de circuler plus librement. L'ampleur des PCs permet d'intégrer des organisations de la périphérie de l'UE dans l'espace européen de la recherche, tout en resserrant les liens entre les pays. Ceci est censé provoquer un phénomène "pop-up" par lequel les pays périphériques de l'UE seront économiquement et scientifiquement incités à rattraper le noyau. On obtient ainsi un réseau de collaboration à l'échelle de l'UE. L'analyse de ce réseau constitue un outil privilégié, car elle permet d'apprécier l'adéquation entre les objectifs de l'UE et la structure du réseau. De nombreux articles s'intéressent à la structure du réseau[3] et tentent d'identifier les agents centraux. Dans un travail précédent [4], nous avons montré que certaines structures de la théorie des graphes telles que l'ensemble dominant minimum (MDS) peuvent être utilisées pour déterminer quels membres sont les plus impliqués dans ces réseaux collaboratifs.

**Définition 1 (MDS)** Soit  $G = (V, E)$  un graphe non orienté, non pondéré (connecté ou non). Un ensemble dominant  $S \subseteq V$  de  $G$  est un ensemble de sommets tel que  $\forall v \in V - S$ ,  $N(v) \cap S \neq \emptyset$ . Un ensemble dominant minimum (MDS) est un ensemble dominant de taille minimum.

Un MDS peut-être vu comme la colonne vertébrale d'un réseau, permettant le transfert d'informations d'un membre du réseau à un autre. Bien que les réseaux issus des PCs soient de grande taille, leur structure rend possible le calcul d'un MDS optimal en un temps raisonnable. En particulier, nous avons montré que certains sommets sont présents dans toute solution optimale, et montré comment les calculer. Ces sommets (dits persistants) permettent de mieux

	FP7 - Organisations				H2020 - Organisations					
		A	$\Delta$	B	$\Delta$		A	$\Delta$	B	$\Delta$
Sommets	30438	14304	-53%	27225	-11%	20603	10243	50%	18703	-9%
Arêtes	721674	175291	-76%	584176	-19%	404008	115817	71%	330742	-8%
# composantes	23	39	70%	35	52%	203	206	1%	183	-10%
Taille du MDS	602	512	-15%	661	10%	768	640	-17%	771	0%
# persistants	270	192	-28%	286	6%	245	169	-31%	204	-17%

TAB. 1 – Impact des deux scenarii sur le réseau de collaboration pour FP7 et H2020

comprendre l'effet structurant de l'EU sur les collaborations entre organisations innovantes. Dans ce travail, nous poursuivons notre étude de la structure des réseaux de collaborations au moyen des MDS et des sommets persistants en évaluant l'impact du retrait du Royaume-Uni (RU) de l'UE et donc du réseau de collaboration.

**Méthodologie.** Nous considérons les projets financés par FP7 et H2020. Les données proviennent du service communautaire d'information sur la recherche et le développement (CORDIS) [5]. Nous avons généré 2 types de graphes à la fois pour FP7 et pour H2020 (simples, sans boucle et non orientés), soit un total de 4 graphes. Le premier type correspond aux relations entre les pays des différentes organisations. Deux pays  $A$  et  $B$  sont connectés par une arête ( $A, B$ ) si deux organisations sont impliquées dans un même projet, la première située dans le pays  $A$  et la seconde dans le pays  $B$ . Le second type (graphe des organisations), correspond aux relations entre les organisations impliquées dans les différents projets. Chaque sommet correspond à une organisation et deux organisations sont reliées par une arête si elles sont toutes les deux impliquées dans un même projet financé par le PC. Pour chacun de ces graphes, qui nous servent de références, nous avons évalué l'impact du retrait du RU en considérant deux scenarii. **Scenario A :** Suppression de toutes les organisations appartenant au RU ainsi que de tous les projets ayant au moins une organisation participante appartenant au RU. **Scenario B :** Suppression de toutes les organisations appartenant au RU. Les variations induites par ces deux scenarii sur le graphe des organisations sont synthétisées dans le tableau 1.

**Conclusion.** L'application de nos deux scenarii indique que malgré le Brexit, le réseau demeure propice à l'innovation avec une intégration beaucoup plus prononcée des pays périphériques. La part d'organisations issues de pays périphériques (i.e hors 15 plus riches européens) faisant partie des sommets persistants augmente passant de 15-17% avant Brexit à 33-37% en cas d'un Brexit dur (scenario A) et à 41-43% en cas de Brexit soft (scenario B). Cela est valable pour FP7 comme pour H2020.

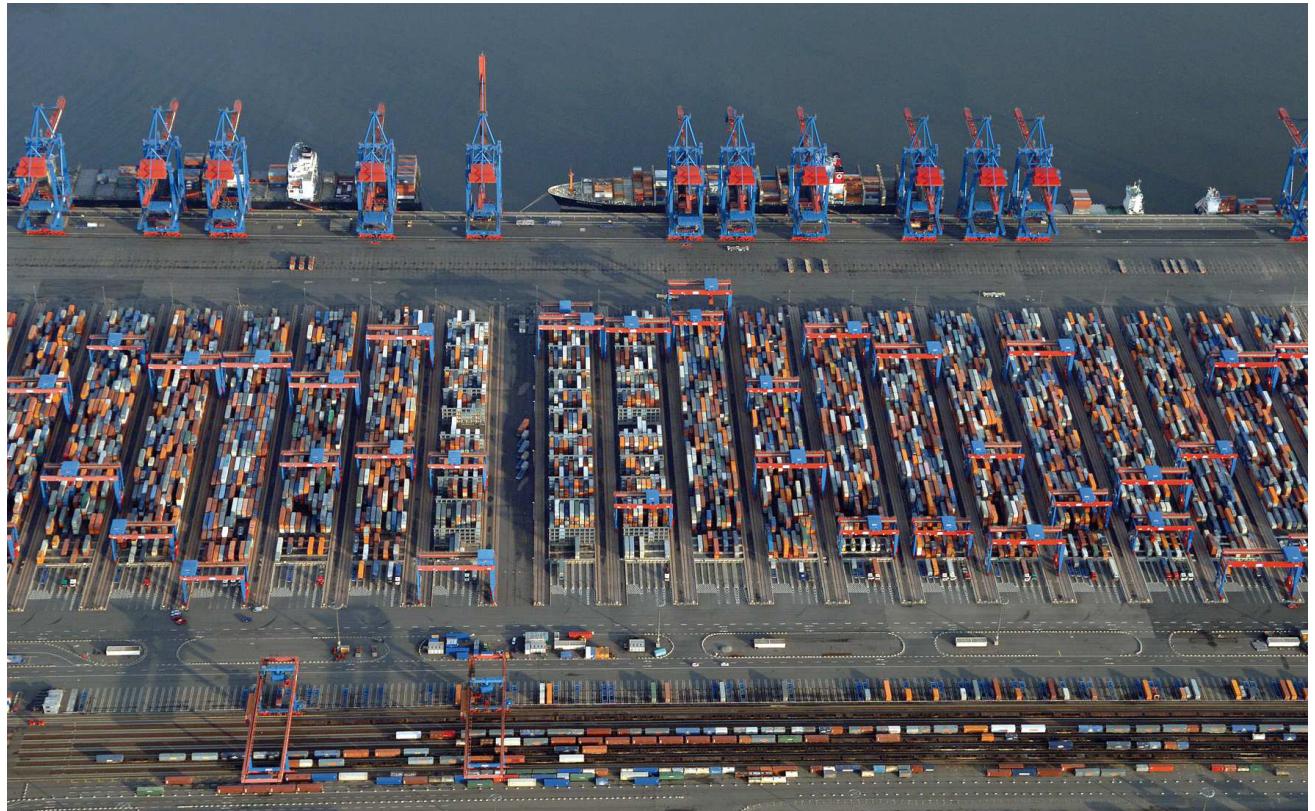
## Références

- [1] Breschi, S. and Malerba, F. : Clusters, networks and innovation. Oxford University Press (2005).
- [2] Foray : The patent system and the dynamics of innovation in Europe, pp. 449-456. Science and Public Policy, volume 31. (2004)
- [3] Malerba, F., Vonortas, N., Breschi, S. and Cassi, L. : Evaluation of progress towards a European Research Area for information society technologies. Report to European Commission, DG Information Society and Media (2006).
- [4] Bouquet, V., Christophe, K., Delbot, F., Le Chat, G., Pradat-Peyre, J.F. : Minimum Dominating Set and Maximum Independent Set for evaluation of EU funding policies in collaboration networks. OR 2018 : International Conference on Operations Research. (2018).
- [5] Community Research and Development Information Service. <https://data.europa.eu/euodp/data/>

## 1.6 Identifier les problématiques issues de problèmes concrets

Dans cette partie, vous devez identifier les différentes problématiques inspirées par les différentes images proposées. Ne vous restreignez pas, et n'hésitez pas à extrapoler. Après chaque image, quelques exemples sont donnés, à titre indicatif.

### 1.6.1 Un port de commerce



D'après cette image, on peut identifier un certain nombre de problématiques. En voici une liste non exhaustive :

1. Il est nécessaire de bien organiser les containers de manière à minimiser le coût, le temps de déplacement.
  - (a) Ordre de placement des arrivées et des départs prévus.
  - (b) Perte d'un conteneur : il faut prévoir qu'il y aura des erreurs.
  - (c) En cas de perte, comment retrouver un container ? A quel moment faudra-t-il lancer les recherches ?
2. Espace de stockage limité et nombre d'emplacements de chargement/déchargement limité.
  - (a) Quels bateaux servir en premier ? Quelle politique adopter ?
  - (b) premier arrivé, premier servi
  - (c) Celui qui va libérer le plus d'espace dans le port en premier ?
  - (d) Faire passer les petits en premier pour minimiser le temps d'attente ?
3. Quelle est la vitesse de déplacement d'un container ? Faut-il minimiser la somme des distances parcourue par les containers ?

4. Les camions et trains qui apportent les conteneurs consomment de la main d'œuvre. Il faut donc adapter la main d'œuvre au flux des marchandises.
5. Il faut minimiser le temps d'immobilisation des véhicules (camions, trains et cargos).

### 1.6.2 Un aéroport



D'après cette image, on peut identifier un certain nombre de problématiques. En voici une liste non exhaustive :

1. Où placer un aéroport : Quelle région ? Quel terrain ? Quel seront les impacts économiques, humains et écologiques ? Il y a des contraintes difficilement conciliables, qui peuvent imposer des compromis :
  - (a) Un aéroport doit être placé là où il gène le moins.
  - (b) Il doit aussi être très accessible aux voyageurs.
  - (c) Il y a des contraintes politiques.
  - (d) Il y a des contraintes liées à la sécurité. Par exemple, les phases de décollage et d'atterrissement sont les phases les plus critiques d'un vol. Les couloirs aériens doivent tenir compte d'un éventuel accident afin de limiter les dégâts.
  - (e) Il est interdit aux avions de survoler une centrale nucléaire.
2. La place au sol est limitée :
  - (a) Afin de maximiser le nombre d'avions au sol pour débarquer les passagers simultanément, les terminaux des aéroports ont pris cette forme circulaire.

- (b) Cette forme ne résout pas tous les problèmes. Il faut ordonner l'ensemble des arrivés et départs pour éviter de saturer l'aéroport.
  - (c) Les avions n'ont pas tous la même priorité pour atterrir (niveau de carburant, contrainte matérielle...).
3. Afin de minimiser les temps d'attente lors de l'embarquement et du débarquement (et ainsi maximiser le flux des passagers), ainsi que les plaintes, une logistique sans faille pour la gestion des bagages est nécessaire.
  4. L'aéroport doit être correctement desservi afin de minimiser les phénomènes d'engorgement (embouteillage).

Pour plus d'informations concernant les problématiques liées aux aéroports, je vous conseille de visionner les vidéos suivantes :

1. <https://www.youtube.com/watch?v=GNCeq4Jy074&t=736s>
2. <https://www.youtube.com/watch?v=22MQOPYHtSs>
3. <https://www.dailymotion.com/video/x12vh9>

### 1.6.3 Intensification du flux automobile



D'après cette image, on peut identifier un certain nombre de problématiques. En voici une liste non exhaustive :

1. Le flux automobile est de plus en plus dense. Cela occasionne des ralentissements. Par exemple, un conducteur parisien va, en moyenne, passer une soixantaine d'heures dans les bouchons. Cela représente un coût important, tant individuellement qu'à l'échelle de la société. Ainsi, fluidifier le trafic revient à minimiser ce coût.

2. Il est nécessaire de gérer ce flux. Mais les infrastructures ont un coût élevé. Il faut donc choisir le meilleur compromis entre le coût des embouteillages et le coût des infrastructures.
3. Les outils tels google maps ou mappy peuvent tenir compte de la densité du flux afin de vous indiquer le chemin le plus court en temps plutôt que le plus court en distance. A grande échelle, cela permet de détourner une partie du flux et de maximiser le débit du réseau routier.
4. Des travaux risquent de réduire la capacité d'un axe routier. Est-il possible de retarder ces travaux ? A quelle période de l'année, de la semaine, du jour les placer ? Comment détourner efficacement le flux ?
5. Les intersections sont souvent la source d'embouteillages et/ou d'accidents. Cette vidéo en présente quelques uns ainsi que leur impact sur la densité du flux automobile :<https://www.youtube.com/watch?v=T5Rayj0y0eg>

#### 1.6.4 Acheminement de l'électricité



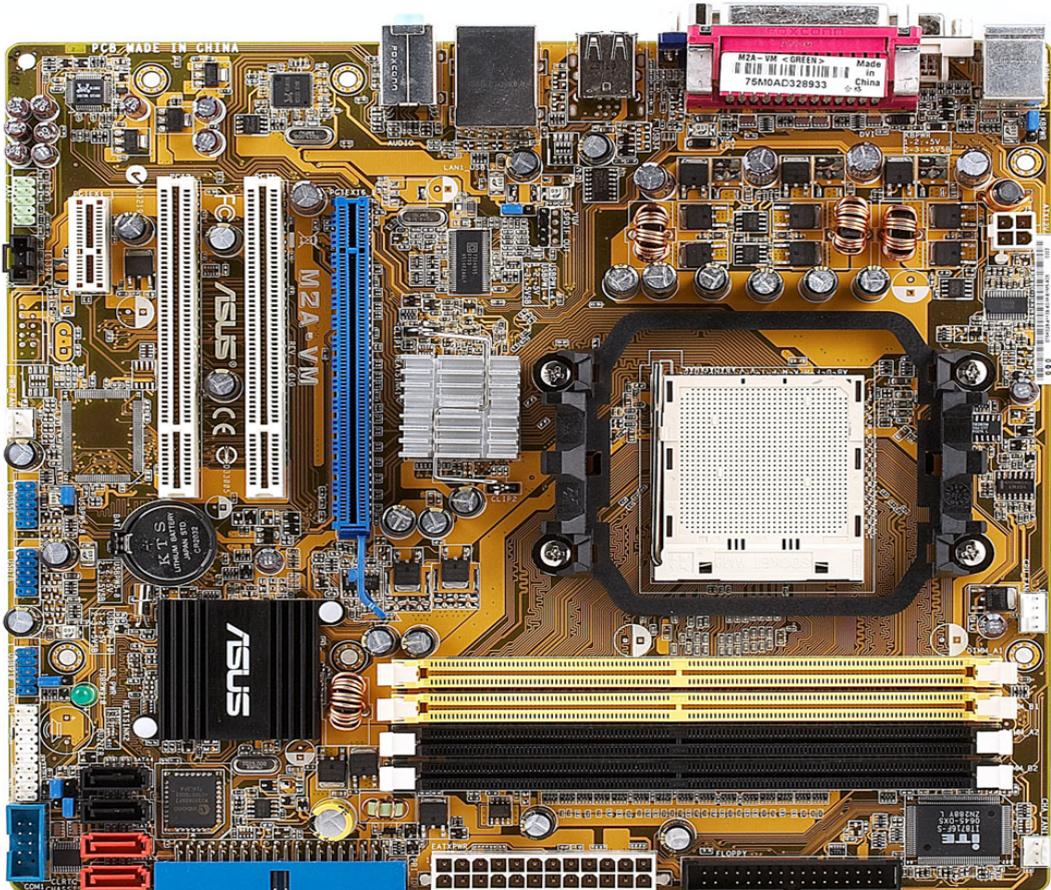
D'après cette image, on peut identifier un certain nombre de problématiques. En voici une liste non exhaustive :

1. L'énergie électrique se dissipe sous forme de chaleur. Cela limite la rentabilité sur de longues distances. Il faut donc un maillage efficace sur le territoire. Le réseau électrique doit aussi être le plus direct possible entre producteur et consommateur.
2. L'ensemble de la population doit pouvoir accéder à l'énergie électrique.
3. Les différents types de centrale ne démarrent pas à la même vitesse. Chaque type de centrale n'a donc pas le même rôle (production stable sur de longues périodes, ou besoin instantané).

4. Le coût de l'énergie produite varie en fonction du type de centrale, des ressources nécessaires (prix du charbon) et peut même être polémique (Le coût du démantèlement d'une centrale nucléaire n'est pas connu de manière exacte).
5. Les gens ne souhaitent pas habiter près d'une ligne haute tension ni à coté d'une centrale, quelle que soit son type. L'enfouissement du réseau électrique n'est pas toujours possible (sols gelés au canada, relief accidenté, ... et peuvent augmenter le coût d'une intervention).
6. Énergies vertes :
  - (a) Au printemps 2017, la ville de Lille a connue 404 heures d'ensoleillement cumulé, contre 753 heures de moyenne nationale. Le rendement énergétique d'une ferme solaire ne sera pas optimale.
  - (b) De même, une éolienne n'aura pas nécessairement le même rendement suivant la région où elle est placée.
  - (c) Les variations brusques de rendement posent des problèmes au réseau et risquent de le faire tomber : [https://fr.wikipedia.org/wiki/Liste\\_de\\_pannes\\_de\\_courant\\_importantes#2006,\\_Europe](https://fr.wikipedia.org/wiki/Liste_de_pannes_de_courant_importantes#2006,_Europe)

Une très bonne vidéo reprenant l'évolution du réseau électrique français peut être vue ici : <https://www.youtube.com/watch?v=4K1KHVOVEjc>. Les différentes problématiques vues ici sont sous-jacentes. Et une seconde vidéo expliquant l'équilibre imposé entre production et consommation : <https://www.youtube.com/watch?v=rMwuReV9DXk>.

#### 1.6.5 Agencement des circuits d'une carte mère



Vous trouverez dans cette vidéo une bonne introduction sur la manière de produire les composants électroniques : <https://www.youtube.com/watch?v=oq601550K3w>. Après l'avoir visionné, on peut identifier un certain nombre de problématiques. En particulier :

1. De très nombreux composants qui doivent communiquer entre eux.
2. Une miniaturisation de plus en plus importante pour aller le plus vite possible tout en prenant le moins de place possible.
3. Une partie du courant se dissipe sous forme de chaleur et cela risque de faire fondre différents composants. Il faut donc minimiser la somme des distances entre les différents composants.

Minimiser la somme des distances entre les composants tout en minimisant la surface utilisée est un problème d'optimisation combinatoire classique, mais pour lequel il est extrêmement difficile de trouver une solution optimale, du fait du grand nombre de variables.

#### 1.6.6 Approvisionner des troupes



D'après cette image, on peut identifier un certain nombre de problématiques. En voici une liste non exhaustive :

1. Comment approvisionner des troupes en nourriture, munitions, et armes sans rupture de stocks ? Des vies sont en jeu.
2. Des millions de litres de carburant sont consommés chaque jour. Et les troupes sont mobiles. Comment acheminer le carburant jusqu'aux véhicules ? Un pont aérien n'est pas toujours possible et un approvisionnement terrestre peut se faire intercepter facilement. Un convoi pillé est une chose grave puisque les troupes ne sont pas approvisionnées et les ennemis disposent des ressources volées.

3. Des erreurs se produisent. Par exemple, un bataillon reçoit les munitions dont il avait besoin, ainsi que celles d'un autre bataillon. L'autre bataillon ne reçoit rien. Peut-on trouver une méthode qui diminue fortement les erreurs ? Si diminuer les erreurs n'est pas réalisable, comment s'assurer que les troupes reçoivent tout de même ce qu'il faut ?

## Chapitre 2

# Théorie des graphes

Ce cours à pour objectif de vous familiariser avec la théorie des graphes. Pour ceux qui souhaitent en découvrir d'avantage, je vous conseille très fortement la chaîne youtube de Christian Laforest, Professeur des Universités (ISIMA/LIMOS) : [https://www.youtube.com/channel/UCHtJVeNLyR1yuJ1\\_xCK1WRg/featured](https://www.youtube.com/channel/UCHtJVeNLyR1yuJ1_xCK1WRg/featured)

### 2.1 Origines de la théorie des graphes

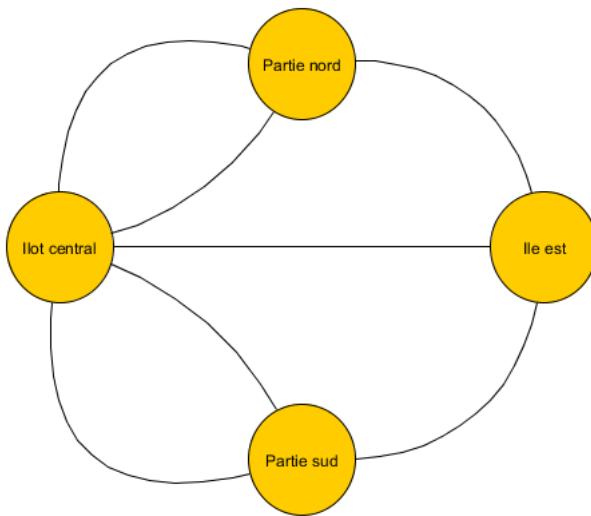
La ville de Königsberg (aujourd'hui Kaliningrad) est au centre d'un problème historique. Cette ville est composée de 4 parties : la partie nord, la partie sud, l'ilot central et l'île est. L'îlot central est relié par deux ponts à la partie nord, par deux ponts à la partie sud et par un pont à l'île est. L'île est est reliée à la partie nord par un pont et à la partie sud par un autre pont, pour un total de 7 ponts. Au XVII<sup>e</sup> siècle, les habitants se querellaient sur l'existence ou non d'une promenade permettant de partir de l'une des parties de la ville, de passer une fois, et une fois seulement par chaque pont, tout en retournant en fin de promenade sur la même partie de la ville. Ce problème est connu comme celui des **7 ponts de Königsberg**.



C'est le mathématicien Leonhard Euler qui a apporté la preuve qu'il n'existe pas de solution à ce problème. Une intuition de la preuve consiste à représenter les ponts horizontalement et

à les trier de manière à ce que deux extrémités adjacentes de deux ponts successifs soient sur le même terrain (à savoir la partie nord, la partie sud, l'ilot central ou encore l'île est), et en considérant que le premier et le dernier soient consécutifs, puisqu'on doit retourner au point de départ. Ainsi tout terrain est-il obligatoirement incident à un nombre pair de ponts (puisque si le terrain est incident à un pont, il est aussi incident au précédent ou bien au suivant dans notre tri.). Mais la ville de Königsberg possède des terrains (L'ilot central et la partie nord) qui sont incidents à trois ponts, d'où l'impossibilité.

Cette idée de preuve montre que l'on peut se passer de la configuration exacte de la ville. L'important étant de connaître les différentes parties de la ville et les liens (les ponts) permettant de passer de l'une à l'autre. C'est l'idée sous-jacente à toute la théorie des graphes. Le problème peut donc se modéliser ainsi :



Dans ce modèle, les différentes parties de la ville sont représentées par des ronds, et les ponts par des traits. Ce dessin est ce qu'on appelle un graphe.

## 2.2 Les graphes, un outil de modélisation

Un graphe est un outil permettant de modéliser de nombreuses situations où des objets sont en interaction. Malgré sa simplicité apparente (des ronds et des traits), un graphe permet d'utiliser un grand nombre de techniques, outils mathématiques et algorithmes issus de la théorie des graphes et des mathématiques pour démontrer des propriétés, déterminer des méthodes de résolution, etc.

**Exemples d'applications** : déterminer le plus court chemin entre deux villes, former des couples fortement compatibles sur un site de rencontre, routage de messages dans un réseau, régulation du flux des usagers dans le RER etc.

## 2.3 Définitions et concepts fondamentaux

**Convention.** Tout au long de ce document, nous désignerons lorsque cela ne crée pas d'ambiguïté par  $n$  le nombre de sommets d'un graphe, et par  $m$  le nombre d'arêtes.

### 2.3.1 Définitions

**Définition 3 (Graphe non orienté)** Un graphe non-orienté  $G = (V, E)$  est défini par un couple d'ensembles :  $V = \{v_1, v_2, \dots, v_n\}$  l'ensemble des sommets du graphe (vertices en anglais) et  $E = \{e_1, e_2, \dots, e_m\}$  l'ensemble des arêtes du graphe (edges en anglais).

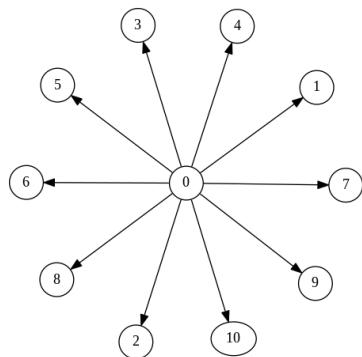
**Définition 4 (Arête)** Soit  $G = (V, E)$  un graphe non-orienté,  $u \in V$  et  $v \in V$  deux de ses sommets. Une arête  $e = (u, v) \in E$  est un couple de sommets et représente un lien entre ces deux sommets.  $u$  et  $v$  sont dits voisins ou adjacents.

FIGURE 2.1 – Exemple de graphe non-orienté 

- 10 sommets et 14 arêtes.
- Ensemble des sommets :  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Ensemble des arêtes :  $\{(0, 3), (0, 6), (0, 7), (1, 4), (1, 5), (2, 4), (2, 9), (3, 4), (3, 5), (4, 6), (4, 7), (5, 9), (6, 8), (7, 8)\}$

**Définition 5 (Graphe orienté)** Un graphe  $G = (V, A)$  est défini par un couple d'ensembles :  $V$  l'ensemble des sommets du graphe et  $A$  l'ensemble des arcs du graphe. La différence avec un graphe non-orienté est que les liens (les arcs) sont directionnels et ne peuvent donc être parcourus que dans un seul sens.

**Définition 6 (Arc)** Soit  $G = (V, A)$  un graphe orienté,  $u \in V$  et  $v \in V$  deux de ses sommets. Un arc  $e = (u, v) \in A$  est un couple de sommets et représente un lien directionnel allant du sommet  $u$  vers le sommet  $v$ .  $u$  est le prédécesseur de  $v$  et  $v$  est le successeur de  $u$ .

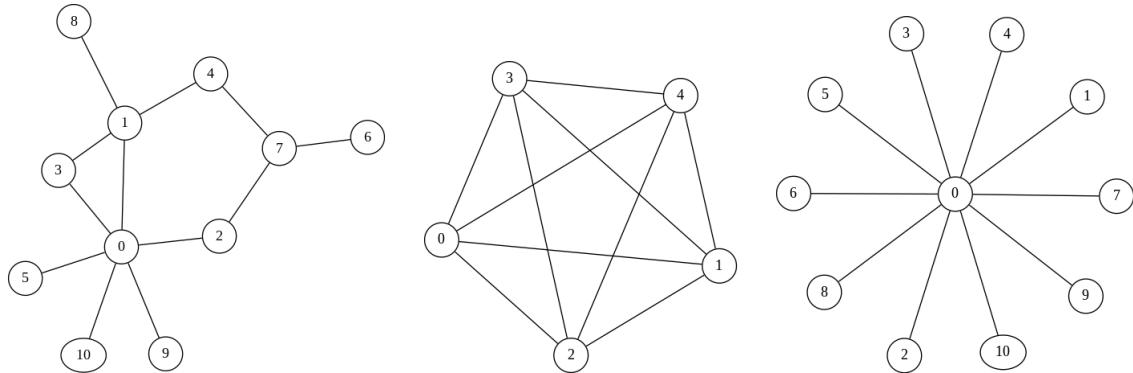


- Ensemble des sommets :  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- Ensemble des arcs :  $\{(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10)\}$
- le sommet 0 est le prédécesseur de tous les autres sommets.
- Les sommets 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 sont les successeurs du sommet 0.

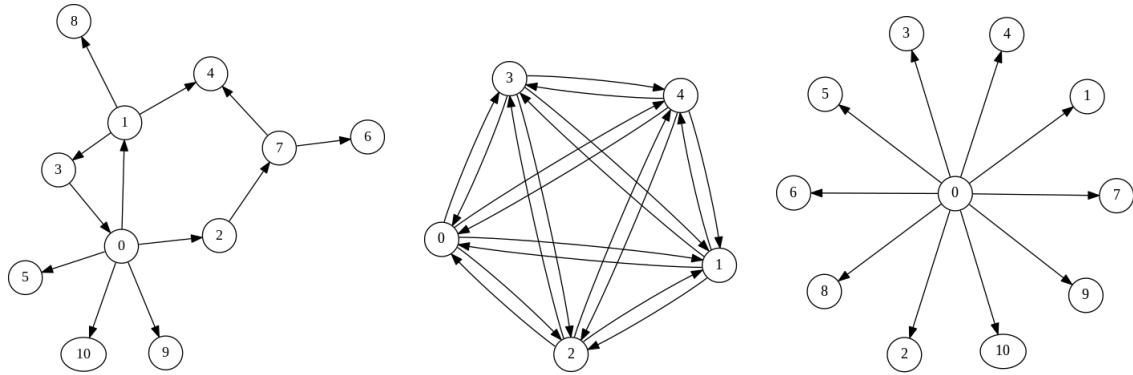
FIGURE 2.2 – Exemple de graphe orienté

Il est important de noter que dans le cas d'une arête  $(u, v)$  l'ordre des sommets n'a pas d'importance puisque le lien n'est pas directionnel. Parler de l'arête  $(u, v)$  revient à parler de l'arête  $(v, u)$ . Au contraire, dans le cas d'un arc  $(u, v)$  l'ordre est important puisqu'il implique une relation allant de  $u$  vers  $v$  mais pas l'inverse.

### 2.3.1.1 Quelques exemples de graphes non orientés :



### 2.3.1.2 Quelques exemples de graphes orientés :



**Définition 7 (Taille d'un graphe)** La taille d'un graphe correspond au nombre de sommets du graphe.

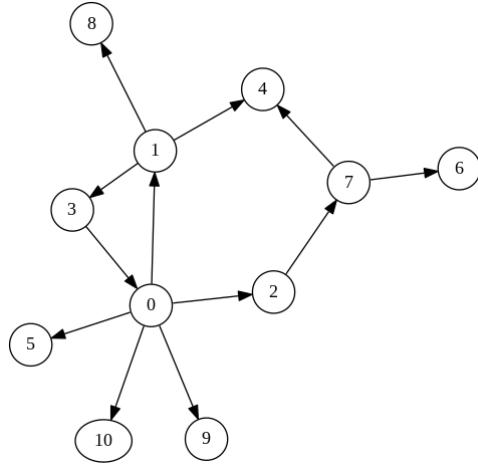


FIGURE 2.3 – Exemple de graphe orienté

Par exemple, le graphe de la figure 2.3 contient 11 sommets. Il est donc de taille 11.

**Définition 8 (Voisinage sortant)** Soit  $G = (V, A)$  un graphe orienté et  $u \in V$  l'un de ses sommets. Le voisinage sortant de  $u$ , noté  $N^+(u)$ , est l'ensemble des sommets ayant  $u$  comme prédecesseur.

Par exemple, le sommet 7 du graphe de la figure 2.3 a pour voisinage sortant :

$$N^+(7) = \{4, 6\}$$

**Définition 9 (Voisinage entrant)** Soit  $G = (V, A)$  un graphe orienté et  $u \in V$  l'un de ses sommets. Le voisinage entrant de  $u$ , noté  $N^-(u)$ , est l'ensemble des sommets ayant  $u$  comme successeur.

Par exemple, le sommet 7 du graphe de la figure 2.3 a pour voisinage entrant :

$$N^-(7) = \{2\}$$

**Définition 10 (Voisinage d'un sommet)** Soit  $G = (V, E)$  un graphe et  $u \in V$  l'un de ses sommets. Le voisinage de  $u$ , noté  $N(u)$ , est l'ensemble des sommets partageant une arête avec  $u$ . Si le graphe est orienté,  $N(u) = N^+(u) \cup N^-(u)$ .

Par exemple, le sommet 7 du graphe de la figure 2.3 a pour voisinage :

$$N(7) = N^+(7) \cup N^-(7) = \{2, 4, 6\}$$

**Définition 11 (Graphe simple et multi-graphe)** Soit  $G = (V, E)$  un graphe. Si pour tout couple de sommet  $(u, v)$  il existe au plus une arête possédant  $u$  et  $v$  comme extrémités, alors  $G$  est un graphe simple. La même définition s'applique pour les graphes orientés, où un arc du graphe ne peut apparaître plus d'une seule fois. Dans le cas contraire, on parle de multi-graphe.

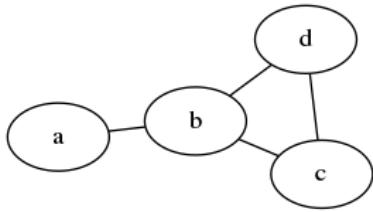


FIGURE 2.4 – Graphe simple

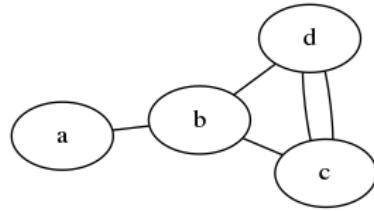


FIGURE 2.5 – Multi-graphe

**Définition 12 (Boucle)** Une boucle est une arête (ou un arc) d'un graphe pour laquelle les deux extrémités sont identiques.

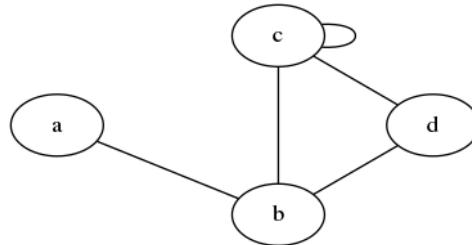


FIGURE 2.6 – Exemple de graphe contenant une boucle sur le sommet c

**Définition 13 (Degré d'un sommet)** Le degré d'un sommet  $u \in V$ , noté  $d(u)$  est le nombre d'arêtes incidentes au sommet  $u$ . Si le graphe est simple et non-orienté, alors  $d(u) = |N(u)|$ . Dans le cas d'un graphe orienté, on note  $d^+(u) = |N^+(u)|$  le degré sortant de  $u$ ,  $d^-(u) = |N^-(u)|$  le degré entrant de  $u$  et  $d(u) = d^+(u) + d^-(u)$  son degré.

- Le sommet 7 de la figure 2.3 à pour degré entrant  $d^-(7) = 1$ , degré sortant  $d^+(7) = 2$  et degré  $d(7) = 3$ .
- Le sommet b de la figure 2.5 a pour degré  $d(b) = 3$ .
- Le sommet c de la figure 2.5 a pour degré  $d(c) = 3$ .
- Le sommet c de la figure 2.6 a pour degré  $d(c) = 4$ .

**Définition 14 (Sommet isolé)** Un sommet sans aucun voisin est dit isolé.

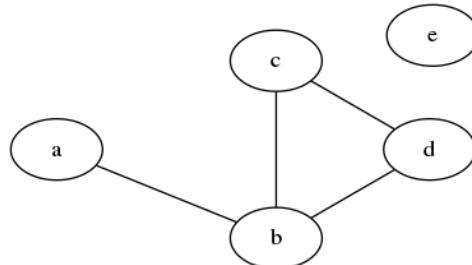


FIGURE 2.7 – Exemple de graphe pour lequel un sommet est isolé

En particulier, un sommet isolé à donc un voisinage vide et est toujours de degré 0.

**Définition 15 (Adjacence)** Soient  $u$  et  $v$  deux sommets d'un graphe. Ces deux sommets sont dits adjacents si il existe, dans le graphe, une arête ou un arc ayant  $u$  et  $v$  pour extrémités.

Par exemple, la figure 2.7 contient un graphe pour lequel les sommets  $c$  et  $d$  sont adjacents tandis que le sommet  $e$  n'est adjacent à aucun sommet.

**Définition 16 (Attributs)** Un graphe est un outil de modélisation très souple pouvant s'adapter à de nombreuses situations. En particulier, il est possible d'ajouter des attributs aux sommets, arcs ou arêtes d'un graphe. Par exemple, il est possible de pondérer (ajouter un poids, une valeur) à un sommet, un arc ou une arête. Si les arcs ou les arêtes sont pondérés, on parle de graphe pondéré.

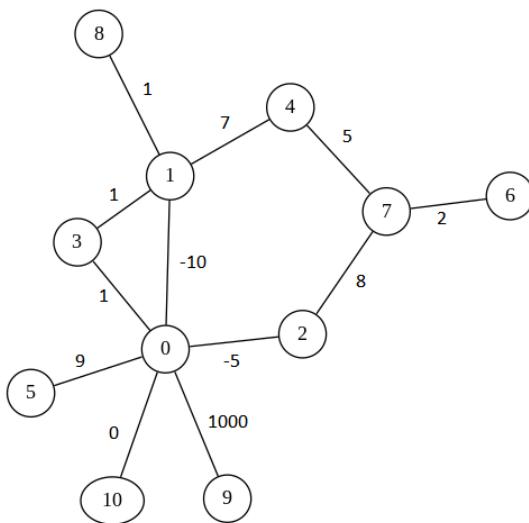


FIGURE 2.8 – Exemple de graphe pondéré

**Définition 17 (chemin, chaîne)** Un chemin (une chaîne si le graphe est non-orienté) reliant deux sommets  $u$  et  $v$  est une succession d'arcs (d'arêtes) consécutifs partant du sommet  $u$  et arrivant au sommet  $v$ .

1. Un chemin (chaîne) est dit élémentaire si il ne passe pas deux fois par le même sommet.
2. Un chemin (chaîne) est dit simple si il ne passe pas deux fois par le même arc (arête).

Par exemple, dans la figure 2.8, le graphe contient

- Une chaîne élémentaire reliant le sommet 8 au sommet 9 :  $(8, 1, 0, 9)$ .
- Une autre chaîne élémentaire reliant le sommet 8 au sommet 9 :  $(8, 1, 3, 0, 9)$ .
- Une chaîne non-élémentaire reliant le sommet 8 au sommet 9 :  $(8, 1, 3, 0, 1, 3, 0, 9)$ .

**Définition 18 (Longueur d'un chemin ou d'une chaîne)**

1. Si le graphe est non pondéré, la longueur d'un chemin (chaîne) correspond au nombre d'arcs (arêtes) traversés.
2. Si le graphe est pondéré, la longueur du chemin (chaîne) correspond à la somme des poids des arcs (arêtes) traversés.

Par exemple, dans la figure 2.8, les chaînes suivantes ont pour poids :

- La chaîne élémentaire reliant le sommet 8 au sommet 9 :  $(8, 1, 0, 9)$  à pour poids

$$1 - 10 + 1000 = 991$$

- Une autre chaîne élémentaire reliant le sommet 8 au sommet 9 :  $(8, 1, 3, 0, 9)$  à pour poids

$$1 + 1 + 1 + 1000 = 1003$$

- Une chaîne non-élémentaire reliant le sommet 8 au sommet 9 :  $(8, 1, 3, 0, 1, 3, 0, 9)$  à pour poids

$$1 + 1 + 1 - 10 + 1 + 1 + 1000 = 995$$

**Définition 19 (Taille d'un chemin ou d'une chaîne)** *La taille d'un chemin (chaîne) correspond au nombre de sommets appartenant à ce chemin (chaîne).*

**Définition 20 (Graphe connexe)** *Un graphe est connexe si, pour tout couple de sommets  $(u, v)$ , il existe un chemin (une chaîne si le graphe est non-orienté) allant de  $u$  vers  $v$ .*

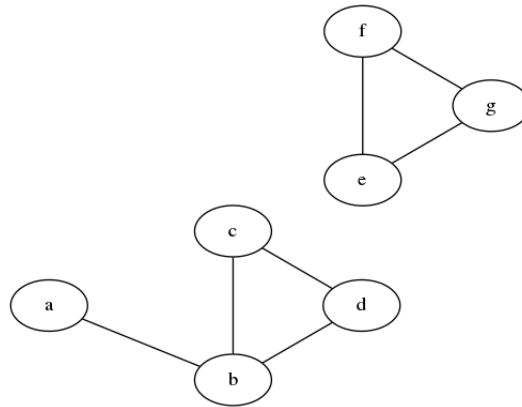


FIGURE 2.9 – Exemple de graphe non connexe

**Définition 21 (Composante (fortement) connexe)** *Une composante connexe  $CC \in V$  est un sous ensemble connexe et maximal des sommets du graphe. Le mot maximal signifie ici que si on ajoute un autre sommet de  $V$  à  $CC$ , alors  $CC$  ne sera plus connexe. Dans le cas d'un graphe orienté, si chaque sommet  $u$  de la composante connexe est atteignable par n'importe quel autre sommet  $v$  de la composante connexe (il existe un chemin allant de  $v$  vers  $u$ ), alors on dit qu'il s'agit d'une composante fortement connexe.*

Par exemple, le graphe de la figure 2.9 contient deux composantes connexes :

1.  $\{a, b, c, d\}$
2.  $\{e, f, g\}$

**Définition 22 (Cycle)** *Un cycle est un chemin (chaîne) de taille strictement supérieur à 1 partant et arrivant au même sommet.*

Par exemple, dans la figure 2.8, on peut trouver plusieurs cycles :

1.  $\{1, 3, 0\}$
2.  $\{4, 1, 0, 2, 7\}$

3.  $\{1, 3, 0, 2, 7, 4\}$

**Définition 23 (Ensemble indépendant)** *Un ensemble indépendant est un sous ensemble des sommets du graphe tel qu'aucun des sommets de cet ensemble ne partage une arête (arc) avec un autre sommet de cet ensemble.*

Par exemple, le graphe de la figure 2.9 contient plusieurs ensembles indépendants dont :

1.  $\{a, c, g\}$
2.  $\{b, f\}$

**Définition 24 (Clique)** *Une clique est un sous ensemble des sommets du graphe tel que tous les sommets de cet ensemble sont reliés à tous les autres sommets de cet ensemble.*

Par exemple, le graphe de la figure 2.9 contient deux cliques de taille 3 :

1.  $\{b, c, d\}$
2.  $\{e, f, g\}$

## 2.4 Exemple de modélisation d'un problème : Le loup, la chèvre et le chou

Un berger, en compagnie d'un loup (qu'il souhaite dresser), d'une chèvre (pour le lait) et d'un chou (son repas du soir), souhaite traverser une rivière. Pour cela, il dispose d'une petite barque qui ne peut contenir que lui-même ainsi qu'un et un seul autre légume ou animal. Malheureusement, laisser le loup sans surveillance avec la chèvre signifie la fin de cette pauvre bête. De la même manière, laisser la chèvre seule avec le chou priverait le berger de son repas du soir.

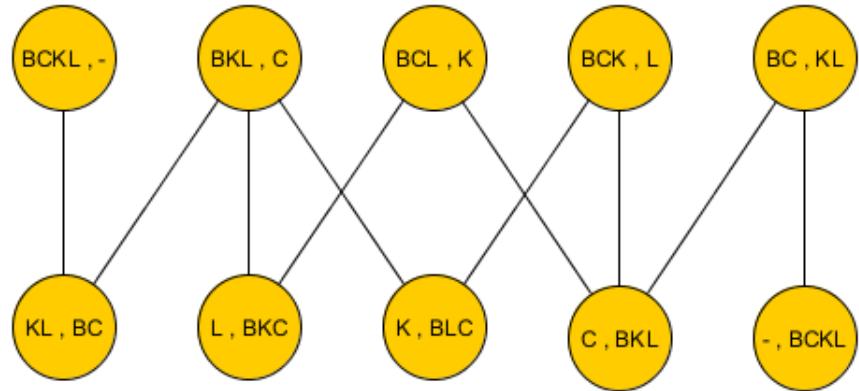
### Questions

1. Trouvez une solution à ce problème.
2. Modélez ce problème sous forme de graphe et expliquez comment obtenir une solution à partir de celui-ci.

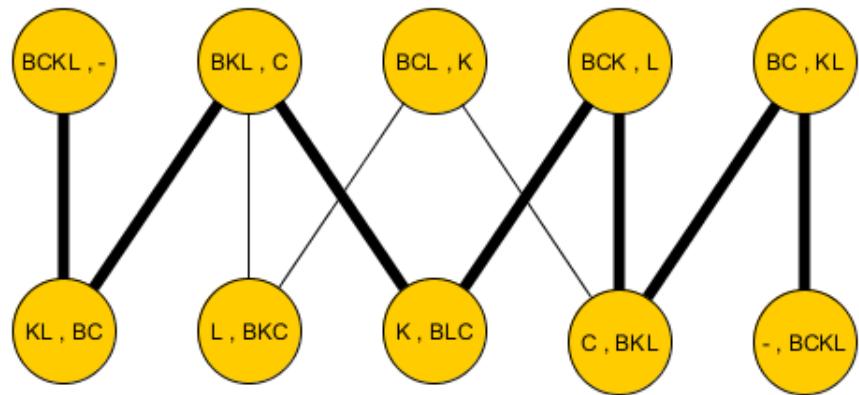
### Solutions

1. Le berger traverse avec la chèvre.
2. Il revient à vide.
3. Le berger traverse avec le loup.
4. Le berger revient avec la chèvre.
5. Le berger fait traverser le chou.
6. Il revient à vide.
7. Le berger traverse avec la chèvre.

Cette situation peut être modélisée à l'aide d'un graphe non orienté. Désignons par B le berger, par C la chèvre, par K le chou et par L le loup. Les sommets du graphe sont des couples précisant qui est sur la rive de départ et qui est sur la rive de destination. Ainsi, le couple (BCK,L) signifie que le berger est sur la rive de départ avec la chèvre et le chou (qui sont donc sous surveillance), alors que le loup est sur l'autre rive. Une arête relie deux sommets lorsque le passeur peut passer d'une situation à l'autre. En transportant la chèvre, le berger passe par exemple du sommet (BCK,L) au sommet (K,BCL). Dans ce graphe, nous n'ajouterons pas les situations interdites. Le graphe suivant représente l'ensemble des états possibles ainsi que les passages possibles d'un état à un autre :



Il suffit ensuite de trouver un chemin (n'importe lequel, mais le plus court étant le plus simple pour nous) entre la situation initiale ( $BCKL, -$ ) et notre objectif ( $-, BCKL$ ).



## 2.5 Graphe d'intervalle

### 2.5.1 Définition

Un intervalle est un ensemble compris entre deux valeurs. On peut parler d'intervalle de temps entre deux horaires, par exemple entre 17h00 et 18h30. Considérons ensuite un ensemble d'intervalles  $I_1, I_2, \dots, I_n \subset \mathbb{R}$ . Ces intervalles peuvent bien sûr s'intersecter :

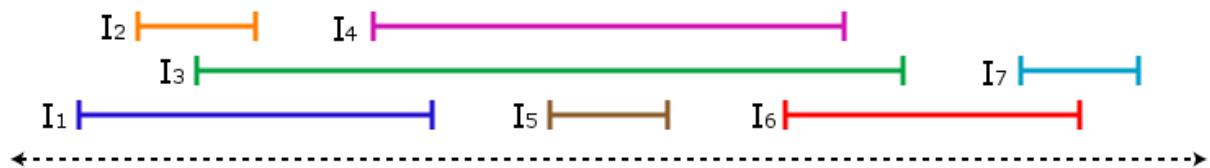


FIGURE 2.10 – Exemple de 7 intervalles

**Définition 25 (Graphe d'intervalle)** *Un graphe d'intervalle est le graphe d'intersection d'un ensemble d'intervalles et se construit de la manière suivante :*

1. Chaque sommet du graphe d'intervalle représente un intervalle de l'ensemble.

2. Une arête relie deux sommets lorsque les deux intervalles correspondants s'intersectent.

En reprenant l'exemple de la figure 2.10 on obtient donc le graphe d'intervalle suivant :

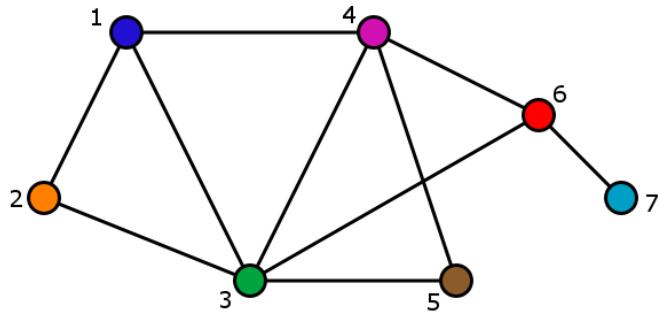


FIGURE 2.11 – Exemple de graphe d'intervalle

### 2.5.2 Déterminer si un graphe est un graphe d'intervalle

**Définition 26 (Corde)** *Une corde est une arête reliant deux sommets non adjacents d'un cycle.*

Par exemple, si on considère le graphe de la figure 2.8 et le cycle formé par les sommets  $(1, 3, 0, 2, 7, 4)$ , on peut remarquer que l'arête  $(1, 0)$  constitue une corde pour ce cycle.

**Définition 27 (Graphe cordal)** *Un graphe est cordal si chaque cycle de taille supérieure ou égale à 4 possède une corde. Un graphe cordal est aussi appelé graphe triangulé.*

En particulier, le graphe de la figure 2.11 est cordal. Cette définition revient à dire que tout cycle sans corde possède au plus trois sommets.

**Définition 28 (Graphe de comparabilité)** *Un graphe de comparabilité est un graphe non orienté qui relie les paires d'éléments qui sont comparables les uns aux autres dans un ordre partiel donné. Cela revient à dire que dans un tel graphe, si les arêtes  $(a, b)$  et  $(b, c)$  sont présentes, alors l'arête  $(a, c)$  est également présente.*

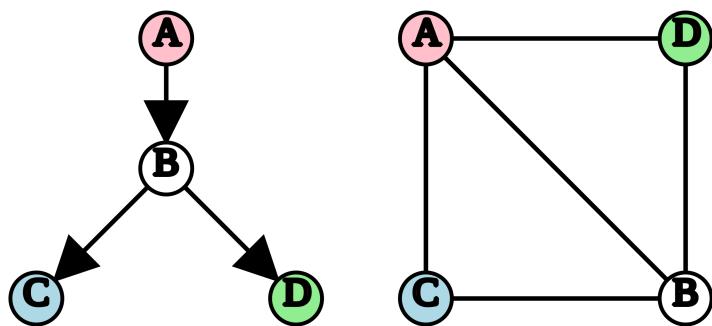


FIGURE 2.12 – Exemple de graphe représentant un ordre partiel et son graphe de comparabilité associé.

Par exemple, le graphe orienté de la figure 2.12 représente un ordre partiel :

- Le sommet A est situé avant le sommet B.
- Le sommet B est situé avant les sommets C et D.

- On ne peut pas comparer l'ordre relatif des sommets C et D.

Ainsi, le graphe non orienté de la figure 2.12 représente le graphe de comparabilité associé au graphe orienté :

- Le sommet A est comparable au sommet B, et donc aux sommets C et D : les arêtes (A,B), (A,C) et (A,D) existent.
- Le sommet B est comparable aux sommets A, C et D : les arêtes (A,B), (B,C) et (B,D) existent.
- Les sommets C et D sont comparables au sommet B et donc au sommet A, mais ne sont pas comparables entre eux : les arêtes (B,C), (B,D), (A,C), (A,D) existent, mais l'arête (C,D) n'existe pas.

**Algorithm 1:** Algorithme permettant de déterminer si un graphe est un graphe de comparabilité

```

Data:  $G = (V, E)$  un graphe non-orienté
Résultat: Vrai si le graphe est un graphe de comparabilité, faux sinon.
 $F = \emptyset;$ 
while  $F \neq E$  do
    Choisir une arête  $e \in E - F$ ;
    Donner une orientation à e;
    Propager cette orientation de sorte à assurer une orientation transitive de  $G$ ;
    if une arête est orientée dans les deux sens then
        return faux;
    end
    else
        Rajouter à  $F$  toutes les arêtes nouvellement orientées;
    end
end
return Vrai;
```

**Exemple de déroulement de l'algorithme** Nous allons vérifier si le graphe non-orienté de la figure 2.12 est un graphe de comparabilité.

- **[Initialisation.]**  $F = \emptyset$ .  $E = \{(A, B), (A, C), (A, D), (B, C), (B, D)\}$ .
- **[Etape 1.]** On choisit l'arête  $(B, C)$ . On oriente cette arête de  $C$  vers  $B$ .  $F = \{(C, B)\}$ .
- **[Etape 2.]** On choisit l'arête  $(B, D)$ . On oriente cette arête de  $D$  vers  $B$ .  $F = \{(C, B), (D, B)\}$ . Le choix de l'orientation de l'arête nous est imposé. En effet, choisir l'orientation allant de  $B$  vers  $D$  aurait imposé la transitivité de  $C$  vers  $D$ . Ors, l'arête  $(C, D)$  n'existe pas.
- **[Etape 3.]** On choisit l'arête  $(A, C)$ . On oriente cette arête de  $A$  vers  $C$ . Ce choix impose également de propager cette orientation à l'arête  $(A, B)$  afin d'assurer la transitivité ( $A \rightarrow C$  et  $C \rightarrow B$  donc  $A \rightarrow B$ )  $F = \{(C, B), (D, B), (A, C), (A, B)\}$ .
- **[Etape 4.]** On choisit la dernière arête disponible  $(A, D)$  ainsi qu'une orientation. De fait, les deux orientations possibles sont compatibles avec la transitivité. Choisissons de  $D$  vers  $A$ .  $F = \{(C, B), (D, B), (A, C), (A, B), (D, A)\}$ .
- **[Fin.]**  $F \Leftrightarrow E$ . Retourner vrai, le graphe est un graphe de comparabilité.

**Définition 29 (Graphe complémentaire)** Un graphe complémentaire  $G_c$  d'un graphe simple  $G$  est un graphe simple tel que deux sommets distincts de  $G_c$  sont adjacents si et seulement si ils ne sont pas adjacents dans  $G$ .

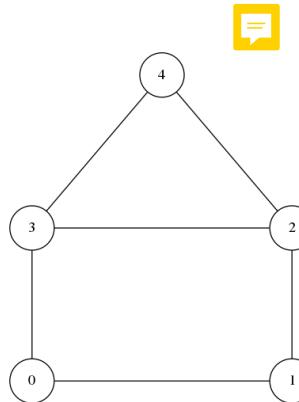


FIGURE 2.13 – Exemple de graphe

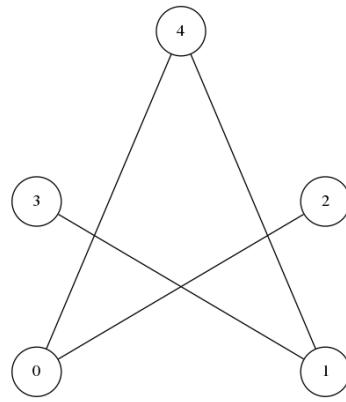


FIGURE 2.14 – Graphe complémentaire associé

**Théorème 1 (Graphe d'intervalle)** Un graphe  $G$  est un graphe d'intervalles si et seulement si son graphe complémentaire est un graphe de comparabilité et si  $G$  est un graphe cordal.

### 2.5.3 Exemple (ludique) d'application

**Qui a tué le Duc de Densmore :** Cet exercice est inspiré de la nouvelle de Claude Berge « Qui a tué le Duc de Densmore ». Une version est disponible à l'adresse suivante, page 77 : <http://iti.mff.cuni.cz/series/2013/598.pdf>. Dans cette nouvelle policière, le lecteur peut découvrir le meurtrier en manipulant des intervalles, et en tentant de construire le graphe d'intervalle associé.

Un jour, Sherlock Holmes reçoit la visite de son ami Watson que l'on avait chargé d'enquêter sur un assassinat mystérieux datant de plus de trois ans. À l'époque, le Duc de Densmore avait été tué par l'explosion d'une bombe, qui avait entièrement détruit le château de Densmore où il s'était retiré. Les journaux d'alors relataient que le testament, détruit lui aussi dans l'explosion, avait tout pour déplaire à l'un de ses proches. Or, avant de mourir, le Duc les avait tous invités à passer quelques jours dans sa retraite écossaise.

- **Holmes :** Je me souviens de cette affaire ; ce qui est étrange, c'est que la bombe avait été fabriquée spécialement pour être cachée dans l'armure de la chambre à coucher, ce qui suppose que l'assassin a nécessairement effectué plusieurs visites au château !
- **Watson :** Certes, et pour cette raison, j'ai interrogé chacun de ses proches : Anne, Betty, Charlotte, Edith, François, Grégoire et Hamed. Il ont tous juré qu'ils n'avaient été au château de Densmore qu'une seule fois dans leur vie.
- **Holmes :** Hum ! Leur avez-vous demandé à quelle période ils ont eu leur séjour respectif ?
- **Watson :** Hélas ! Aucun ne se rappelait les dates exactes, après plus de trois ans ! Néanmoins, je leur ai demandé qui ils avaient rencontré :
  - Anne a rencontré Betty, Charlotte, François et Grégoire.
  - Betty a rencontré Anne, Charlotte, Edith, François et Hamed.

- Charlotte a rencontré Anne, Betty et Edith.
- Edith a rencontré Betty, Charlotte et François.
- François a rencontré Anne, Betty, Edith et Hamed.
- Gregoire a rencontré Anne et Hamed.
- Hamed a rencontré Betty, François et Grégoire.

– **Watson** : Vous voyez, mon cher Holmes, les réponses sont concordantes !

C'est alors que Holmes prit un crayon et dessina un étrange petit dessin, avec des points marqué A, B, C, E, F, G, H et des lignes reliant certains de ces points. Puis, en moins de trente secondes, Holmes déclara :

– **Holmes** : Tiens, tiens ! Ce que vous venez de me dire détermine d'une façon unique l'assassin.

## Qui est l'assassin ?

## Chapitre 3

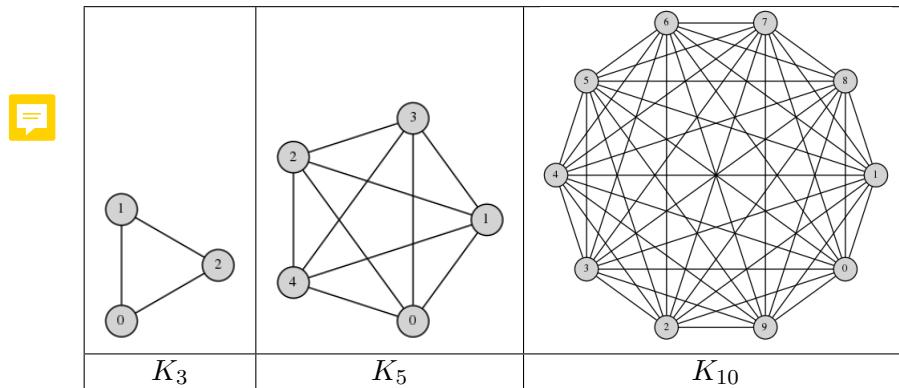
# Exemples de familles de graphes

Dans la littérature, il existe de très nombreux résultats et algorithmes qui nécessitent que le graphe considéré possède certaines propriétés. Par exemple, nous avons vu dans le chapitre précédent que le lecteur pouvait découvrir le meurtrier d'une nouvelle policière en manipulant des intervalles, et en tentant de construire le graphe d'intervalle associé. Ce type de graphe peut être défini comme l'ensemble de tous les graphes possibles qui respectent une certaine propriété. Bien qu'il existe une infinité de familles de graphes, certaines sont célèbres car elles reviennent régulièrement dans les démonstrations scientifiques ou permettent souvent de modéliser un problème particulier.

Dans ce chapitre, nous allons découvrir quelques familles de graphes parmi les plus célèbres, sans atteindre l'exhaustivité. Si vous souhaitez vous lancer des défis, n'hésitez pas à implémenter des fonctions permettant de les générer. Cela constitue un très bon exercice algorithmique.

### 3.1 Graphe complet à $n$ sommets

Un graphe complet à  $n$  sommets, noté  $K_n$  est un graphe pour lequel chaque sommet est voisin de tous les autres sommets du graphe. Il s'agit des graphes ayant le plus d'arêtes.



Ce type de graphe peut être utilisé pour modéliser toutes les interactions possibles entre divers protagonistes.

**Exemple, bataille de pokemons.** Les pokemons sont des créatures fictives, qui possèdent diverses caractéristiques et peuvent combattre en duel. Chaque pokemon possède un type (sol, air, vol, psy etc.) et peut lancer des attaques pour faire chuter les points de vie de son adversaire. Si les points de vie chutent à 0, le pokemon perd le combat. Chaque attaque possède également

un type. Le nombre de points de dégâts infligés par une attaque dépend de son type et du type du pokémon attaqué, sur le même principe que le jeu « pierre, feuille, ciseaux ». Les données concernant les pokemons sont disponibles en open data sur le site kaggle : <https://www.kaggle.com/rounakbanik/pokemon>. Le graphe<sup>1</sup> suivant représente le bonus (ou le malus) dont bénéficie une attaque en fonction du type du pokémon attaqué (noir  $\times 0$ , jaune  $\times \frac{1}{2}$ , bleu  $\times 1$  et rouge  $\times 2$ ) :

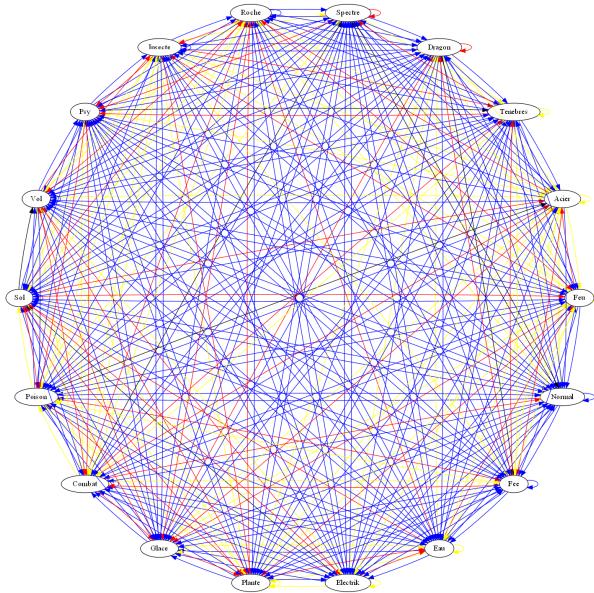
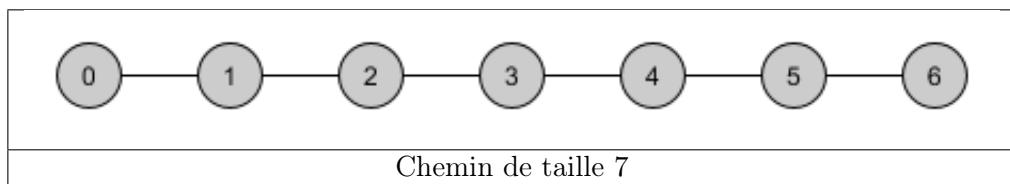


FIGURE 3.1 – Bonus et malus d’attaque en fonction du type des pokemons

## 3.2 Chemin

Un chemin à  $n$  sommets est un graphe connexe sans cycle où tous les sommets sont de degré 2 sauf deux qui sont de degré 1. Cette famille de graphe possède exactement  $n-1$  arêtes. la suppression d’un sommet ou d’une arête permet d’obtenir deux chemins de taille inférieure. Cette propriété permet, par exemple, d’analyser facilement la taille de la solution retournée par un algorithme non déterministe grâce à l’obtention d’une expression récursive. C’est pour cette raison qu’il s’agit généralement d’une des premières familles de graphes à être analysée lorsqu’on souhaite comprendre le comportement d’un algorithme ou analyser la structure d’un problème. Si cela vous intéresse, vous pouvez consulter l’article suivant [3].

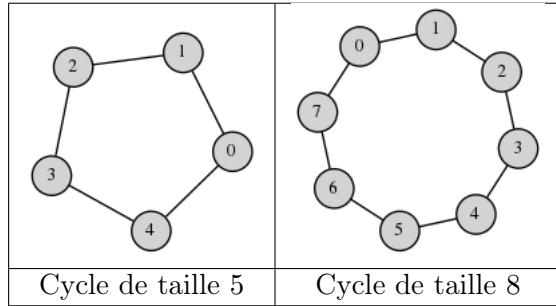


## 3.3 Cycle

Un cycle de taille  $n$  est un **graphe connexe** et dont tous les sommets sont de degré 2. On peut le voir comme un chemin de taille  $n$  auquel on ajoute l’une arête entre le premier et le

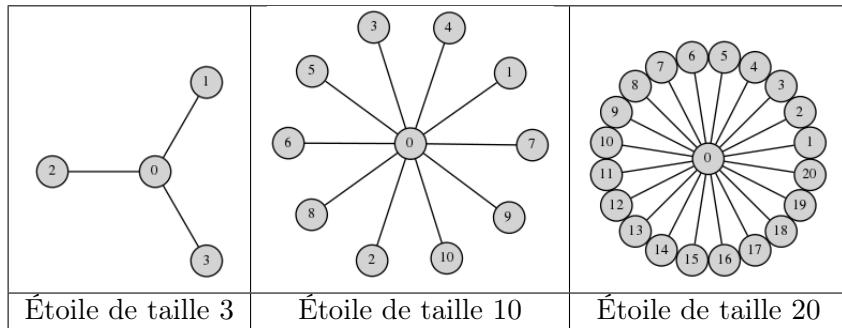
1. Merci à Anani Agondja Thibault et Lourme Matthieu, étudiants de la Miage de Nanterre pour la génération de ce graphe.

dernier sommet. Cette arête supplémentaire peut avoir un impact important sur la structure des solutions.



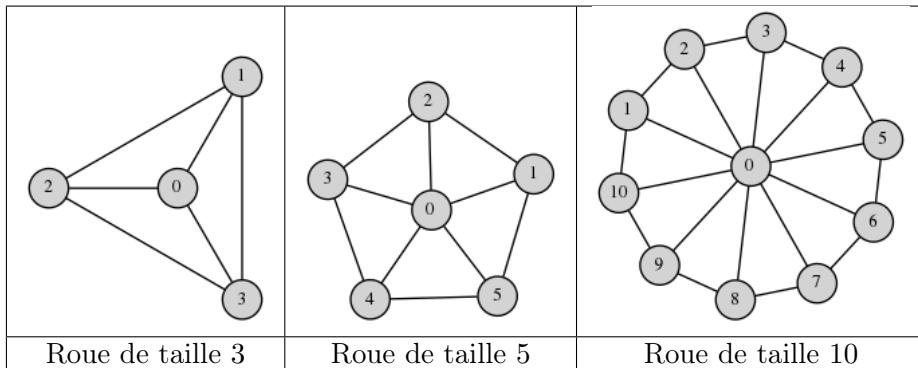
### 3.4 Étoile

Une étoile de taille  $n$  est un graphe à  $n + 1$  sommets. L'un des sommets est relié à tous les autres et il n'y a aucune autre arête. Ce graphe permet d'obtenir de manière triviale un sommet de degré maximum et autant de sommets de degré minimum, soit 1, tout en conservant la connexité. Cette famille de graphe permet souvent d'exhiber un contre exemple lors d'une tentative de preuve ou un cas d'exécution pathologique pour un algorithme.



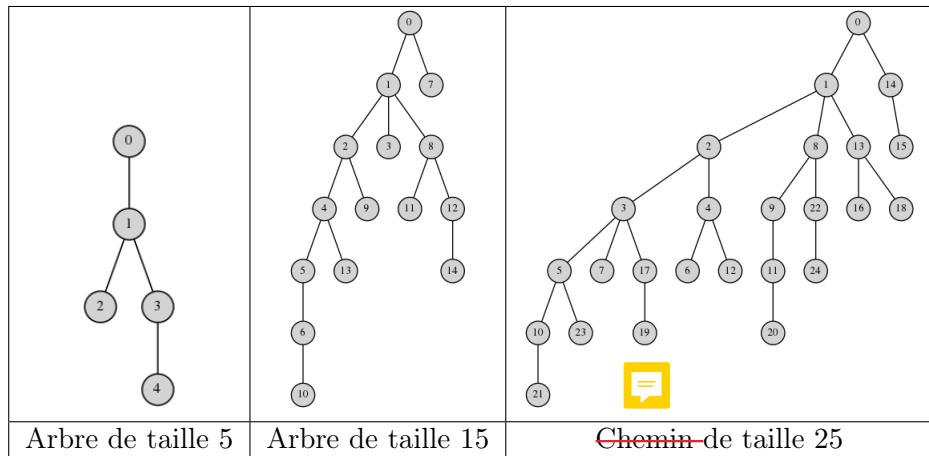
### 3.5 Roue

Une roue de taille  $n$  est un graphe à  $n + 1$  sommet.  $n$  de ces sommets forment un cycle de taille  $n$ , et le dernier sommet est relié à tous les autres. On peut voir cette famille comme la fusion d'une étoile et d'un cycle. Comme précédemment, cette famille permet d'illustrer un comportement algorithmique.



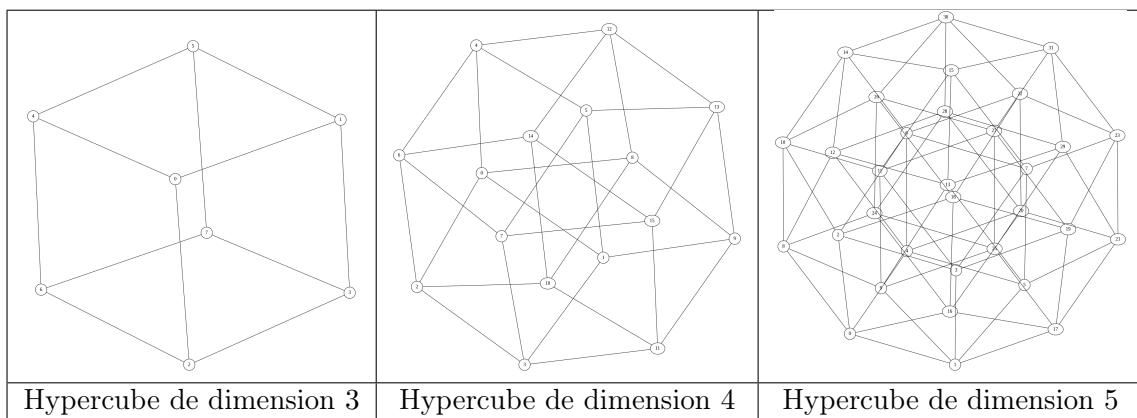
## 3.6 Arbre

Un arbre de taille  $n$  est un graphe connexe contenant  $n$  sommets et sans cycle. Il s'agit d'une famille de graphes très importante. En effet, les arbres sont souvent utilisés dans les réseaux pour créer une structure de communication entre les différents membres, sans pour autant permettre de boucle et donc de réception multiple de messages. A noter également, la suppression d'un sommet ou d'une arête entraîne automatiquement la perte de connexité. On obtient donc un ensemble d'arbres de diverses tailles et formes. Un tel ensemble est appelé « une forêt ».



## 3.7 hypercubes

Un hypercube de dimension  $n$  peut être vu comme la généralisation multi-dimensionnelle d'un carré ( $n = 2$ ) et d'un cube ( $n = 3$ ). Pour construire un cube de dimension  $n$ , il faut construire deux hypercubes de dimension  $n - 1$ , puis relier chaque sommet du premier à son équivalent dans le second. Une autre manière de construire est hypercube est de remarquer qu'il possède exactement  $2^n$  sommets. Si chaque sommet est numéroté de 0 à  $2^n - 1$ , il suffit de relier chaque sommet à tous les sommets dont la décomposition en binaire ne diffère que d'un bit (distance de Hamming de 1).



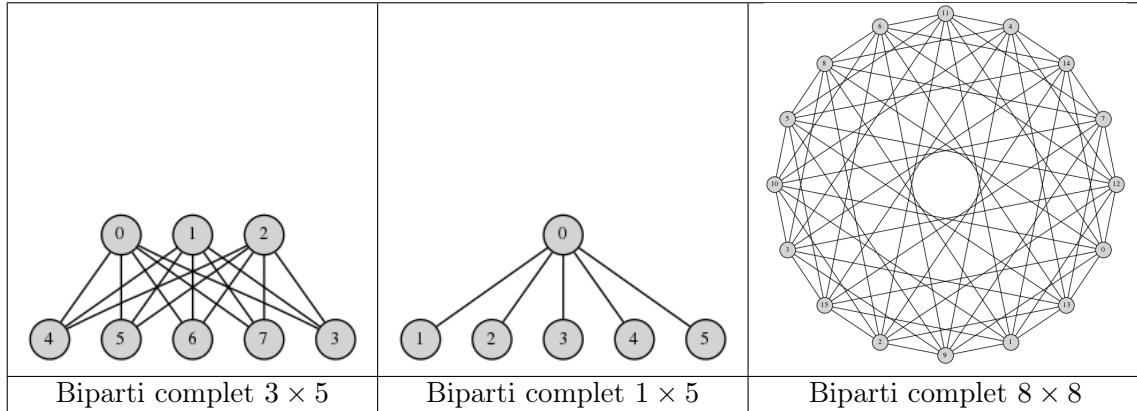
## 3.8 Graphe biparti

Un graphe biparti est un graphe sans cycle de longueur impaire. Concrètement, on peut partitionner ses sommets en deux ensembles tels qu'il n'existe pas d'arête entre deux sommets d'un même ensemble. Ainsi, les seules arêtes autorisées sont celle qui possèdent une extrémité

dans chaque ensemble. Ces graphes sont très souvent utilisés pour modéliser les relations entre deux populations. Par exemple, un site de rencontre peut souhaiter représenter les affinités entre un groupe de femmes et un groupe d'hommes pour ensuite déterminer le nombre de couples maximum pouvant être constitués. Voici à titre d'exemple deux « sous familles » de graphes bipartis :

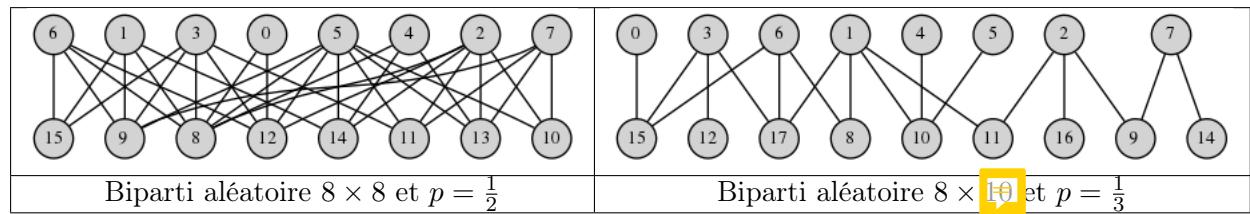
### 3.8.1 Graphe biparti complets

Un graphe biparti complet est un graphe biparti pour lequel chaque sommet d'un ensemble est relié à tous les sommets de l'autre ensemble.



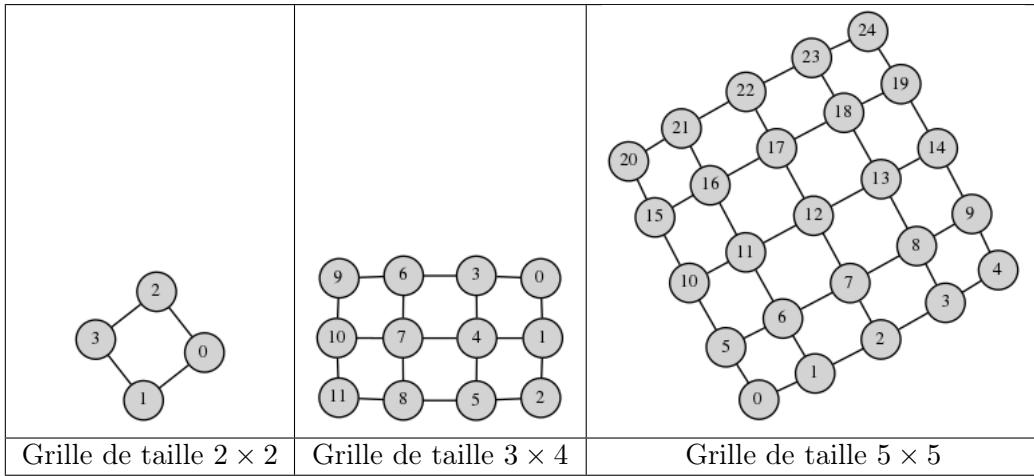
### 3.8.2 Graphe biparti aléatoire

Un graphe biparti aléatoire est un graphe biparti pour lequel les arêtes sont tirées aléatoirement, avec une probabilité fournie en entrée.



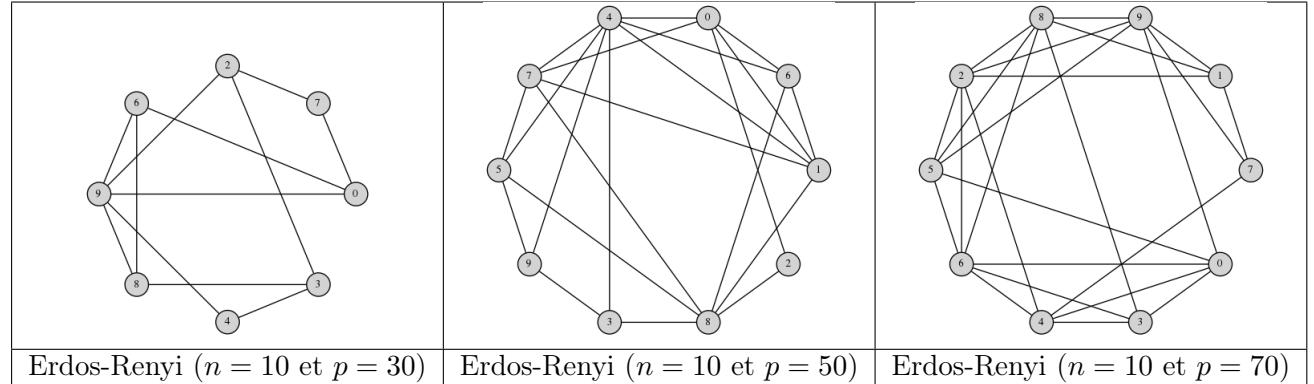
## 3.9 Grille

Une grille de taille  $n \times m$  est un graphe contenant  $n \times m$  sommets disposés en  $m$  lignes de  $n$  sommets. Chaque sommet est relié aux sommets se situant à sa gauche, sa droite, au dessus et en dessous de lui (lorsqu'ils existent). Cette famille possède un certain nombre de propriétés (par exemple il s'agit de graphes bipartis, et à partir de l'identifiant d'un sommet on peut déduire la liste de ses voisins).



### 3.10 Graphe aléatoire de Erdös-Renyi

Un graphe aléatoire de Erdos-Renyi, noté  $G_{n,p}$ , est un graphe aléatoire de taille  $n$  dans lequel chaque arête potentielle existe avec probabilité  $p$ . Notez bien que l'existence d'une arête est indépendante de celle des autres. Cette famille est très importante, en particulier pour l'analyse théorique. La maîtrise de la probabilité d'existence d'une arête permet de contrôler l'espérance du nombre d'arêtes présentes dans le graphe soit  $p \cdot \frac{n \cdot (n-1)}{2}$  ou l'espérance du nombre de triangles soit  $p^3 \cdot \frac{n \cdot (n-1) \cdot (n-2)}{6}$ .



On peut distinguer plusieurs valeurs de probabilité d'arête :

1.  $p < \frac{1}{n}$ , alors avec forte probabilité, le graphe ne possédera pas de composante connexe de taille plus grande que  $\log(n)$ .
2.  $p = \frac{1}{n}$ , alors avec forte probabilité, le graphe possédera pas de composante connexe de taille  $n^{\frac{2}{3}}$ .
3.  $p \geq \frac{1}{n}$ , alors avec forte probabilité, le graphe possédera une composante connexe géante et aucune autre composante ne sera de taille plus grande que  $\log(n)$ .
4.  $p = \frac{c}{n}$ , avec  $c$  une constante. Permet d'obtenir un graphe dont le degré moyen est de  $c$ . Ces graphes sont considérés comme peu denses (relativement au nombre d'arêtes possible).
5.  $p = \frac{1}{2}$ , reviens à tirer au sort un graphe de taille  $n$  de manière équiprobable parmi l'ensemble des graphes de taille  $n$ .

### 3.11 Graphe petit monde

Le concept de petit monde est l'hypothèse selon laquelle chaque individu peut être relié (par un lien de connaissance, d'amitié) à n'importe quel autre individu du monde par une courte chaîne de relations sociales. Par exemple, un individu connaît le maire de son village. Le maire connaît un député, qui connaît le président de la république, qui connaît n'importe quel autre chef d'état. Il existe un mécanisme similaire pour « redescendre » de ce chef d'état vers n'importe quel individu de son propre pays. Ainsi chaque individu du monde doit pouvoir être relié à n'importe quel autre individu par une chaîne de taille 8. On dit d'un graphe qu'il est « petit monde », si :

1. La distance moyenne entre deux sommets est au plus logarithmique en le nombre de sommets du graphe.
2. La densité locale est assez forte.
3. La densité globale est plutôt faible.

En particulier, les réseaux sociaux respectent généralement cette propriété. Le graphe suivant reprends le graphe des organisations présenté dans l'article [1] du chapitre 1 de ce cours.

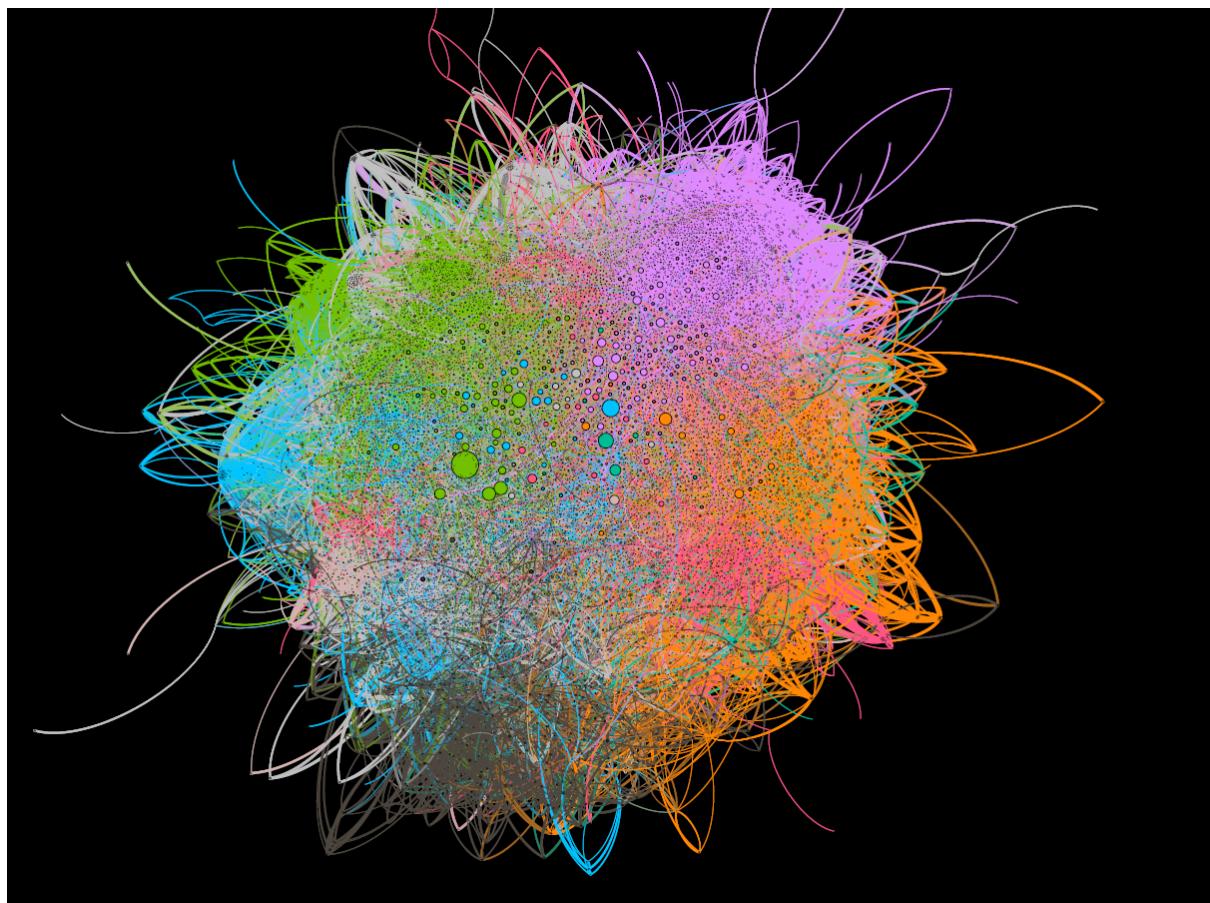


FIGURE 3.2 – Graphe du réseau de collaboration issu du Programme Cadre européen H2020

Ce qui est intéressant dans ce type de graphe, c'est qu'il est possible de détecter différentes communautés. C'est à dire un ensemble d'individus plus fortement liés entre eux par rapport au reste du graphe (par exemple un groupe d'amis sur facebook). Chaque communauté peut être représentée par une couleur différente, ce qui peut donner un résultat assez esthétique. Le

scientifique interviendra par la suite pour tenter de caractériser ces différentes communautés.

Le graphe suivant a été obtenu grâce aux données issues du grand débat national réalisé en France en 2019. A partir données, anonymisées et obtenues par un questionnaire en ligne ont permis de calculer une distance « politique » entre deux individus. Si cette distance était inférieure à une certaine valeur, une arête était ajoutée. A partir de ce graphe, un algorithme de détection de communautés a été lancé, pour finalement obtenir le résultat suivant :

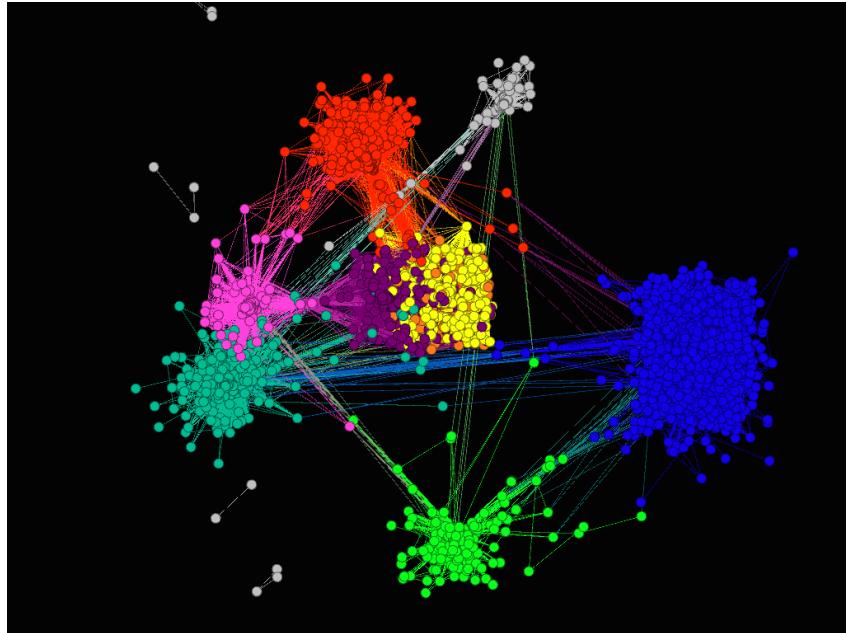


FIGURE 3.3 – Communautés issues du grand débat

Comme vous pouvez le voir, outre un aspect esthétique certain, la théorie des graphes permet à la fois d'obtenir des résultats théoriques, mathématiques, mais aussi pratiques, ancrés dans le réel. C'est un outil de modélisation extrêmement puissant et pouvant avoir un impact dans tous les domaines de notre société.

## Chapitre 4

# Résolution exacte et approchée, ainsi que quelques problèmes fameux

Dans cette partie, vous allez découvrir quelques problèmes célèbres issus de la théorie des graphes. Tout d'abord, nous allons étudier de manière approfondie le problème du vertex cover afin de montrer comment le résoudre, de manière exacte mais aussi approchée. Pourquoi ce problème ? Tout simplement parce que de nombreux autres problèmes sont très fortement liés à celui-ci [4] et ne sont, finalement, que des variations. Ensuite, de manière succincte, seront présentés d'autres problèmes tout aussi importants, par leur utilité, leur originalité ou la fréquence dans laquelle on peut les retrouver dans l'industrie. Si vous trouvez l'un de ces problèmes passionnant (à juste titre), vous êtes invités à consulter la bibliographie associée.

### 4.1 Le problème du vertex cover

En 1972, un an après que Cook ait formalisé la notion de NP-complétude et prouvé que le problème SAT est NP-complet, Richard Karp a publié un article fondateur [5] en théorie de la complexité, dans lequel il prouve la NP-complétude de 21 problèmes dont celui du Minimum Vertex Cover (MVC) :

**Définition 30 (Minimum Vertex Cover)** Étant donné un graphe  $G = (V, E)$ , avec  $V$  l'ensemble de ses sommets et  $E$  l'ensemble de ses arêtes, le problème du vertex cover consiste à trouver le plus petit ensemble possible  $C \subseteq V$  tel que pour toute arête  $(i, j) \in E$  on ait  $i \in C$  ou  $j \in C$  (ou bien les deux). On dit que  $C$  est une couverture minimale de  $G$  (un vertex cover de  $G$ ).

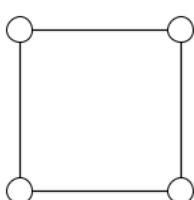


FIGURE 4.1 – Un graphe  $G$

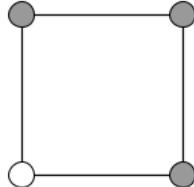


FIGURE 4.2 – Une couverture des sommets de  $G$

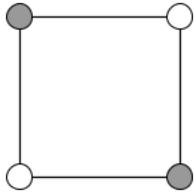


FIGURE 4.3 – Une couverture optimale des sommets de  $G$

**Par exemple,** la figure 4.1 représente un graphe avec 4 sommets et 4 arêtes. Les sommets en gris des figures 4.2 et 4.3 sont les sommets faisant partie de la solution. Clairement, les sommets de la figure 4.2 forment bien une couverture (un vertex cover) car toutes les arêtes possèdent au moins une extrémité dans l'ensemble des sommets gris, mais elle n'est pas de taille minimum et n'est pas non plus minimale<sup>1</sup>. La figure 4.3 propose un vertex cover minimum (et cette solution est donc minimale).

**Temps de résolution.** Le fait que ce problème soit NP-complet signifie qu'il est très improbable qu'on puisse trouver un algorithme le résolvant de manière exacte en un temps polynomial. De plus, ce problème reste NP-complet même pour certaines classes de graphes particulières.

**Un problème fondamental.** La recherche d'un vertex cover intervient souvent dans la résolution d'autres problèmes (le plus souvent lors d'une étape intermédiaire) comme par exemple dans l'alignement de séquences multiples, la résolution de conflits biologiques ou bien encore dans la surveillance des réseaux. Ces applications nécessitent des méthodes de résolution différentes (résolution exacte, heuristique ou d'approximation avec garantie de performances) le choix s'effectuant en fonction du temps dont on dispose et de la qualité de solution requise. Ainsi, de nombreuses heuristiques ont été proposées et bien que ces algorithmes ne garantissent pas de trouver la meilleure solution, leur temps d'exécution est acceptable.

### 4.1.1 Résolution exacte

Dans cette partie, nous allons voir deux méthodes de résolution exacte. J'ai adapté ces deux méthodes génériques afin de les utiliser pour le problème du vertex cover à titre d'exemple. Face à un autre problème, il sera nécessaire de les adapter spécifiquement.

#### 4.1.1.1 Algorithmes de coupe et de branchement (Branch and Bound)

Dans la plupart des problèmes rencontrés dans l'industrie, le nombre de solutions est fini. Il est donc possible, en principe, d'énumérer toutes ces solutions, et ensuite de choisir celle qui nous arrange, celle qui optimise un critère en particulier. Malheureusement, le nombre de solutions à énumérer peut être gigantesque, ce qui peut rendre cette solution inenvisageable. Les méthodes de type branch and bound (procédures par évaluation et séparation progressive) consistent à énumérer ces solutions d'une manière intelligente, en exploitant la structure du problème et son contexte. En particulier, cette technique va éliminer des solutions partielles qui ne mènent pas à une solution optimale. Il devient donc possible d'effectuer une résolution exacte, même si dans certains cas le temps nécessaire reste prohibitif.

**Exemple de remarque permettant d'exploiter la nature du problème.** Si on cherche à savoir si il existe une solution de taille au plus  $k$  pour le problème du vertex cover, et qu'il existe un sommet  $s$  de degré  $k + 1$ , il est certain que ce sommet fera partie de la solution (si elle existe). En effet, ne pas sélectionner le sommet  $s$  implique, nécessairement, de sélectionner son voisinage afin de couvrir toutes les arêtes. Cette remarque nous permet d'éliminer directement toutes les solutions potentielles qui ne contiennent pas le sommet  $s$ .

---

1. On distingue le terme minimum qui représente ici la valeur de la plus petite solution du terme minimal que l'on utilisera au sens de l'inclusion. Dans le cas présent, cette solution n'est pas minimale car si on retire le sommet gris en haut et à droite du graphe, notre solution est toujours une couverture. Évidemment, une solution de taille minimum est forcément minimale.

**Exemple d’algorithme pour le problème du vertex cover** L’algorithme présenté ici est un algorithme « jouet ». De nombreuses améliorations sont possibles. Vous pouvez, en particulier, consulter l’ouvrage suivant : [7].

1. Nous allons créer un arbre binaire.
2. Chaque noeud de l’arbre va représenter une partition des sommets du graphe : ceux qui font partie de la solution, ceux qui ne font pas partie de la solution et ceux pour lesquels une décision reste à prendre.
3. Ainsi, la racine de notre arbre contient un seul ensemble contenant l’ensemble des sommets du graphe.
4. A partir d’un noeud de notre arbre, on va choisir un sommet  $s$  du graphe pour lequel une décision reste à prendre.
5. Pour ce sommet, nous allons prendre les deux décisions possibles : l’inclure dans la solution ou non. Cela se matérialise par la création de deux nouveaux noeuds de notre arbre, l’un reprenant les décisions précédentes et incluant  $s$  dans la solution courante, l’autre reprenant les décisions précédentes et excluant  $s$  de la solution.
6. A partir de là, il suffit de recommencer l’opération (récursevement) sur les deux nouveaux noeuds créés jusqu’à ce qu’une décision soit prise pour tous les sommets.

Notez bien que ne pas inclure un sommet dans une solution implique nécessairement d’ajouter tous les sommets de son voisinage dans la solution. Dans le cas contraire, certaines arêtes ne seraient pas couvertes.

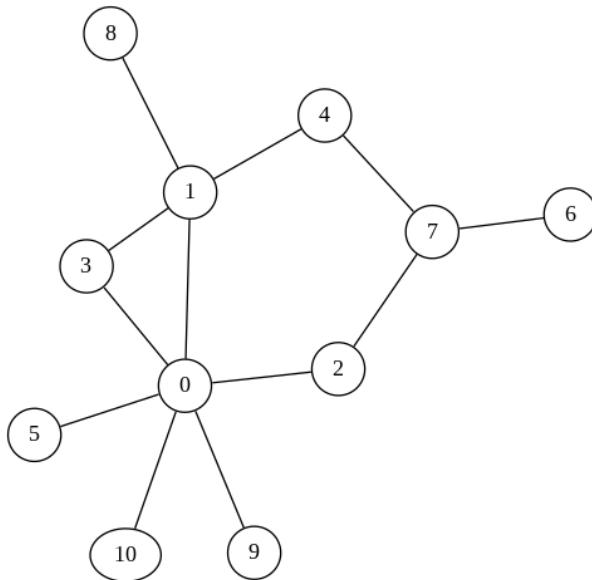
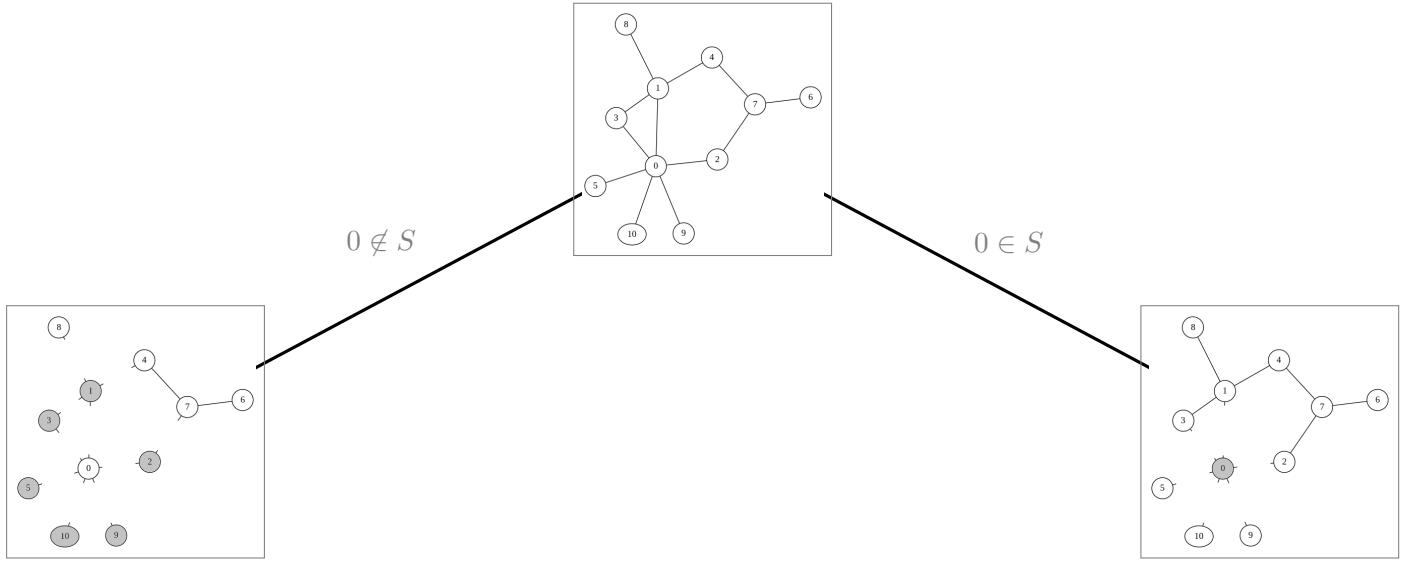


FIGURE 4.4 – Exemple de graphe pour lequel nous allons exécuter notre algorithme de branchement.

**Premier niveau de l'arbre.** Nous allons prendre une décision concernant un sommet. Par exemple le sommet 0 :

$0 \notin S$  Les arêtes ayant le sommet 0 pour extrémité doivent être couvertes. Comme le sommet 0 ne fait pas partie de la solution, cela implique de sélectionner les sommets 1, 2, 3, 5, 9 et 10 pour couvrir ces arêtes.

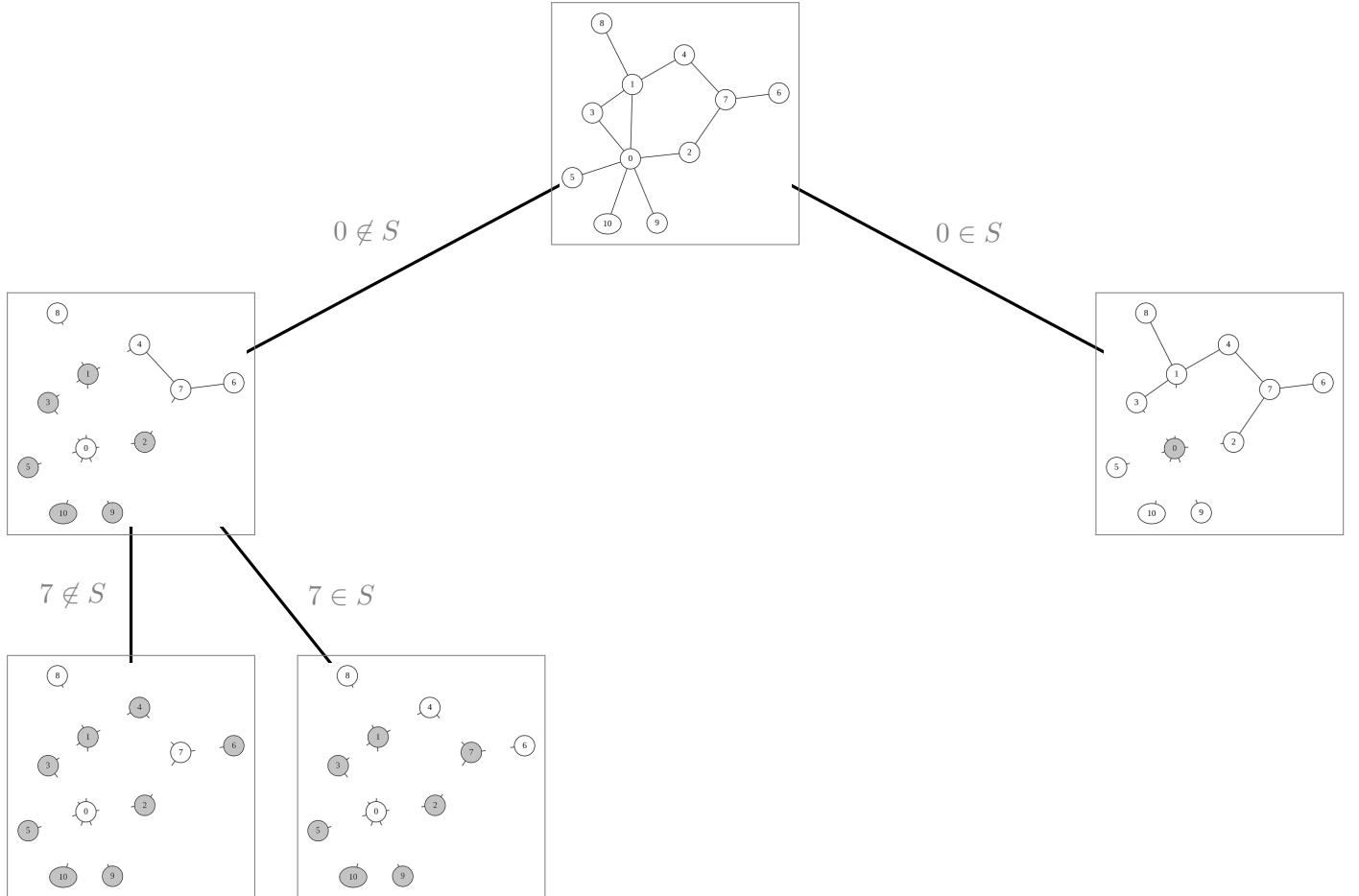
$0 \in S$  Les arêtes  $(0, 1), (0, 2), (0, 3), (0, 5), (0, 9)$  et  $(0, 10)$  sont couvertes. De plus, sélectionner les sommets 5, 9 ou 10 ne permettrait pas de couvrir de nouvelle arête. Il ne sera donc plus nécessaire de les considérer.



**Deuxième niveau de l'arbre.** Nous allons considérer tout d'abord le cas pour lequel le sommet 0 n'a pas été inclus dans la solution  $S$  ( $0 \notin S$ ). Nous allons prendre une décision concernant un sommet incident à une arête. Par exemple le sommet 7.

$7 \notin S$  Les arêtes ayant 7 pour extrémité doivent être couvertes. Il est donc nécessaire de sélectionner les sommets 4 et 6 pour couvrir ces arêtes. Toutes les arêtes sont couvertes. Fin de l'algorithme dans cette branche de l'arbre. La solution trouvée est la suivante :  $S = \{1, 2, 3, 4, 5, 6, 9, 10\}$ .

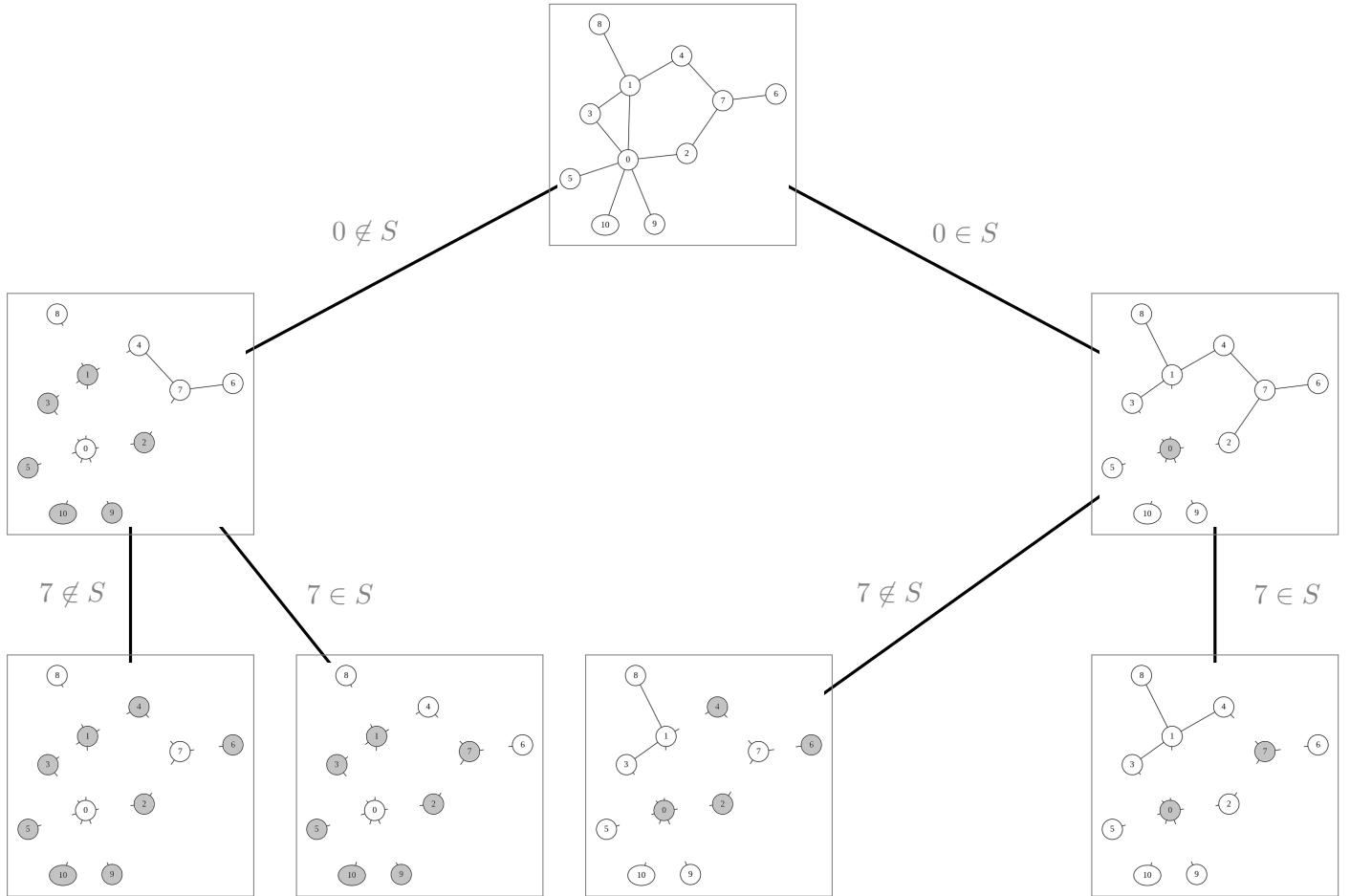
$7 \in S$  Les arêtes  $(7, 4), (7, 6)$  sont couvertes. Toutes les arêtes sont couvertes. Fin de l'algorithme dans cette branche de l'arbre. La solution trouvée est la suivante :  $S = \{1, 2, 3, 5, 7, 9, 10\}$ .



**Deuxième niveau de l'arbre (suite).** Considérons maintenant le cas pour lequel le sommet 0 a été inclus dans la solution  $S$  ( $0 \in S$ ). Nous allons prendre une décision concernant un sommet incident à une arête. Par exemple (et encore une fois) le sommet 7.

$7 \notin S$  Les arêtes ayant 7 pour extrémité doivent être couvertes. Il est donc nécessaire de sélectionner les sommets 2, 4 et 6 pour couvrir ces arêtes.

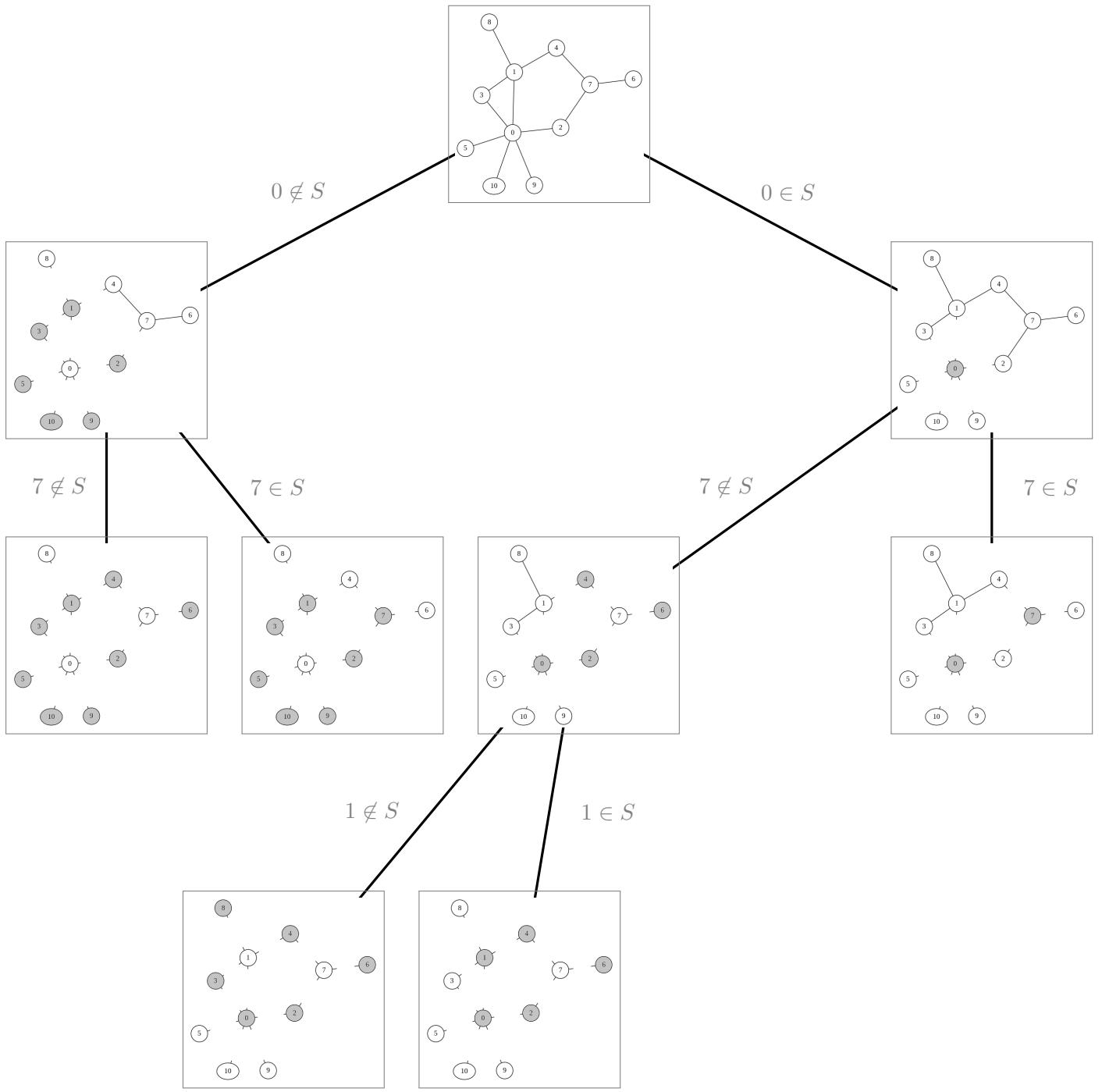
$7 \in S$  Les arêtes  $(7, 2), (7, 4)$  et  $(7, 6)$  sont couvertes.



**Troisième niveau de l'arbre.** On peut remarquer que les choix de ne pas placer le sommet 0 dans la solution puis d'ajouter ou non le sommet 7 mènent à deux solutions  $\{1, 2, 3, 5, 7, 9, 10\}$  et  $\{1, 2, 3, 4, 5, 6, 9, 10\}$ . Comme toutes les arêtes sont couvertes, l'algorithme s'arrête (dans ces branches). Au contraire, le choix de placer le sommet 0 dans la solution puis d'ajouter ou non le sommet 7 ne permet pas de couvrir toutes les arêtes. L'algorithme doit donc continuer à prendre des décisions. Par exemple en prenant des décisions concernant le sommet 1. Tout d'abord, considérons le cas où  $0 \in S$  et  $7 \notin S$  :

$1 \notin S$  Les arêtes ayant 1 pour extrémité doivent être couvertes. Il est donc nécessaire de sélectionner les sommets 3 et 8 pour couvrir ces arêtes (le sommet 4 étant déjà sélectionné lors de l'étape précédente).

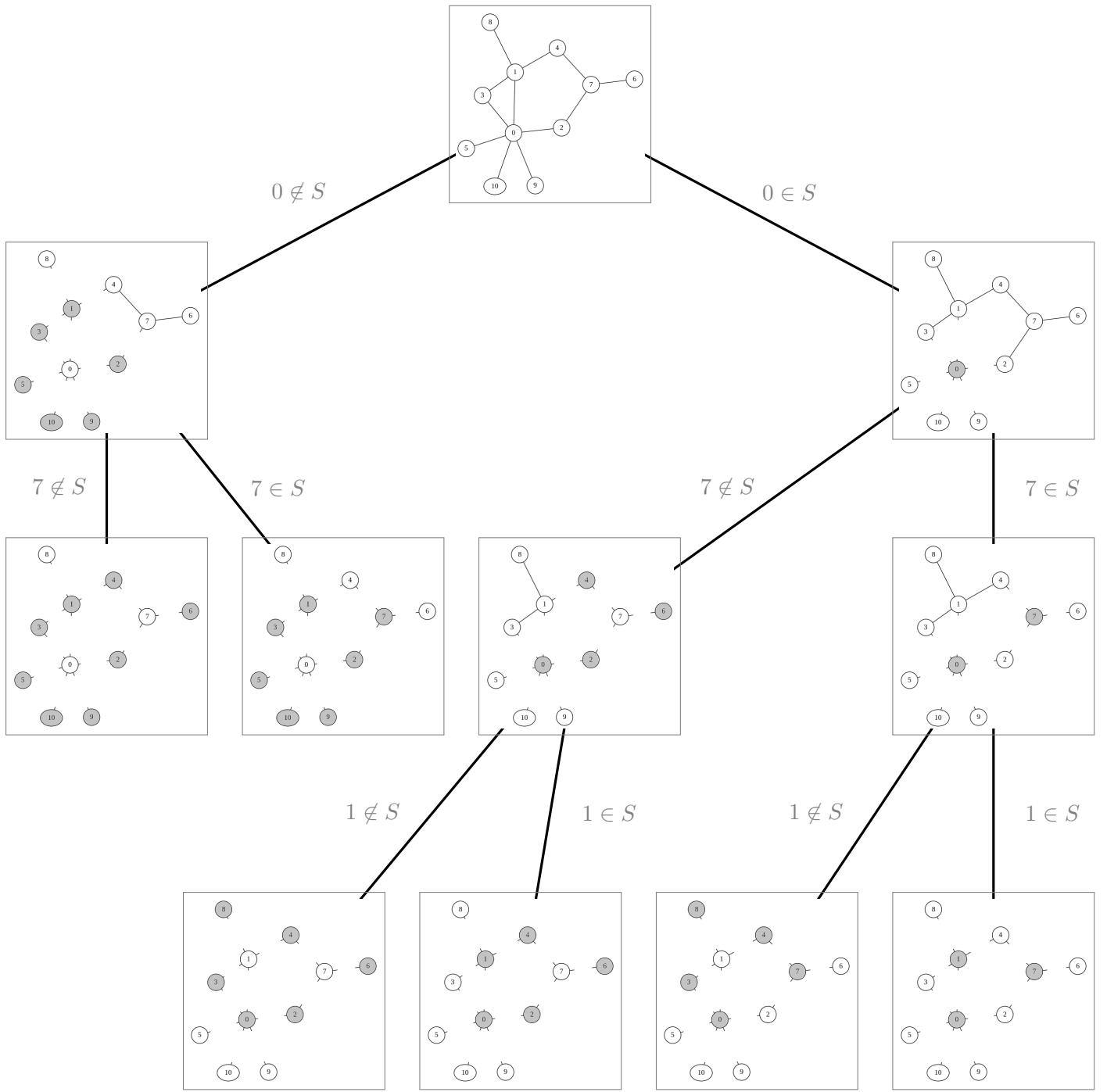
$1 \in S$  Les arêtes  $(1, 3), (1, 4)$  et  $(1, 8)$  sont couvertes.



**Troisième niveau de l'arbre (suite).** Considérons maintenant le cas où  $0 \in S$  et  $7 \in S$  :

$1 \notin S$  Les arêtes ayant 1 pour extrémité doivent être couvertes. Il est donc nécessaire de sélectionner les sommets 3, 4 et 8 pour couvrir ces arêtes.

$1 \in S$  Les arêtes  $(1,3)$ ,  $(1,4)$  et  $(1,8)$  sont couvertes.



**Ordre des décisions.** Il est clair que l'ordre dans lequel l'algorithme prend des décisions va affecter la vitesse à laquelle l'algorithme va atteindre des solutions. Il est d'usage, pour le problème du vertex cover, de considérer en premier les sommets de plus fort degré. Si le sommet est sélectionné, ajouté à la solution, cela implique de couvrir un maximum d'arêtes. Si au contraire on ne sélectionne pas ce sommet, alors cela implique d'ajouter tout son voisinage à la solution courante, ce qui va couvrir un nombre d'arêtes encore plus grand.

**Parcours de l'arbre de branchement.** Dans cet exemple, nous avons parcouru l'arbre de décision au fur et à mesure de sa construction. Bien entendu, une implémentation réelle ne

conserverait pas en mémoire chacune des branches, étant donné leur nombre. De plus, il est possible d'exploiter les solutions déjà parcourues. Par exemple, si la branche  $0 \in S$ ,  $7 \in S$ ,  $1 \in S$  est explorée en premier cela signifie qu'une solution de taille 3 aura été rencontrée. Ainsi, lorsque la branche  $0 \notin S$  sera explorée, il sera raisonnable de ne pas explorer les sous branches (prenant la décision pour le sommet 7) car la taille de la solution courante sera de 6, ce qui est supérieur à 3. Il ne sera donc pas possible de trouver une solution optimale dans le sous arbre, ce qui rend son exploration non nécessaire.

#### 4.1.1.2 Programme linéaire

Nous allons voir une autre méthode de résolution exacte, faisant intervenir un solveur de programmes linéaires. Nous verrons la programmation linéaire plus tard dans ce cours. Vous pourrez revenir sur cette partie par la suite. Comprenez juste que les solveurs de Programmes Linéaires en Nombres Entiers sont très puissant, et constituent un axe de recherche très important. Ainsi, une méthode de résolution exacte consiste à décrire l'instance que l'on souhaite résoudre sous forme de programme linéaire. Cela évite en particulier de concevoir un programme ad hoc.

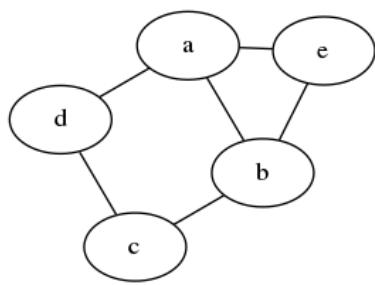
Voici comment faire dans notre cas :

1. On souhaite minimiser la taille de la solution (qui correspond au nombre de sommets).
2. On associe à chaque sommet  $u \in V$  une variable  $X_u$ .
3. Chacune de ces variables ne pourra prendre que deux valeurs, 0 ou 1.
4. Le solveur va attribuer la valeur 0 à une variable  $X_u$  si le sommet  $u$  n'est pas retenu dans la solution, et 1 si le sommet  $u$  est intégré dans la solution.
5. Pour chaque arête du graphe, au moins l'un des deux sommets doit faire partie de la solution. On ajoute donc une contrainte par arête imposant qu'au moins un des deux sommets soit sélectionné.

Ce qui donne un problème mathématique de la forme suivante :

$$\begin{aligned} & \text{Minimiser} && \sum_{u \in V} X_u \\ & \text{Tel que :} && X_u + X_v \geq 1 \quad \forall (u, v) \in E \\ & && X_v \in \{0, 1\} \quad \forall v \in V \end{aligned}$$

**Exemple de graphe et de programme linéaire associé :**



$$\begin{aligned} & \text{Minimiser} && X_a + X_b + X_c + X_d + X_e \\ & \text{Tel que :} && X_a + X_b \geq 1 \\ & && X_a + X_d \geq 1 \\ & && X_a + X_e \geq 1 \\ & && X_b + X_c \geq 1 \\ & && X_b + X_e \geq 1 \\ & && X_c + X_d \geq 1 \\ & && X_v \in \{0, 1\} \quad \forall v \in V \end{aligned}$$

FIGURE 4.5 – Exemple de graphe

FIGURE 4.6 – Programme mathématique associé

Il ne reste plus qu'à utiliser votre solveur préféré.

### 4.1.2 Résolution approchée

Les algorithmes d'approximation sont, le plus souvent, associés aux problèmes NP-difficiles. En effet, puisqu'il est improbable que l'on puisse trouver un algorithme polynomial résolvant de façon exacte un problème NP-difficile, l'utilisation d'algorithmes polynomiaux retournant des solutions non optimales s'est généralisée. Cependant, il est souhaitable que la qualité des solutions retournées ne soit pas trop dégradée et on demande donc à ces algorithmes de donner une garantie sur leurs performances.

Par exemple, pour un problème de minimisation, un algorithme  $\alpha$ -approché est un algorithme qui s'exécute (généralement) en un temps qui est polynomial en la taille de l'instance à traiter et qui retourne une solution dont on garantit que la valeur ne dépasse pas  $\alpha$  fois la valeur de la solution optimale. Autrement dit, si on note  $Opt$  la valeur de la solution optimale, et  $Sol$  la valeur de la solution retournée par l'algorithme, on a  $Opt \leq Sol \leq \alpha Opt$ .

A partir de cette définition, l'objectif devient donc de trouver, pour un problème donné, l'algorithme possédant le meilleur rapport d'approximation en pire cas possible. C'est ainsi que dans la littérature, on en vient à considérer qu'un algorithme est plus performant qu'un autre parce qu'il possède un meilleur rapport d'approximation. Cependant, il faut être conscient que cette mesure ne prend pas en compte la réalité de toutes les exécutions possibles d'un algorithme. On peut imaginer, par exemple et de manière extrêmement caricaturale, qu'un algorithme soit optimal pour toutes les instances possibles sauf une pour laquelle il retournera une solution dont le rapport d'approximation sera très grand (plus grand que celui des autres algorithmes d'approximation pour le même problème). Du point de vue de l'évaluation en pire cas, cet algorithme sera très mauvais alors que ses performances globales sont excellentes. Un autre problème de l'évaluation en pire cas provient du fait que pour une instance particulière, un rapport d'approximation en pire cas de  $\alpha$  (pour un problème de minimisation par exemple) peut situer la valeur d'une solution approchée au delà de la valeur de la pire solution possible pour cette instance.

#### 4.1.2.1 L'algorithme Edge Deletion (ED)

Cet algorithme, proposé par Gavril, repose sur le fait que les sommets d'un couplage maximal forment une couverture des sommets. Il retourne un vertex cover dont la taille est au plus  $2 \cdot |OPT|$  et s'exécute en un temps linéaire au nombre d'arêtes du graphe. La figure 4.7 présente un graphe pour lequel ED retourne toujours une solution (représentée par les arêtes doubles) qui est 2-approchée, tandis que la figure 4.8 présente un graphe pour lequel ED peut retourner une solution optimale (représentée elle aussi par une arête double).

---

#### Algorithm 2: Edge Deletion

---

**Data:** Un graphe  $G = (V, E)$

**Result:** Un vertex cover de  $G$

$C \leftarrow \emptyset;$

**while**  $E \neq \emptyset$  **do**

sélectionner  $uv \in E$ ;

$C \leftarrow C \cup \{u, v\}$ ;

$V \leftarrow V - \{u, v\}$ ;

**end**

**return**  $C$ ;

---

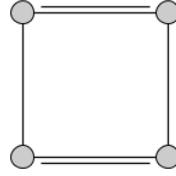


FIGURE 4.7 – Exemple de graphe pour lequel **Edge Deletion** atteint son rapport d'approximation en pire cas

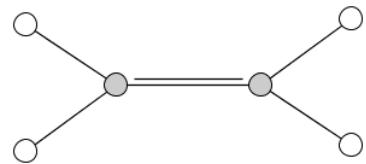


FIGURE 4.8 – Exemple de graphe pour lequel **Edge Deletion** peut retourner une solution optimale

**Un algorithme non déterministe.** Dans la boucle de l'algorithme **Edge Deletion**, le choix de l'arête à ajouter au couplage peut s'effectuer de plusieurs façons. On peut sélectionner une arête parmi l'ensemble des arêtes disponibles (c'est le choix qu'effectue l'algorithme que nous utilisons), mais il est possible de sélectionner un sommet non isolé, puis de sélectionner un de ses voisins. Cette dernière méthode semble retourner des couplages de taille plus importante que la première et donc des tailles de vertex cover plus importantes, c'est pour cette raison que nous ne l'utilisons pas ici.

**L'un des meilleurs rapport d'approximation en pire cas.** Étant donné qu'on ne connaît pas d'algorithme dont le rapport d'approximation en pire cas est borné par une constante meilleure que 2, et si la conjecture selon laquelle il n'en existe pas est vérifiée, alors on peut dire que **Edge Deletion** a atteint une certaine forme d'optimalité. Dans la littérature, cet algorithme est considéré comme le meilleur algorithme connu.

#### 4.1.2.2 L'algorithme Maximum Degree Greedy (MDG)

Cet algorithme est sûrement le premier auquel on pense lorsqu'on essaye de résoudre le problème du vertex cover. Soit  $G$  un graphe possédant un sommet de degré supérieur à  $k$ , avec  $k$  la taille d'une couverture minimum de  $G$ . Supposons qu'un algorithme choisisse de ne pas sélectionner le sommet de degré maximum. Il va donc être obligé de sélectionner tout son voisinage, sinon certaines arêtes ne seraient pas couvertes. Or, la taille du voisinage étant supérieure à  $k$ , l'algorithme est sûr de ne pas retourner une solution optimale. Ainsi, couvrir le maximum d'arêtes possibles à chaque étape en sélectionnant le sommet de degré maximum semble être une bonne idée :

---

##### Algorithm 3: Maximum Degree Greedy

---

```

Data: Un graphe  $G = (V, E)$ 
Result: Un vertex cover de  $G$ 
 $C \leftarrow \emptyset;$ 
while  $E \neq \emptyset$  do
    | sélectionner un sommet  $u$  de degré maximum;
    |  $V \leftarrow V - \{u\}$ ;
    |  $C \leftarrow C \cup \{u\}$ ;
end
return  $C$ ;

```

---

**Rapport d'approximation en pire cas.** L'algorithme **MDG** retourne un vertex cover dont la taille est au plus  $H(\Delta) \cdot |OPT|$ , avec  $H(n) = 1 + \frac{1}{2} + \dots + \frac{1}{n}$  la série harmonique et  $\Delta$  le degré

maximum du graphe.

**En pratique, quel est le meilleur algorithme ?** Bien que l'algorithme ED possède le meilleur rapport d'approximation en pire cas, l'algorithme MDG se comporte mieux, c'est-à-dire qu'il retourne des solutions de plus petite taille, sur de nombreux graphes. A vous d'expérimenter !

## 4.2 Le problème de l'ensemble indépendant maximum(stable)

**Définition 31 (Maximum Independent Set (MIS))** Étant donné un graphe  $G = (V, E)$ , avec  $V$  l'ensemble de ses sommets et  $E$  l'ensemble de ses arêtes, le problème de l'ensemble indépendant de taille maximum consiste à trouver le plus grand ensemble possible  $I \subseteq V$  tel que pour tout couple de sommets  $(a, b) \in I^2$ , l'arête  $(a, b) \notin E$ . On dit que  $I$  est un ensemble indépendant maximum de  $G$ .

**Équivalence.** Il est intéressant de remarquer que  $V - I$  est un vertex cover optimal. Ainsi, résoudre le problème de l'ensemble indépendant maximum revient à rechercher un vertex cover optimal.

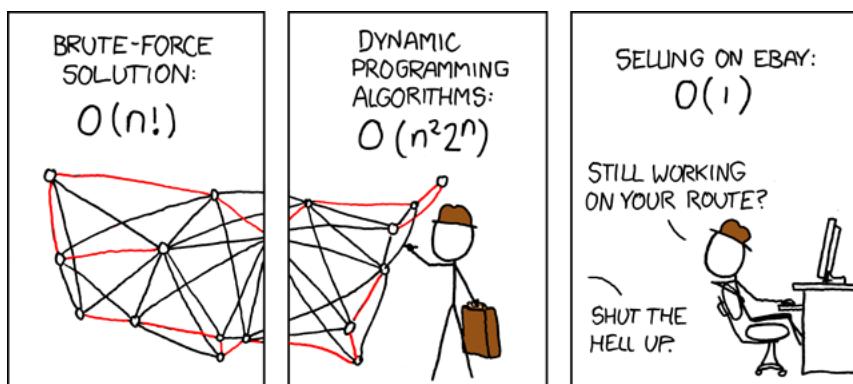
## 4.3 Le problème de la clique maximum

**Définition 32 (Clique maximum)** Étant donné un graphe  $G = (V, E)$ , avec  $V$  l'ensemble de ses sommets et  $E$  l'ensemble de ses arêtes, le problème de la clique maximum consiste à trouver le plus grand ensemble possible  $C \subseteq V$  tel que pour tout couple de sommets  $(a, b) \in C^2$ , l'arête  $(a, b) \in E$ . On dit que  $C$  est une clique maximum de  $G$ .

**Équivalence.** Résoudre le problème de la clique maximum revient à rechercher un ensemble indépendant de taille maximum dans le graphe complémentaire.

## 4.4 Le problème du voyageur de commerce

Un commercial représentant un laboratoire pharmaceutique doit présenter une nouvelle molécule aux différents hôpitaux de sa région. Chaque hôpital se trouve dans une ville différente. Afin d'optimiser son trajet (et donc son temps, ou son argent ou autre), il souhaite ne passer qu'une et une seule fois par chacune des villes concernées et revenir à son point de départ tout en minimisant la distance totale parcourue. Trouver une solution à ce problème est une chose très difficile en général.



L'un des meilleurs algorithmes exact, proposé par Held et Karp basé sur la programmation dynamique possède une complexité en  $O(n^2 2^n)$ . A titre d'information,  $2^{100}$  est un nombre plus grand que le nombre de secondes qui nous séparent du Big Bang jusqu'à aujourd'hui.

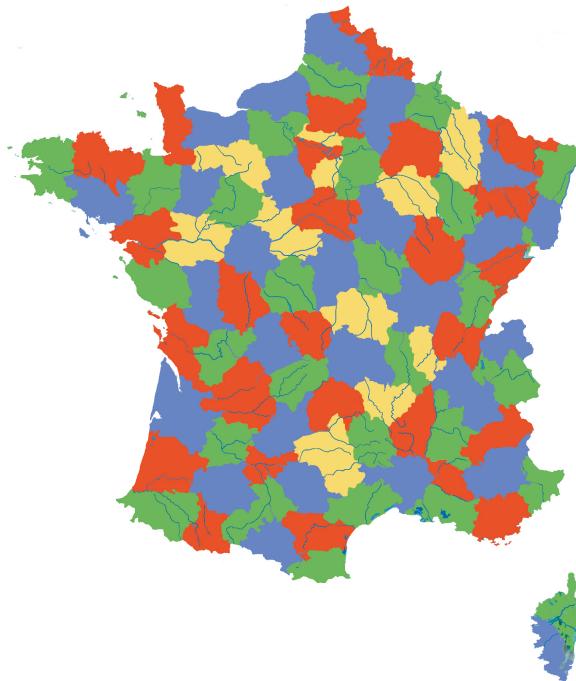
## 4.5 Le problème des 4 couleurs

Étant donné un graphe, attribuer une couleur à chacun de ses sommets de telle sorte que deux sommets reliés par une arête soient de couleur différente. On cherche, bien entendu, à utiliser un nombre minimum de couleurs. Ce nombre minimum de couleurs est aussi appelé nombre chromatique.

**Définition 33 (Graphe planaire)** *Un graphe planaire est un graphe pouvant être dessiné sans qu'aucune arêtes n'en croise une autre.*

**Le théorème des 4 couleurs.** Ce théorème, très célèbre, indique que toute carte planaire peut être coloriée avec au plus 4 couleurs distinctes de telle sorte qu'aucune zone d'une couleur n'ait de frontière commune (différente d'un point) avec une autre zone de la même couleur.

**La première preuve informatisée.** Ce qui est intéressant avec la preuve de ce théorème, c'est qu'il s'agit d'une des premières preuves faisant intervenir un ordinateur. En effet, les mathématiques ont permis de limiter le nombre de cas à analyser, mais ce nombre (1478) restait trop important pour être traité par un humain. L'outil informatique a ainsi été utilisé pour traiter ces différents cas (plus de 1200 heures de calcul), et démontrer ainsi la validité du théorème. Cependant, cette preuve a divisé la communauté scientifique, car il aurait fallu prouver que le programme utilisé se comportait bien comme prévu. C'est chose faite depuis 2005 avec une version entièrement formalisée, formulée avec Coq.





## Chapitre 5

# Parcours dans les graphes

Dans cette partie, nous allons étudier plusieurs algorithmes parmi les plus célèbres de la théorie des graphes. Ces algorithmes sont fondamentaux, dans le sens ils sont souvent utilisés pour concevoir d'autres algorithmes.

### 5.1 Atteignabilité, algorithme de Roy-Warshall

L'algorithme de Warshall va calculer la fermeture transitive d'un graphe (orienté ou non). C'est-à-dire que partant d'un graphe  $G = (V, A)$ , l'algorithme va produire un second graphe  $G' = (V, A')$  tel que si il existe un chemin dans  $G$  allant d'un sommet  $u$  à un sommet  $v$  alors  $(u, v) \in A'$ . Cet algorithme va nous permettre de connaître l'ensemble des relations d'accessibilité au sein du graphe  $G$ . Cet algorithme permet donc d'obtenir des informations sur les composantes connexes ou fortement connexes d'un graphe. En particulier, une fois  $G'$  généré, il est possible de savoir en temps constant, par une simple vérification d'existence d'arête, si deux sommets font partie de la même composante connexe ou fortement connexe, et donc de savoir si il existe un chemin entre ces deux sommets. La complexité de cet algorithme est en  $O(n^3)$ , ce qui le rend couteux. Il est donc préférable de le calculer une fois, et de le sauvegarder.

---

#### Algorithm 4: Algorithme de Roy-Warshall

---

**Data:**  $G = (V, A)$  un graphe (orienté ou non)  
**Réultat:** Un graphe  $G'$  représentant la fermeture transitive de  $G$

```
A' = A;  
forall u ∈ V do  
    forall v ∈ N-(u) do  
        forall w ∈ N+(u) do  
            | A' = A' ∪ (v, w);  
            | end  
        | end  
    | end  
end  
return G' = (V, A');
```

---

### **5.1.1 Exemple de déroulement de l'algorithme.**

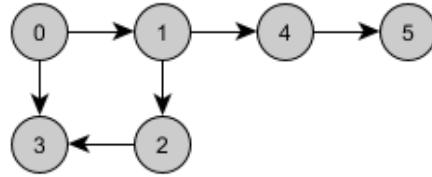


FIGURE 5.1 – Graphe de départ

#### **5.1.1.1 Étape 1. Considérons le sommet 0.**

Le voisinage entrant de 0 est vide. On passe au sommet suivant.

#### **5.1.1.2 Étape 2. Considérons le sommet 1.**

$N^-(1) = \emptyset$  et  $N^+(1) = \{2, 4\}$ . Ainsi, on ajoute les deux arcs  $(0, 2)$  et  $(0, 4)$ .

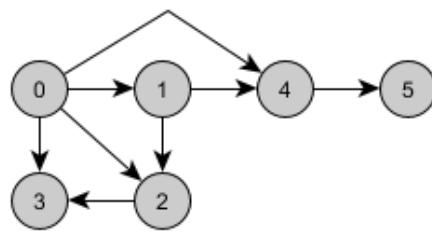


FIGURE 5.2 – Graphe augmenté des arêtes  $(0, 2)$  et  $(0, 4)$

#### **5.1.1.3 Étape 3. Considérons le sommet 2.**

$N^-(2) = \{0, 1\}$  et  $N^+(2) = \{3\}$ . L'arête  $(0, 3)$  existe déjà. Ainsi, on ajoute uniquement l'arc  $(1, 3)$ .

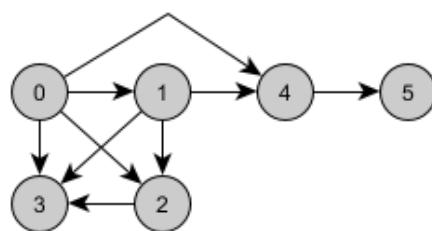


FIGURE 5.3 – Graphe augmenté de l'arête  $(1, 3)$

#### **5.1.1.4 Étape 4. Considérons le sommet 3.**

Le voisinage sortant de 3 est vide. On passe au sommet suivant.

#### 5.1.1.5 Étape 5. Considérons le sommet 4.

$N^-(4) = \{0, 1\}$  et  $N^+(4) = \{5\}$ . Ainsi, on ajoute les deux arcs  $(0, 5)$  et  $(1, 5)$ .

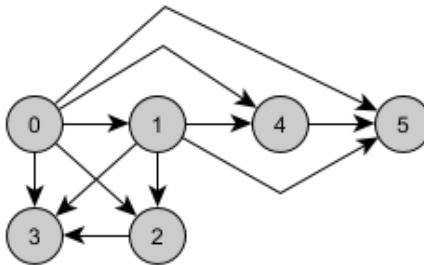
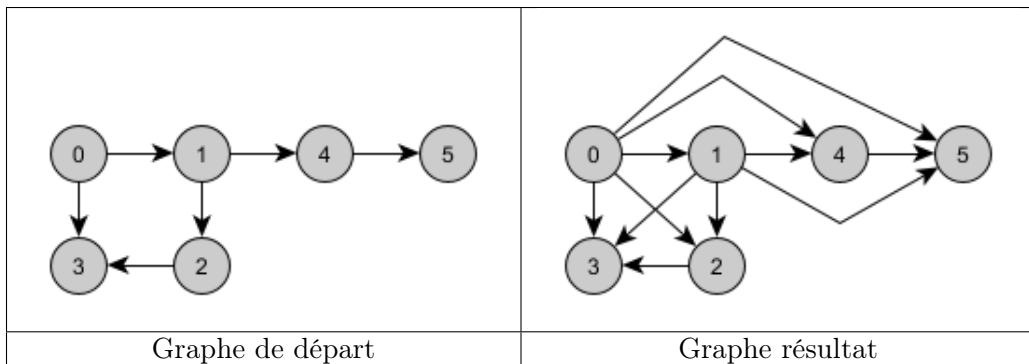


FIGURE 5.4 – Graphe augmenté des arêtes  $(0, 5)$  et  $(1, 5)$

#### 5.1.1.6 Étape 6. Considérons le sommet 5.

Le voisinage sortant de 5 est vide. L'algorithme se termine.

#### 5.1.1.7 Résultat et utilisation



1. Est-ce que le sommet 5 est atteignable depuis le sommet 0 ? L'arc  $(0, 5)$  existe dans le graphe résultat, donc oui.
2. Est-ce que le sommet 5 est atteignable depuis le sommet 3 ? L'arc  $(3, 5)$  n'existe pas dans le graphe résultat, donc non.

## 5.1.2 Parcours en largeur

Cet algorithme permet de parcourir un graphe depuis un sommet d'origine. Il va parcourir les sommets du graphe, de manière concentrique. Les sommets situés à une distance 1 de l'origine, puis les sommets à distance 2 etc. Grâce à ce parcours, nous allons obtenir un arbre des plus courts chemins (si le graphe est non pondéré) et il sera également possible de savoir si le graphe est connexe.

### 5.1.2.1 Algorithme

---

**Algorithm 5:** Parcours\_en\_Largeur( $G, s$ )

---

**Data:**  $G = (V, E)$  un graphe et  $s \in V$  un sommet du graphe

**Réultat:** Un parcours en largeur issu du sommet  $s$

**forall**  $u \in V - \{s\}$  **do**

couleur[u]=Blanc;

distance[u]= $\infty$ ;

pere[u]=NIL; 

**end**

couleur[s]=Gris;

distance[s]=0;

pere[s]=NIL;

F={s};

**while**  $F \neq \emptyset$  **do**

u=Defiler[F];

**for**  $v \in N(u)$  **do**

**if** couleur[v]=Blanc **then**

couleur[v]=Gris;

distance[v]=distance[u]+1;

pere[v]=u;

Enfiler(F,v);

**end**

**end**

couleur[u]=Noir;

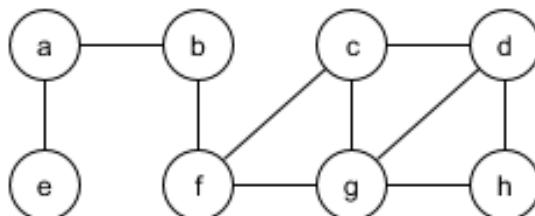
**end**

---

### 5.1.2.2 Exemple d'application de l'algorithme

Nous allons voir un exemple de déroulement de cet algorithme. Nous reprenons l'exemple classique, repris dans de nombreux ouvrages tels que [2].

**Graphe initial** Nous allons appliquer l'algorithme sur ce graphe, en partant du sommet b.

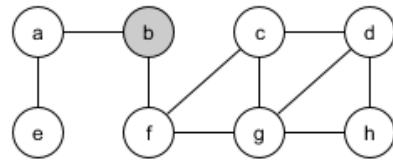


### Phase d'initialisation

On place la couleur du sommet b à Gris, et celle de tous les autres sommets à Blanc. Le sommet b est à une distance de 0 de lui-même, et tous les autres sommets sont, pour le moment, à une distance infinie. Le sommet b est ajouté à la file F.

Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	B	B	B	B	B	B
Distance	$\infty$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
Père	-	-	-	-	-	-	-	-

$$F = [b]$$

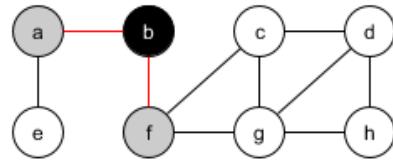


### Étape 1.

On défile le premier élément de la file F, soit le sommet b. On parcourt ses voisins de couleur blanche, soient les sommets f et a (peu importe l'ordre de ces sommets). Leur couleur devient grise, leur distance passe à  $0 + 1 = 1$  et ils sont atteignables par le sommet b, qui devient donc leur père. Le sommet b devient noir.

Sommet	a	b	c	d	e	f	g	h
Couleur	G	N	B	B	B	G	B	B
Distance	1	0	$\infty$	$\infty$	$\infty$	1	$\infty$	$\infty$
Père	b	-	-	-	-	b	-	-

$$F = [f, a]$$

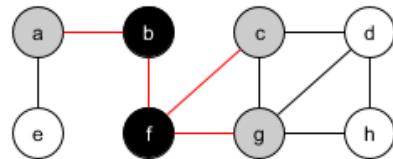


### Étape 2.

On défile le premier élément de la file F, soit le sommet f. On parcourt ses voisins de couleur blanche, soient les sommets c et g (peu importe l'ordre de ces sommets). Leur couleur devient grise, leur distance passe à  $1 + 1 = 2$  et ils sont atteignables par le sommet f, qui devient donc leur père. Le sommet f devient noir.

Sommet	a	b	c	d	e	f	g	h
Couleur	G	N	G	B	B	N	G	B
Distance	1	0	2	$\infty$	$\infty$	1	2	$\infty$
Père	b	-	f	-	-	b	f	-

$$F = [a, c, g]$$

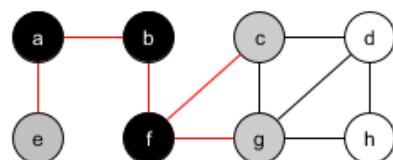


### Étape 3.

On défile le premier élément de la file F, soit le sommet a. On parcourt ses voisins de couleur blanche (il n'y en a qu'un), soit le sommet e. Sa couleur devient grise, sa distance passe à  $1 + 1 = 2$  et il est atteignable par le sommet a, qui devient donc son père. Le sommet a devient noir.

Sommet	a	b	c	d	e	f	g	h
Couleur	N	N	G	B	G	N	G	B
Distance	1	0	2	$\infty$	2	1	2	$\infty$
Père	b	-	f	-	a	b	f	-

$$F = [c, g, e]$$



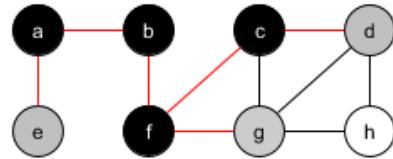
### Étape 4.

On défile le premier élément de la file F, soit le sommet c. On parcourt ses voisins de couleur blanche (il n'y en a qu'un), soit le sommet d. Sa couleur devient grise, sa distance passe à  $2 + 1 = 3$  et il est atteignable par le sommet c, qui devient donc son père. Le sommet c devient

noir.

Sommet	a	b	c	d	e	f	g	h
Couleur	N	N	N	G	G	N	G	B
Distance	1	0	2	3	2	1	2	$\infty$
Père	b	-	f	c	a	b	f	-

$$F = [g, e, d]$$

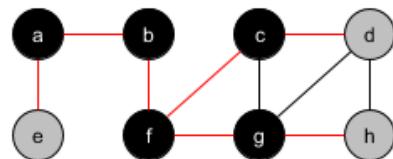


### Étape 5.

On défile le premier élément de la file F, soit le sommet g. On parcourt ses voisins de couleur blanche (il n'y en a qu'un), soit le sommet h. Sa couleur devient grise, sa distance passe à  $2 + 1 = 3$  et il est atteignable par le sommet g, qui devient donc son père. Le sommet g devient noir.

Sommet	a	b	c	d	e	f	g	h
Couleur	N	N	N	G	G	N	N	G
Distance	1	0	2	3	2	1	2	3
Père	b	-	f	c	a	b	f	g

$$F = [e, d, h]$$

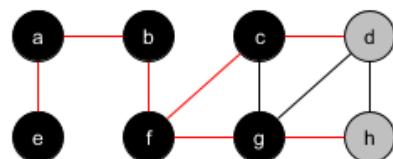


### Étape 6.

On défile le premier élément de la file F, soit le sommet e. On parcourt ses voisins de couleur blanche (il n'y en a aucun). Le sommet e devient noir.

Sommet	a	b	c	d	e	f	g	h
Couleur	N	N	N	G	N	N	N	G
Distance	1	0	2	3	2	1	2	3
Père	b	-	f	c	a	b	f	g

$$F = [d, h]$$

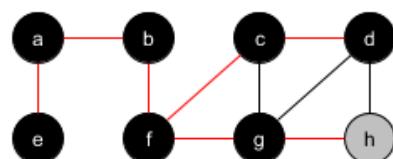


### Étape 7.

On défile le premier élément de la file F, soit le sommet d. On parcourt ses voisins de couleur blanche (il n'y en a aucun). Le sommet d devient noir.

Sommet	a	b	c	d	e	f	g	h
Couleur	N	N	N	N	N	N	N	G
Distance	1	0	2	3	2	1	2	3
Père	b	-	f	c	a	b	f	g

$$F = [h]$$

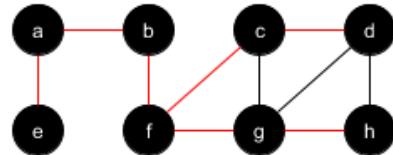


### Étape 8.

On défile le premier élément de la file F, soit le sommet h. On parcourt ses voisins de couleur blanche (il n'y en a aucun). Le sommet h devient noir. La file F est vide, l'algorithme se termine.

Sommet	a	b	c	d	e	f	g	h
Couleur	N	N	N	N	N	N	N	N
Distance	1	0	2	3	2	1	2	3
Père	b	-	f	c	a	b	f	g

$$F = []$$



**Note.** Si l'un des sommets se trouve à une distance infinie, cela signifie que le graphe n'est pas connexe. De plus, à partir d'un sommet, et en remontant père après père, on obtient un plus court chemin vers l'origine. Par exemple, le père du sommet h est le sommet g. Le père du sommet g est le sommet f. Le père du sommet f est le sommet b. Nous obtenons ainsi une chaîne reliant le sommet h au sommet b : h - g - f - b.

### 5.1.3 Parcours en profondeur

L'algorithme du parcours en profondeur permet, tout comme le parcours en largeur, de vérifier si un graphe est connexe. Cependant, il ne permet pas de calculer les plus courts chemins à partir du sommet initial, du fait de l'ordre de parcours des sommets. Cet ordre va permettre de détecter la présence de cycles, ce qui est très utile dans le cas d'un ensemble de tâches dépendantes les unes des autres pour vérifier qu'il n'existe pas de cycle de dépendance.

#### 5.1.3.1 Pseudo code de l'algorithme

La version présentée ici est récursive, mais il est tout à fait possible d'en réaliser une version itérative en utilisant une pile. Une première fonction (itérative), *DFS*, se charge de l'initialisation et d'appeler la fonction *DFS\_Visit* (réursive) qui va se charger de parcourir effectivement le graphe. La fonction *DFS* permet de gérer le cas où le graphe ne serait pas connexe car elle va appeler la fonction *DFS\_Visit* pour chaque sommet non encore parcouru. Ainsi, le résultat de cet algorithme sera une forêt de parcours en profondeur.

---

#### Algorithm 6: $\text{DFS}(G)$

---

**Data:**  $G = (V, E)$  un graphe

**Résultat:** Une forêt de parcours en profondeur du graphe  $G$ .

**forall**  $u \in V$  **do**

couleur[u]=Blanc;  
pere[u]=NIL;

**end**

**temps** = 0;

**forall**  $u \in V$  **do**

**if** couleur[v]=Blanc **then**  
| DFS\_Visit(G,u);  
**end**

**end**

---

---

**Algorithm 7:** DFS\_Visit( $G, u$ )

---

**Data:**  $G = (V, E)$  un graphe  
**Réultat:** Une forêt de parcours en profondeur du graphe  $G$ .  
couleur[u]=Gris;  
debut[u]=temps;  
temps=temps+1;  
**forall**  $v \in N(u)$  **do**  
    **if** couleur[v]=Blanc **then**  
        pere[v]=u;  
        DFS\_Visit(G,v);  
    **end**  
**end**  
couleur[u]=Noir;  
fin[u]=temps;  
temps=temps+1;

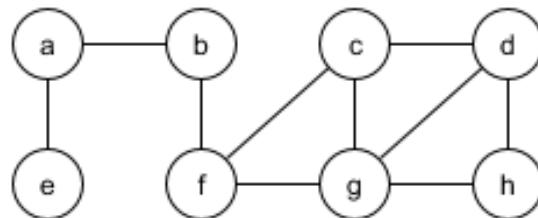
---

Nous allons noter le temps (discret) de la coloration d'un sommet en gris, qui correspond au moment de la première visite de ce sommet. Nous allons également noter le moment de la coloration en noir d'un sommet, qui correspond au moment de la fin de son traitement. Pour cela, nous utilisons une variable nommée *temps*, initialisée à 0 qui va augmenter de 1 lors de chaque étape.

#### 5.1.4 Exemple d'application de l'algorithme

Voici un exemple de déroulement de cet algorithme. Nous reprenons l'exemple vu pour le parcours en largeur. Mais il faut bien comprendre que cet algorithme peut s'appliquer sur un graphe orienté, qu'il soit connexe ou non.

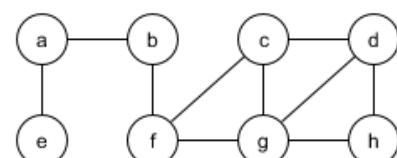
**Graphe initial** Nous allons appliquer l'algorithme sur le graphe suivant. La fonction DFS va considérer les sommets dans l'ordre (arbitraire) suivant :  $\{b, c, d, e, f, g, h, a\}$  :



#### Phase d'initialisation

Tous les sommets sont initialisés à la couleur blanche, et les pères à NIL.

Sommet	a	b	c	d	e	f	g	h
Couleur	B	B	B	B	B	B	B	B
Père	-	-	-	-	-	-	-	-
Debut								
Fin								

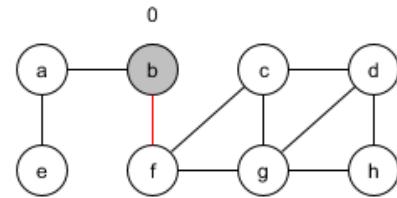


La fonction  $DFS$  va réaliser un appel à la fonction  $DFS\_Visit$  pour chacun des sommets si, au moment de l'appel, le sommet est de couleur blanche. En particulier, c'est le cas du premier voisin considéré. Un appel est donc effectué en partant du sommet  $b$  :  $DFS\_Visit(G, b)$ .

### Étape 1.

Le sommet  $b$  devient gris. Le premier passage (debut) se fait lorsque temps vaut 0. S'agissant du premier sommet, il n'a pas de père. On augmente le temps de 1. On considère ensuite les voisins de  $b$  dans l'ordre suivant :  $\{f, a\}$ , par exemple. La couleur du sommet  $f$  est blanche, donc le père de  $f$  devient  $b$  et on réalise un appel récursif sur le sommet  $f$ .

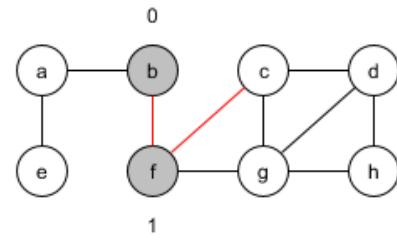
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	B	B	B	B	B	B
Père	-	-	-	-	-	b	-	-
Debut		0						
Fin								



### Étape 2.

Le sommet  $f$  devient gris. Le premier passage (debut) se fait lorsque temps vaut 1. On augmente le temps de 1. On considère les voisins de  $f$  dans l'ordre suivant :  $\{b, c, g\}$ , par exemple. La couleur du sommet  $b$  n'est pas blanche, donc on ne réalise pas d'appel récursif sur ce sommet. On considère ensuite le sommet  $c$ . Sa couleur est blanche donc le père de  $c$  devient  $f$  et on réalise un appel récursif sur  $c$ .

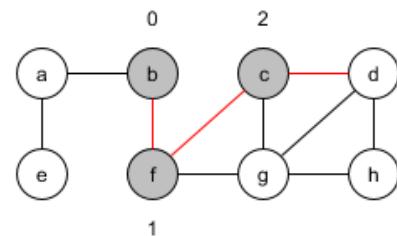
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	B	B	B	G	B	B
Père	-	-	f	-	-	b	-	-
Debut		0				1		
Fin								



### Étape 3.

Le sommet  $c$  devient gris. Le premier passage (debut) se fait lorsque temps vaut 2. On augmente le temps de 1. On considère les voisins de  $c$  dans l'ordre suivant :  $\{d, f, g\}$ , par exemple. On considère le sommet  $d$ . Sa couleur est blanche donc le père de  $d$  devient  $c$  et on réalise un appel récursif sur  $d$ .

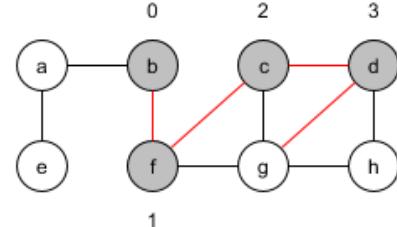
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	G	B	B	G	B	B
Père	-	-	f	c	-	b	-	-
Debut		0	2			1		
Fin								



#### Étape 4.

Le sommet d devient gris. Le premier passage (debut) se fait lorsque temps vaut 3. On augmente le temps de 1. On considère les voisins de  $d$  dans l'ordre suivant :  $\{c, g, h\}$ , par exemple. On considère le sommet c. Sa couleur n'est pas blanche donc on ne réalise pas d'appel récursif sur ce sommet. On considère ensuite le sommet g. Sa couleur est blanche donc le père de g devient d et on réalise un appel récursif sur g.

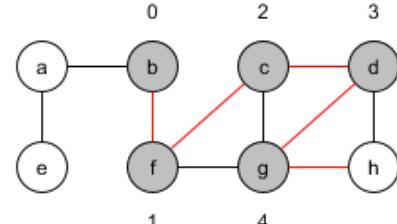
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	G	G	B	G	B	B
Père	-	-	f	c	-	b	d	-
Debut		0	2	3		1		
Fin								



#### Étape 5.

Le sommet g devient gris. Le premier passage (debut) se fait lorsque temps vaut 4. On augmente le temps de 1. On considère les voisins de  $g$  dans l'ordre suivant :  $\{c, d, f, h\}$ , par exemple. On considère le sommet c. Sa couleur n'est pas blanche donc on ne réalise pas d'appel récursif sur ce sommet. On considère le sommet d. Sa couleur n'est pas blanche donc on ne réalise pas d'appel récursif sur ce sommet. On considère le sommet f. Sa couleur n'est pas blanche donc on ne réalise pas d'appel récursif sur ce sommet. On considère ensuite le sommet h. Sa couleur est blanche donc le père de h devient g et on réalise un appel récursif sur h.

Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	G	G	B	G	G	B
Père	-	-	f	c	-	b	d	g
Debut		0	2	3		1	4	
Fin								

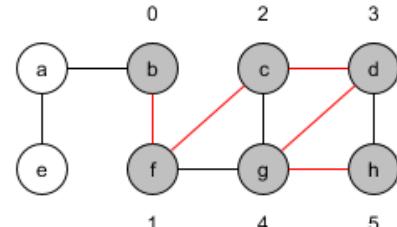


temps = 5.

#### Étape 6.

Le sommet h devient gris. Le premier passage (debut) se fait lorsque temps vaut 5. On augmente le temps de 1. On considère les voisins de  $h$  dans l'ordre suivant :  $\{d, g\}$ , par exemple. On considère le sommet d. Sa couleur n'est pas blanche donc on ne réalise pas d'appel récursif sur ce sommet. On considère le sommet g. Sa couleur n'est pas blanche donc on ne réalise pas d'appel récursif sur ce sommet. La phase d'appels récursifs sur le voisinage du sommet  $h$  est terminée.

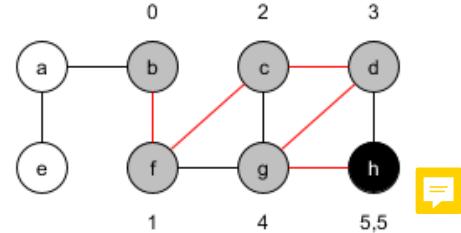
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	G	G	B	G	G	G
Père	-	-	f	c	-	b	d	g
Debut		0	2	3		1	4	5
Fin								



### Étape 7.

La phase d'appels récursifs sur le voisinage du sommet  $h$  est terminée. Le sommet  $h$  devient noir. Le moment de fin de traitement vaut 5. On augmente le temps de 1. L'appel récursif se termine. On retourne sur le traitement des voisins du sommet  $g$ .

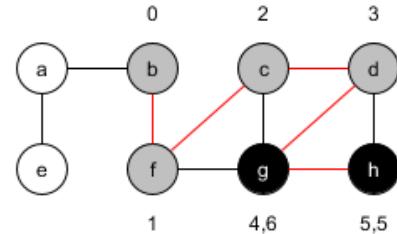
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	G	G	B	G	G	N
Père	-	-	f	c	-	b	d	g
Début	0	2	3		1	4	5	
Fin							5	



### Étape 8.

La phase d'appels récursifs sur le voisinage du sommet  $g$  est terminée. Le sommet  $g$  devient noir. Le moment de fin de traitement vaut 6. On augmente le temps de 1. L'appel récursif se termine. On retourne sur le traitement des voisins du sommet  $d$ .

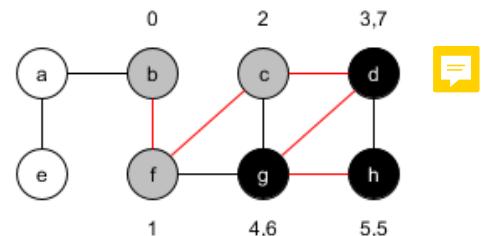
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	G	G	B	G	N	N
Père	-	-	f	c	-	b	d	g
Début	0	2	3		1	4	5	
Fin							6	5



### Étape 9.

Le voisinage du sommet  $d$  contient les sommets  $\{c, g, h\}$ . Il reste à traiter le cas du sommet  $h$ . La couleur du sommet  $h$  n'est pas blanche, donc on ne réalise pas d'appel récursif. La phase d'appels récursifs sur le voisinage du sommet  $d$  est terminée. Le moment de fin de traitement vaut 7. On augmente le temps de 1. L'appel récursif se termine. On retourne sur le traitement des voisins du sommet  $c$ .

Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	G	G	B	G	N	N
Père	-	-	f	c	-	b	d	g
Début	0	2	3		1	4	5	
Fin				7			6	5



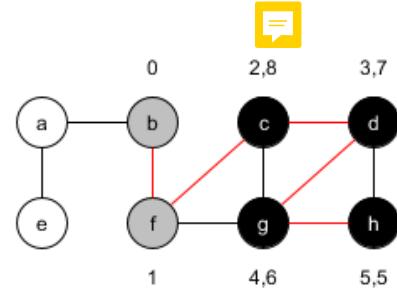
### Étape 10.

Le voisinage du sommet  $c$  contient les sommets  $\{d, f, g\}$ . Il reste à traiter le cas des sommets  $f$  et  $g$ . La couleur de ces deux sommets n'est pas blanche, donc on ne réalise pas d'appel récursif. La phase d'appels récursifs sur le voisinage du sommet  $c$  est terminée. Le sommet  $c$  devient noir. Le temps passe à 9. L'appel récursif se termine. On retourne sur le traitement des voisins du

sommet  $f$ .

Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	N	G	B	G	N	N
Père	-	-	f	c	-	b	d	g
Début	0	2	3		1	4	5	
Fin		8	7			6	5	

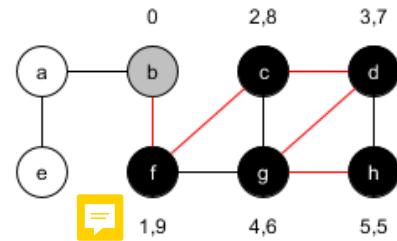
temps = 9. On considère les voisins de  $f$  dans l'ordre suivant :  $\{b, c, g\}$ , par exemple. La couleur du sommet  $b$  n'est pas blanche, donc on ne réalise pas d'appel récursif.



### Étape 11.

Le voisinage du sommet  $f$  contient les sommets  $\{b, c, g\}$ . Il reste à traiter le cas du sommet  $g$ . La couleur de ce sommet n'est pas blanche, donc on ne réalise pas d'appel récursif. La phase d'appels récursifs sur le voisinage du sommet  $f$  est terminée. Le sommet  $f$  devient noir. Le temps passe à 10. L'appel récursif se termine. On retourne sur le traitement des voisins du sommet  $b$ .

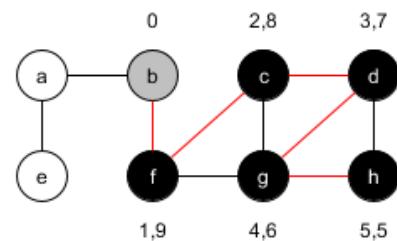
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	N	G	B	N	N	N
Père	-	-	f	c	-	b	d	g
Début	0	2	3		1	4	5	
Fin		8	7			6	5	



### Étape 12.

Le voisinage du sommet  $b$  contient les sommets  $\{a, f\}$ . Il reste à traiter le cas du sommet  $a$ . La couleur de ce sommet est blanche, donc on réalise un appel récursif sur le sommet  $a$ .

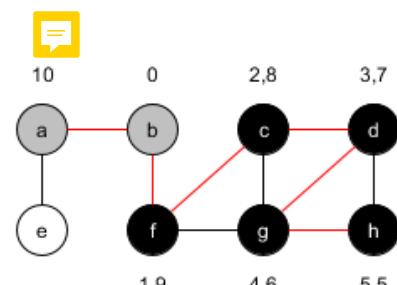
Sommet	a	b	c	d	e	f	g	h
Couleur	B	G	N	G	B	N	N	N
Père	b	-	f	c	-	b	d	g
Début	0	2	3		1	4	5	
Fin		8	7			6	5	



### Étape 13.

Sommet	a	b	c	d	e	f	g	h
Couleur	G	G	N	G	B	N	N	N
Père	b	-	f	c	-	b	d	g
Début	10	0	2	3		1	4	5
Fin		8	7			6	5	

temps = 11. On considère les voisins de  $a$  dans l'ordre suivant :  $\{b, e\}$ , par exemple. La couleur du sommet  $b$  n'est pas blanche, donc on ne réalise pas d'appel récursif.

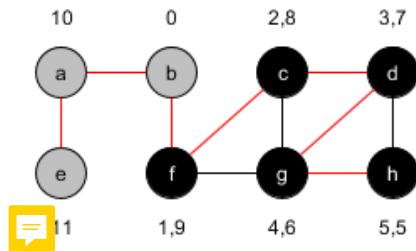


On réalise un appel récursif sur le sommet  $e$ .

### Étape 14.

Sommet	a	b	c	d	e	f	g	h
Couleur	G	G	N	G	B	N	N	N
Père	b	-	f	c	-	b	d	g
Début	10	0	2	3	11	1	4	5
Fin			8	7		9	6	5

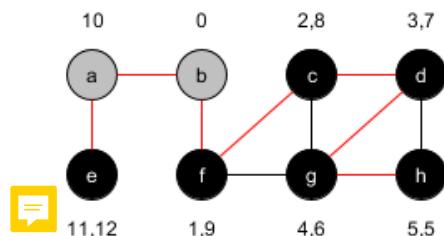
temps = 12. On considère le seul voisin du sommet  $e$  est le sommet  $a$  et sa couleur n'est pas blanche donc on ne réalise pas d'appel récursif.



### Étape 15.

La phase d'appels récursifs sur le voisinage du sommet  $e$  est terminée. Le sommet  $e$  devient noir. Le temps passe à 13. L'appel récursif se termine. On retourne sur le traitement des voisins du sommet  $a$ .

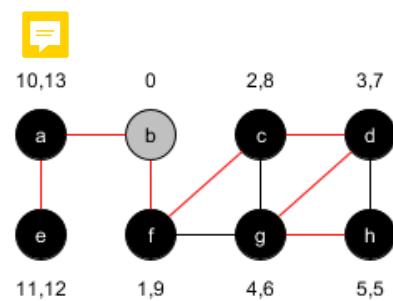
Sommet	a	b	c	d	e	f	g	h
Couleur	G	G	N	N	B	N	N	N
Père	b	-	f	c	-	b	d	g
Début	10	0	2	3	11	1	4	5
Fin			8	7	12	9	6	5



### Étape 16.

La phase d'appels récursifs sur le voisinage du sommet  $a$  est terminée. Le sommet  $a$  devient noir. Le temps passe à 14. L'appel récursif se termine. On retourne sur le traitement des voisins du sommet  $b$ .

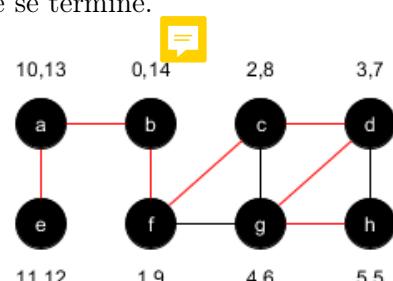
Sommet	a	b	c	d	e	f	g	h
Couleur	N	G	N	N	B	N	N	N
Père	b	-	f	c	-	b	d	g
Début	10	0	2	3	11	1	4	5
Fin	13		8	7	12	9	6	5



### Étape 17.

La phase d'appels récursifs sur le voisinage du sommet  $b$  est terminée. Le sommet  $b$  devient noir. Le temps passe à 15. L'appel récursif se termine. L'algorithme se termine.

Sommet	a	b	c	d	e	f	g	h
Couleur	N	N	N	N	B	N	N	N
Père	b	-	f	c	a	b	d	g
Début	10	0	2	3	11	1	4	5
Fin	13	14	8	7	12	9	6	5



#### 5.1.4.1 Propriétés et applications

1. Trouve l'ensemble des sommets accessibles depuis un sommet donné.
2. Effectue un tri topologique des sommets.
3. Détermine si un graphe possède un cycle et permet d'en exhiber un.

## 5.2 Plus courts chemins

De nos jours, il est rare d'envisager un long trajet en voiture sans GPS, ou pour le moins, sans avoir déterminé le trajet à effectuer grâce à un site internet spécialisé. Nous allons voir dans cette partie comment déterminer le plus court chemin entre deux sommets d'un graphe pondéré pour lequel les sommets représentent des villes ou des intersections, les arcs les routes existantes et le poids d'un arc la distance à parcourir.

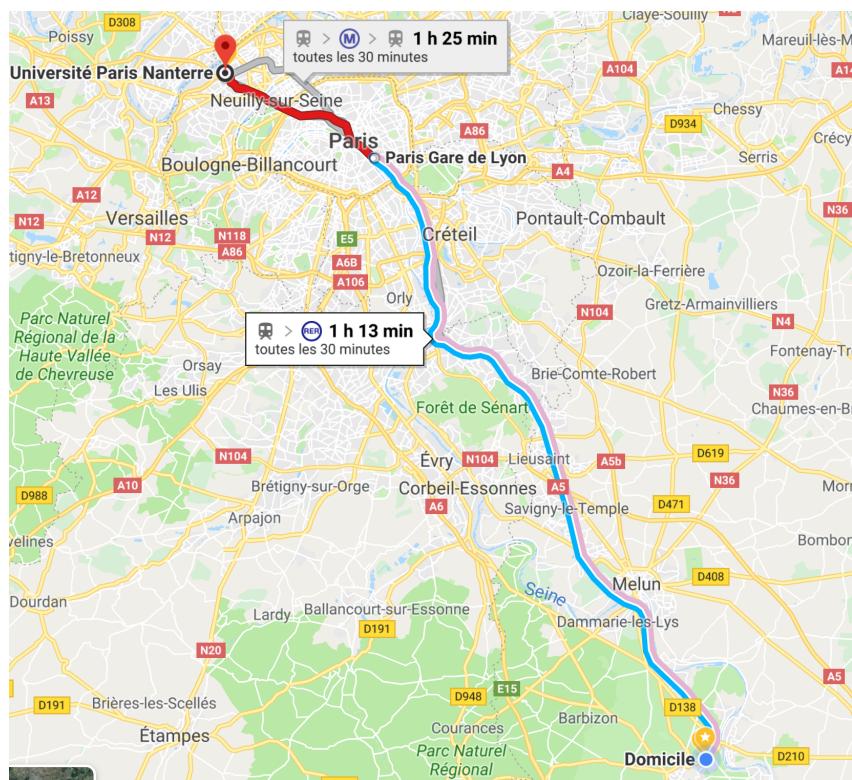


FIGURE 5.5 – Exemple de proposition de trajet obtenue grâce à google maps pour se rendre de la gare de Fontainebleau-Avon à l'Université Paris Nanterre.

#### 5.2.1 Un problème, trois versions

Trouver le chemin le plus court allant d'un point A à un point B peut être considéré de trois manières différentes, qui semblent différentes dans leur difficulté :

1. Plus court chemin entre deux sommets du graphe.
2. Plus court chemin d'un sommet vers tous les autres.
3. Plus courts chemin de chaque sommet vers tous les autres.

Il se trouve qu'avec les algorithmes à notre disposition, trouver un plus court chemin depuis un sommet vers tous les autres n'est pas beaucoup plus coûteux que de calculer le plus court chemin entre deux sommets. Et la troisième version est résolue en appliquant la deuxième à chaque sommet. Nous allons donc nous concentrer sur la deuxième version du problème.

**Important.** Certains graphes ne possèdent pas de plus court chemin d'un sommet à un autre sommet. Cela peut provenir du fait que le graphe n'est pas connexe. Ce cas est généralement réglé sans effort puisque les sommets inaccessibles seront, dans les diverses solutions proposées par les algorithmes, situés à une distance infinie. Le second cas est plus problématique. Considérons le graphe suivant :

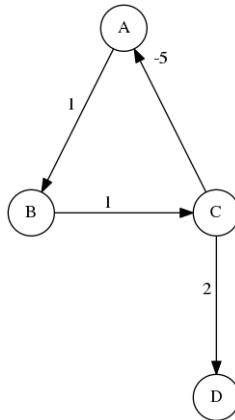


FIGURE 5.6 – Exemple de graphe possédant un cycle de poids négatif

Dans ce graphe, on peut constater que la somme des poids associés au cycle  $A \rightarrow B \rightarrow C \rightarrow A$  est de  $-3$ . Cela signifie que chaque fois que l'on fait un tour de cycle, le poids va diminuer de 3. Dans ce cas, il n'existe pas de plus court chemin. En effet, si un tel chemin existait il suffirait d'ajouter un tour de cycle pour en obtenir un de poids strictement plus faible.

**Restriction.** Nous ne considérons donc par la suite que les graphes sans cycle de poids négatif.

### 5.2.2 Algorithme de Dijkstra

L'algorithme de Dijkstra permet de déterminer le plus court chemin d'un sommet origine vers tous les autres. Il est conçu pour s'appliquer sur des graphes orientés et pondérés.

**Restriction importante :** Cet algorithme ne fonctionne que si, pour chaque arc  $(u, v)$  le poids de l'arc  $(u, v) \geq 0$ .

**Bonne nouvelle 1 :** Si le graphe est non pondéré, on peut considérer que chaque arc possède un poids de 1. L'algorithme est donc utilisable sur les graphes non pondérés.

**Bonne nouvelle 2 :** Si le graphe est non orienté, on peut considérer que chaque arête  $(u, v)$  représente les deux arcs  $(u, v)$  et  $(v, u)$ . L'algorithme est donc utilisable sur un graphe non orienté.

---

**Algorithm 8:** Dijkstra( $G, s$ )

---

**Data:**  $G = (V, E)$  un graphe orienté pondéré et  $s \in V$  un sommet du graphe

**Résultat:** Un arbre des plus courts chemins d'origine  $s$

**foreach**  $u \in V$  **do**

$Distance[u] = \infty;$

$Pere[u] = NIL;$

**end**

$Distance[s] = 0;$

$F = V;$

**while**  $F \neq \emptyset$  **do**

$u = Extraire - Min(F);$

        Choisir un sommet  $u \in F$  tel que  $Distance[u]$  soit minimum;

$F = F - u;$

**foreach**  $v \in N(u)$  **do**

**if**  $Distance[v] > Distance[u] + Poids(u, v)$  **then**

$Distance[v] = Distance[u] + Poids(u, v);$

$Pere[v] = u;$

**end**

**end**

**end**

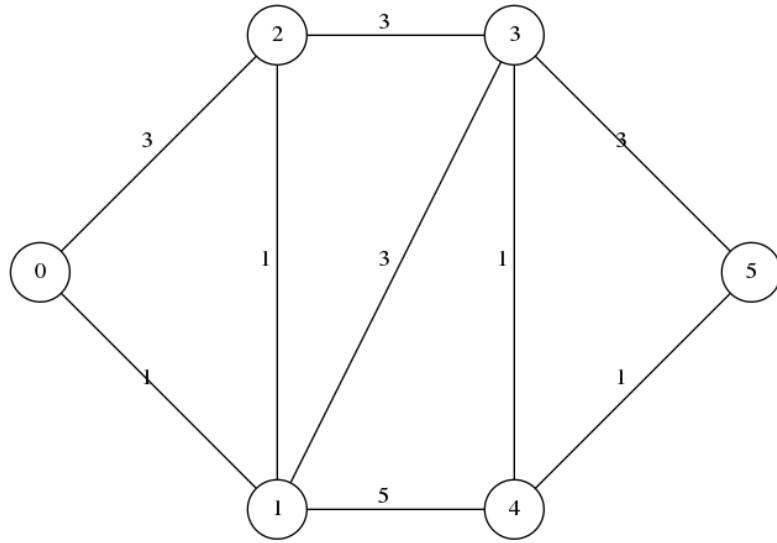
### 5.2.3 Explications succinctes

- Phase d'initialisation.
  - Chaque sommet est à une distance infinie ne possède pas de père.
  - Le tableau des distances va nous permettre de maintenir la plus courte distance allant de l'origine vers chaque sommet.
  - le tableau des pères nous permettra de savoir depuis quel sommet on arrive. A la fin, ce tableau permet de construire un arbre des plus courts chemins ayant pour racine le sommet d'origine.
  - Le sommet de départ est à une distance de 0 de lui-même.
  - L'ensemble  $F$  contient la liste des sommets non traités. A chaque étape, nous allons en choisir un. Il y a donc autant d'étape que de sommet.
- Boucle principale. Continue tant que l'ensemble  $F$  n'est pas vide.
  - La fonction Extraire-Min( $F$ ) parcourt les sommets présents dans  $F$ . On sélectionne le sommet  $u$  qui possède la distance la plus faible.
  - Le sommet est retiré de  $F$ .
  - Pour chaque arc sortant on va regarder si cet arc permet d'améliorer un chemin déjà connu.
  - Si c'est le cas, on met à jour la distance du sommet atteint. Son père devient  $u$ , puisque c'est grâce à cet arc que l'on améliore le chemin.

### 5.2.4 Exemple d'application de l'algorithme

Dans cet exemple, nous allons considérer un graphe pondéré et non orienté en partant du sommet 0. Il est important de noter que l'algorithme peut tout faire, et sans aucune modification, s'appliquer sur un graphe orienté et/ou non pondéré.

#### 5.2.4.1 Graphe initial

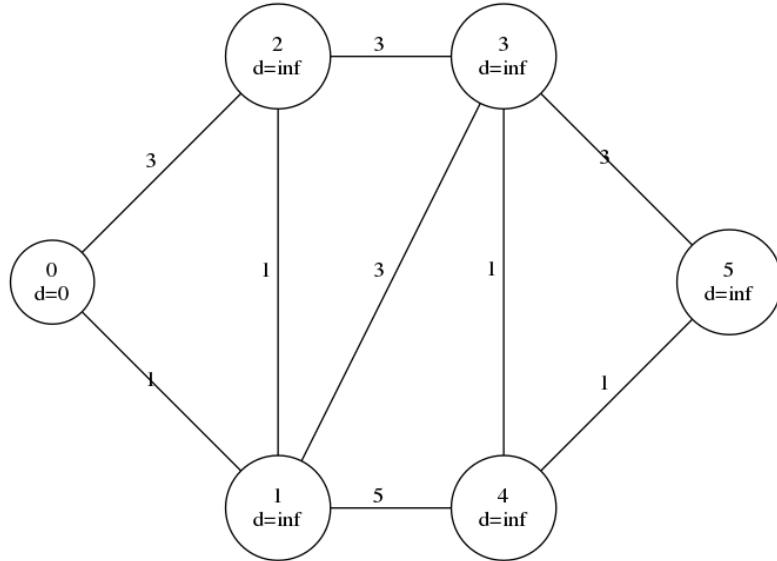


#### 5.2.4.2 Phase d'initialisation



Sommet de départ : 0

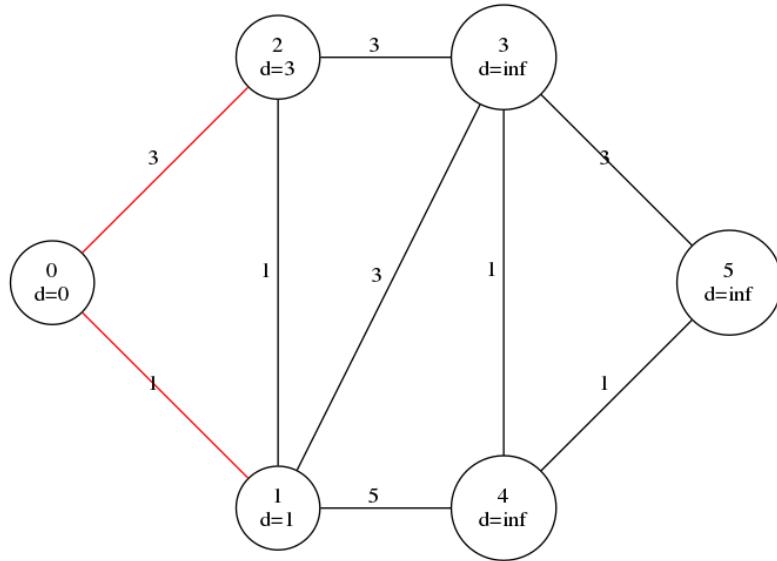
	P[0]	D[0]	P[1]	D[1]	P[2]	D[2]	P[3]	D[3]	P[4]	D[4]	P[5]	D[5]
Initialisation	n/a	0	n/a	$\infty$								



#### 5.2.4.3 Étape 1

- Sommet à distance minimum : 0 distance : 0
- Parcourons les sommets voisins de 0
  1. Le chemin vers le sommet 1 est amélioré en passant par le sommet 0 car  $\infty > 0 + 1$
  2. Le chemin vers le sommet 2 est amélioré en passant par le sommet 0 car  $\infty > 0 + 3$

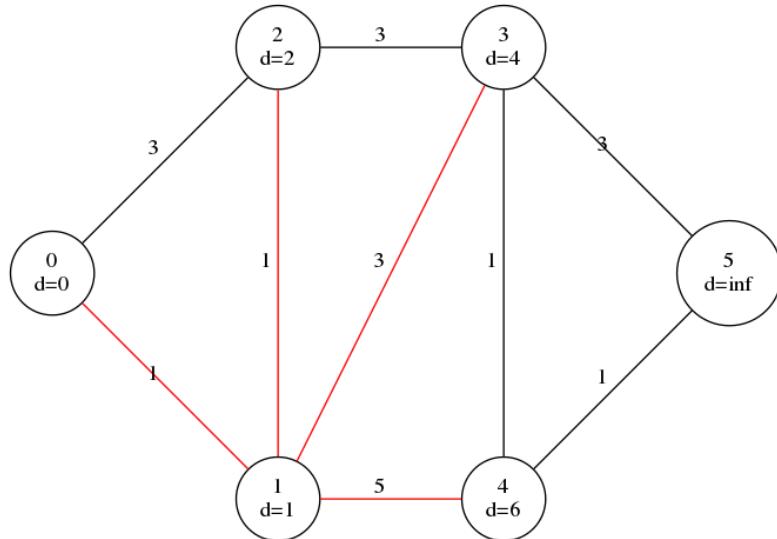
	P[0]	D[0]	P[1]	D[1]	P[2]	D[2]	P[3]	D[3]	P[4]	D[4]	P[5]	D[5]
Initialisation	n/a	0	n/a	$\infty$								
Étape 1	n/a	0	0	1	0	3	n/a	$\infty$	n/a	$\infty$	n/a	$\infty$



#### 5.2.4.4 Étape 2

- Sommet à distance minimum : 1 distance : 1
- Parcourons les sommets voisins de 1
  1. Le chemin vers le sommet 0 n'est pas amélioré en passant par le sommet 1
  2. Le chemin vers le sommet 2 est amélioré en passant par le sommet 1 car  $3 > 1 + 1$
  3. Le chemin vers le sommet 3 est amélioré en passant par le sommet 1 car  $\infty > 1 + 3$
  4. Le chemin vers le sommet 4 est amélioré en passant par le sommet 1 car  $\infty > 1 + 5$

	P[0]	D[0]	P[1]	D[1]	P[2]	D[2]	P[3]	D[3]	P[4]	D[4]	P[5]	D[5]
Initialisation	n/a	0	n/a	$\infty$								
Étape 1	n/a	0	0	1	0	3	n/a	$\infty$	n/a	$\infty$	n/a	$\infty$
Étape 2	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$



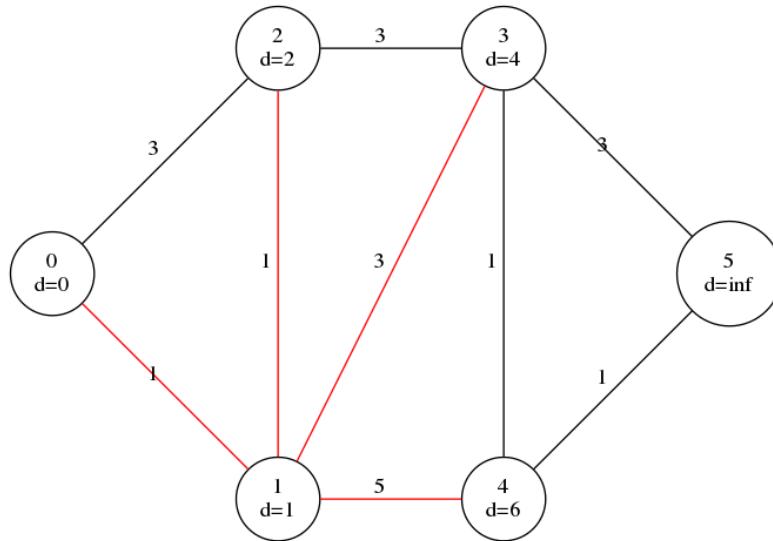
#### 5.2.4.5 Étape 3

- Sommet à distance minimum : 2 distance : 2

- Parcourons les sommets voisins de 2

- Le chemin vers le sommet 0 n'est pas amélioré en passant par le sommet 2
- Le chemin vers le sommet 1 n'est pas amélioré en passant par le sommet 2
- Le chemin vers le sommet 3 n'est pas amélioré en passant par le sommet 2

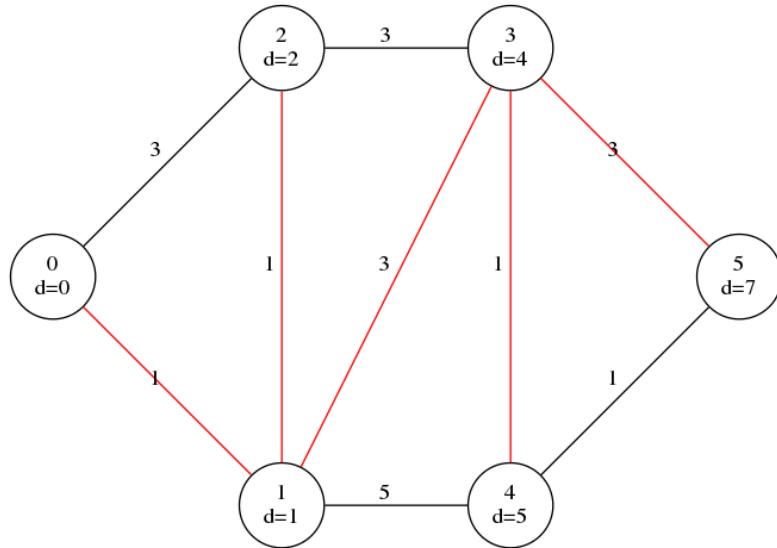
	P[0]	D[0]	P[1]	D[1]	P[2]	D[2]	P[3]	D[3]	P[4]	D[4]	P[5]	D[5]
Initialisation	n/a	0	n/a	$\infty$								
Étape 1	n/a	0	0	1	0	3	n/a	$\infty$	n/a	$\infty$	n/a	$\infty$
Étape 2	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$
Étape 3	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$



#### 5.2.4.6 Étape 4

- Sommet à distance minimum : 3 distance : 4
  - Parcourons les sommets voisins de 3
- Le chemin vers le sommet 1 n'est pas amélioré en passant par le sommet 3
  - Le chemin vers le sommet 2 n'est pas amélioré en passant par le sommet 3
  - Le chemin vers le sommet 4 est amélioré en passant par le sommet 3 car  $6 > 4 + 1$
  - Le chemin vers le sommet 5 est amélioré en passant par le sommet 3 car  $\infty > 4 + 3$

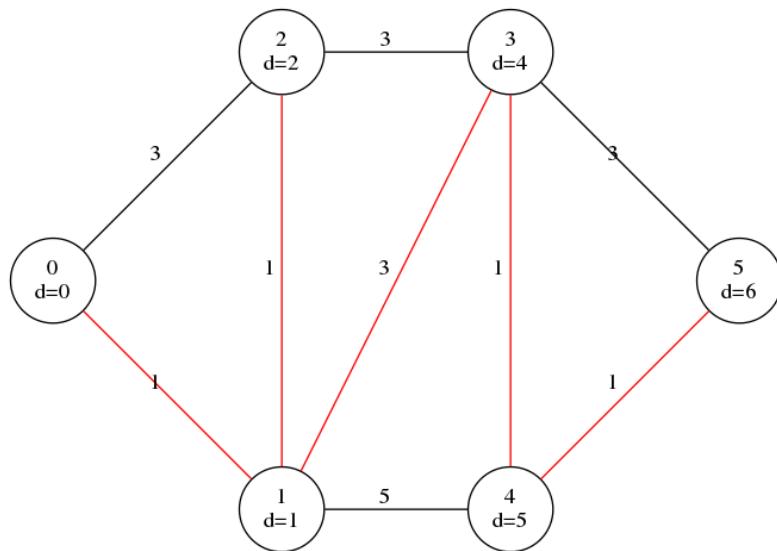
	P[0]	D[0]	P[1]	D[1]	P[2]	D[2]	P[3]	D[3]	P[4]	D[4]	P[5]	D[5]
Initialisation	n/a	0	n/a	$\infty$								
Étape 1	n/a	0	0	1	0	3	n/a	$\infty$	n/a	$\infty$	n/a	$\infty$
Étape 2	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$
Étape 3	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$
Étape 4	n/a	0	0	1	1	2	1	4	3	5	3	7



#### 5.2.4.7 Étape 5

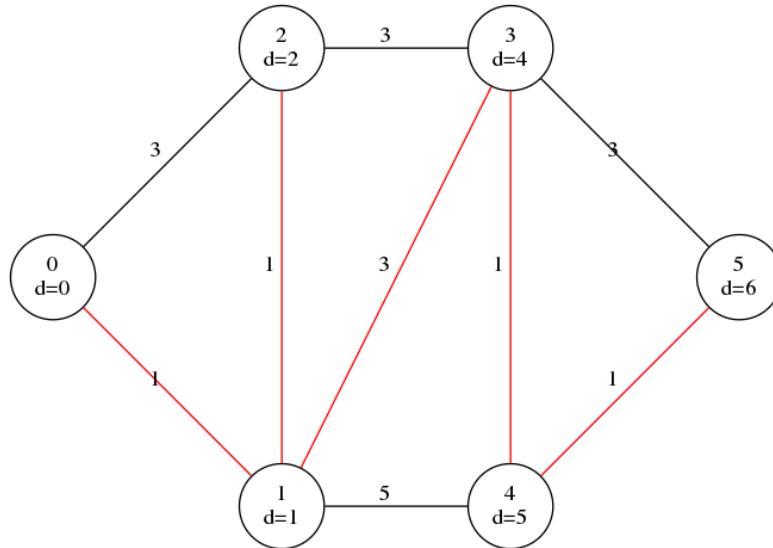
- Sommet à distance minimum : 4 distance : 5
- Parcourons les sommets voisins de 4
  1. Le chemin vers le sommet 1 n'est pas amélioré en passant par le sommet 4
  2. Le chemin vers le sommet 3 n'est pas amélioré en passant par le sommet 4
  3. Le chemin vers le sommet 5 est amélioré en passant par le sommet 4 car  $7 > 5 + 1$

	P[0]	D[0]	P[1]	D[1]	P[2]	D[2]	P[3]	D[3]	P[4]	D[4]	P[5]	D[5]
Initialisation	n/a	0	n/a	$\infty$								
Étape 1	n/a	0	0	1	0	3	n/a	$\infty$	n/a	$\infty$	n/a	$\infty$
Étape 2	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$
Étape 3	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$
Étape 4	n/a	0	0	1	1	2	1	4	3	5	3	7
Étape 5	n/a	0	0	1	1	2	1	4	3	5	4	6



#### 5.2.4.8 Étape 6

- Sommet à distance minimum : 5 distance : 6
- Parcourons les sommets voisins de 5
  1. Le chemin vers le sommet 3 n'est pas amélioré en passant par le sommet 5
  2. Le chemin vers le sommet 4 n'est pas amélioré en passant par le sommet 5



#### 5.2.5 Tableau résumé de l'algorithme

	P[0]	D[0]	P[1]	D[1]	P[2]	D[2]	P[3]	D[3]	P[4]	D[4]	P[5]	D[5]
Initialisation	n/a	0	n/a	$\infty$								
Étape 1	n/a	0	0	1	0	3	n/a	$\infty$	n/a	$\infty$	n/a	$\infty$
Étape 2	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$
Étape 3	n/a	0	0	1	1	2	1	4	1	6	n/a	$\infty$
Étape 4	n/a	0	0	1	1	2	1	4	3	5	3	7
Étape 5	n/a	0	0	1	1	2	1	4	3	5	4	6
Étape 6	n/a	0	0	1	1	2	1	4	3	5	4	6

##### 5.2.5.1 Le cas des graphes pondérés négativement

L'algorithme de Dijkstra va considérer les sommets du graphe dans un certain ordre (fonction Extraire-Min(F)). Au fur et à mesure de son exécution, il va ainsi propager la pondération la plus faible sur les sommets. Une fois qu'un sommet a été considéré, alors les choix le concernant deviennent irrévocables. C'est ce que l'on nomme une stratégie gloutonne. C'est justement ce principe qui rend l'algorithme inutilisable sur les graphes pondérés possédant un arc de poids négatif. En effet si on considère le graphe suivant :

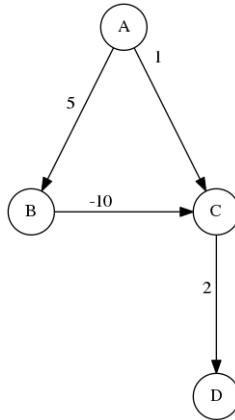


FIGURE 5.7 – Exemple de graphe possédant un arc négatif, sans cycle de poids négatif

Si on déroule l'algorithme de Dijkstra sur ce graphe, en partant du sommet  $A$ , on obtiendra le tableau suivant :

	P[A]	D[A]	P[B]	D[B]	P[C]	D[C]	P[D]	D[D]
Initialisation	n/a	0	n/a	$\infty$	n/a	$\infty$	n/a	$\infty$
Étape 1 (A)	n/a	0	A	5	A	1	n/a	$\infty$
Étape 2 (C)	n/a	0	A	5	A	1	C	3
Étape 3 (D)	n/a	0	A	5	A	1	C	3
Étape 4 (B)	n/a	0	A	5	B	-5	C	3

Si on considère le chemin allant du sommet  $A$  au sommet  $D$  découvert par l'algorithme de Dijkstra on obtient  $A \rightarrow B \rightarrow C \rightarrow D$  (ce qui est correct) pour une distance totale de 3 (ce qui est faux, puisque  $5 - 10 + 2 = -3$ ). L'algorithme est ainsi mis en défaut.

**Exercice.** Serez-vous capable de trouver un graphe pour lequel l'algorithme de Dijkstra propose un chemin faux (et pas uniquement sa distance associée) ?

### 5.2.6 Algorithme de Bellman-Ford

L'algorithme de Dijkstra, bien que rapide, possède un certain nombre d'inconvénients :

1. Ne fonctionne pas en présence d'arêtes de poids négatif.
2. Ne permet pas de détecter les cycles de poids négatif.

L'algorithme de Bellman-Ford permet, tout comme lui, de déterminer un plus court chemin depuis une source unique vers tous les autres sommets du graphe. Cet algorithme peut être vu comme une variation de l'algorithme de Dijkstra. Bien qu'étant plus lent, il fonctionne en

présence d'arcs de poids négatifs, et permet de détecter la présence d'un cycle de poids négatif.

---

**Algorithm 9:** Bellman-Ford( $G, s$ )

---

**Data:**  $G = (V, E)$  un graphe orienté pondéré et  $s \in V$  un sommet du graphe

**Résultat:** Un arbre des plus courts chemins d'origine  $s$

**foreach**  $u \in V$  **do**

$Distance[u] = \infty;$

$Pere[u] = NIL;$

**end**

$Distance[s] = 0;$

**for**  $|V|$  fois **do**

**foreach**  $(u, v) \in E$  **do**

**if**  $Distance[v] > Distance[u] + Poids(u, v)$  **then**

$Distance[v] = Distance[u] + Poids(u, v);$

$Pere[v] = u;$

**end**

**end**

**end**

**foreach**  $(u, v) \in E$  **do**

**if**  $Distance[v] > Distance[u] + Poids(u, v)$  **then**

**return** Faux;

**end**

**end**

**return** Vrai;

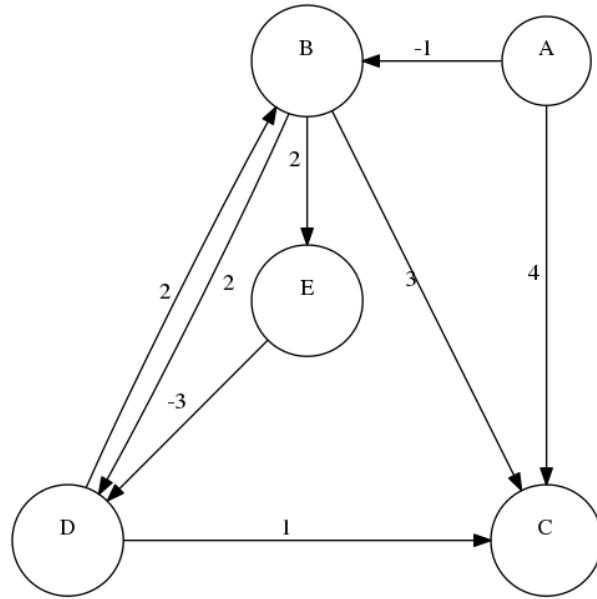
---

### 5.2.7 Explications

- Phase d'initialisation. Identique à celle de l'algorithme de Dijkstra.
- Boucle principale. Le nombre de répétitions permet de garantir que les plus courts chemins se seront correctement propagés.
  - Ne fonctionne pas de manière gloutonne.
  - On va propager des informations à chaque itération. Peut importe les sommets impliqués. On va réaliser suffisamment d'itérations pour que l'ensemble des informations soient prises en compte.
  - Pour chaque itération, on va considérer tous les arcs. Si un arc permet d'améliorer un chemin, alors on l'utilise.
- Dernière boucle. On refait une passe. Si une amélioration est encore possible, cela signifie que le graphe contient un cycle de poids négatif.

### 5.2.8 Déroulement de l'algorithme de Bellman-Ford

Dans cet exemple, nous allons considérer un graphe pondéré et orienté en partant du sommet A. Concernant son exécution, nous devons considérer à chaque itération la liste complète des arcs. Il n'est pas nécessaire que chaque itération les considère dans le même ordre. Cependant, pour plus de simplicité nous allons toujours considérer les arcs dans le même ordre. D'ailleurs, l'ordre dans lequel les arcs sont considérés peut avoir un impact sur le déroulement de l'algorithme.



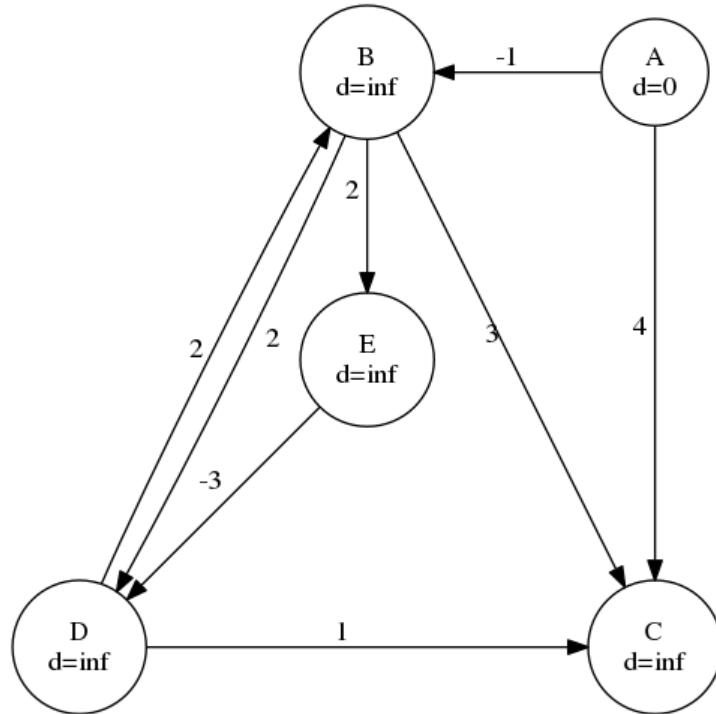
Nous allons considérer les arcs dans l'ordre suivant :

$$(D, C), (D, B), (E, D), (B, C), (B, E), (B, D), (A, B), (A, C)$$

#### 5.2.8.1 Initialisation

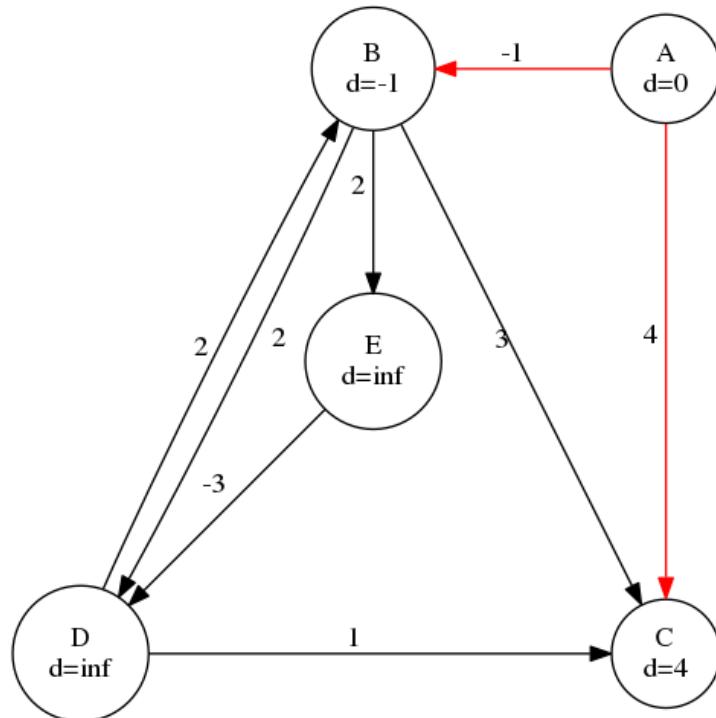
Nous partons du sommet A.

	P[A]	D[A]	P[B]	D[B]	P[C]	D[C]	P[D]	D[D]	P[E]	D[E]
	n/a	0	n/a	inf	n/a	inf	n/a	inf	n/a	inf



### 5.2.8.2 Étape 1

- Chemin vers le sommet C : pas d'amélioration par l'arc  $(D, C)$
- Chemin vers le sommet B : pas d'amélioration par l'arc  $(D, B)$
- Chemin vers le sommet D : pas d'amélioration par l'arc  $(E, D)$
- Chemin vers le sommet C : pas d'amélioration par l'arc  $(B, C)$
- Chemin vers le sommet E : pas d'amélioration par l'arc  $(B, E)$
- Chemin vers le sommet D : pas d'amélioration par l'arc  $(B, D)$
- Chemin vers le sommet B : amélioration en passant par le sommet A car  $inf > 0 + -1$   $(A, B)$
- Chemin vers le sommet C : amélioration en passant par le sommet A car  $inf > 0 + 4$   $(A, C)$

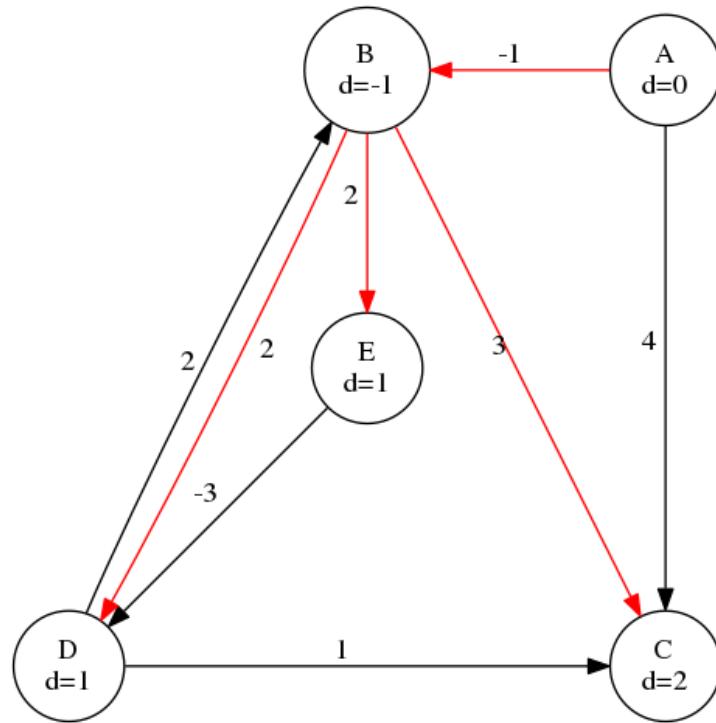


	P[A]	D[A]	P[B]	D[B]	P[C]	D[C]	P[D]	D[D]	P[E]	D[E]
	n/a	0	n/a	inf	n/a	inf	n/a	inf	n/a	inf
Etape 1	n/a	0	A	-1	A	4	n/a	inf	n/a	inf

### 5.2.8.3 Étape 2

- Chemin vers le sommet C : pas d'amélioration par l'arc  $(D, C)$
- Chemin vers le sommet B : pas d'amélioration par l'arc  $(D, B)$
- Chemin vers le sommet D : pas d'amélioration par l'arc  $(E, D)$
- Chemin vers le sommet C : amélioration en passant par le sommet B car  $4 > -1 + 3$   $(B, C)$
- Chemin vers le sommet E : amélioration en passant par le sommet B car  $inf > -1 + 2$   $(B, E)$
- Chemin vers le sommet D : amélioration en passant par le sommet B car  $inf > -1 + 2$   $(B, D)$

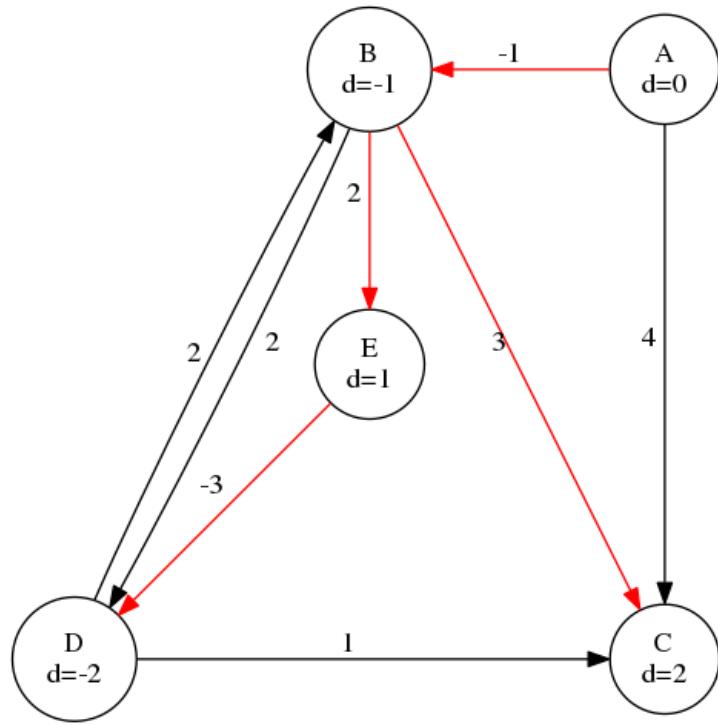
- Chemin vers le sommet B : pas d'amélioration par l'arc  $(A, B)$
- Chemin vers le sommet C : pas d'amélioration par l'arc  $(A, C)$



	P[A]	D[A]	P[B]	D[B]	P[C]	D[C]	P[D]	D[D]	P[E]	D[E]
	n/a	0	n/a	inf	n/a	inf	n/a	inf	n/a	inf
Etape 1	n/a	0	A	-1	A	4	n/a	inf	n/a	inf
Etape 2	n/a	0	A	-1	B	2	B	1	B	1

#### 5.2.8.4 Étape 3

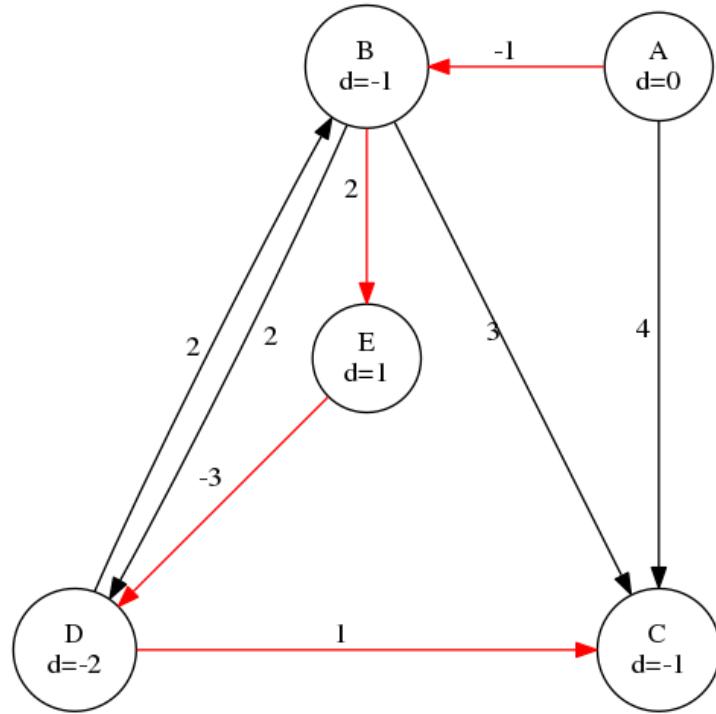
- Chemin vers le sommet C : pas d'amélioration par l'arc  $(D, C)$
- Chemin vers le sommet B : pas d'amélioration par l'arc  $(D, B)$
- Chemin vers le sommet D : amélioration en passant par le sommet E car  $1 > 1 + -3$   $(E, D)$
- Chemin vers le sommet C : pas d'amélioration par l'arc  $(B, C)$
- Chemin vers le sommet E : pas d'amélioration par l'arc  $(B, E)$
- Chemin vers le sommet D : pas d'amélioration par l'arc  $(B, D)$
- Chemin vers le sommet B : pas d'amélioration par l'arc  $(A, B)$
- Chemin vers le sommet C : pas d'amélioration par l'arc  $(A, C)$



	P[A]	D[A]	P[B]	D[B]	P[C]	D[C]	P[D]	D[D]	P[E]	D[E]
	n/a	0	n/a	inf	n/a	inf	n/a	inf	n/a	inf
Etape 1	n/a	0	A	-1	A	4	n/a	inf	n/a	inf
Etape 2	n/a	0	A	-1	B	2	B	1	B	1
Etape 3	n/a	0	A	-1	B	2	E	-2	B	1

#### 5.2.8.5 Étape 4

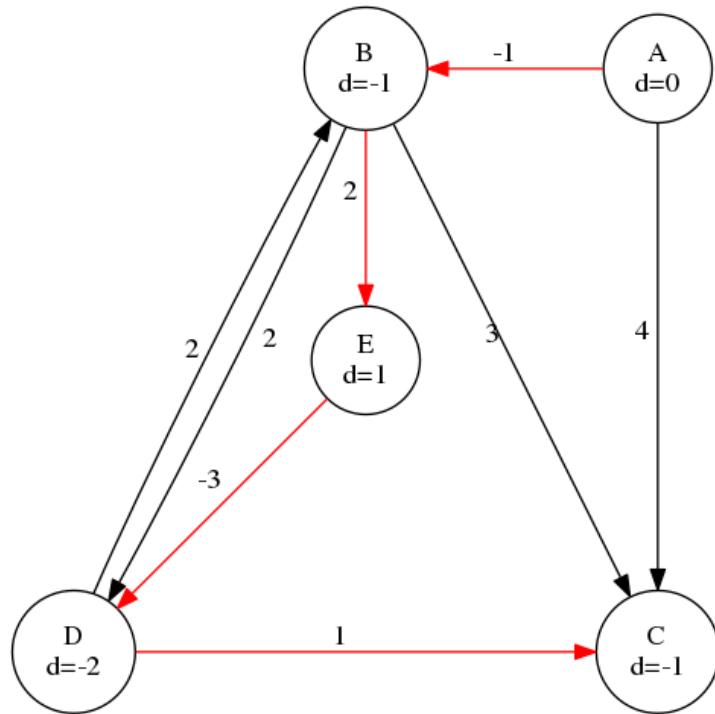
- Chemin vers le sommet C : amélioration en passant par le sommet D car  $2 > -2 + 1$  ( $D, C$ )
- Chemin vers le sommet B : pas d'amélioration par l'arc ( $D, B$ )
- Chemin vers le sommet D : pas d'amélioration par l'arc ( $E, D$ )
- Chemin vers le sommet C : pas d'amélioration par l'arc ( $B, C$ )
- Chemin vers le sommet E : pas d'amélioration par l'arc ( $B, E$ )
- Chemin vers le sommet D : pas d'amélioration par l'arc ( $B, D$ )
- Chemin vers le sommet B : pas d'amélioration par l'arc ( $A, B$ )
- Chemin vers le sommet C : pas d'amélioration par l'arc ( $A, C$ )



	P[A]	D[A]	P[B]	D[B]	P[C]	D[C]	P[D]	D[D]	P[E]	D[E]
	n/a	0	n/a	inf	n/a	inf	n/a	inf	n/a	inf
Etape 1	n/a	0	A	-1	A	4	n/a	inf	n/a	inf
Etape 2	n/a	0	A	-1	B	2	B	1	B	1
Etape 3	n/a	0	A	-1	B	2	E	-2	B	1
Etape 4	n/a	0	A	-1	D	-1	E	-2	B	1

#### 5.2.8.6 Étape 5

- Chemin vers le sommet C : pas d'amélioration par l'arc  $(D, C)$
- Chemin vers le sommet B : pas d'amélioration par l'arc  $(D, B)$
- Chemin vers le sommet D : pas d'amélioration par l'arc  $(E, D)$
- Chemin vers le sommet C : pas d'amélioration par l'arc  $(B, C)$
- Chemin vers le sommet E : pas d'amélioration par l'arc  $(B, E)$
- Chemin vers le sommet D : pas d'amélioration par l'arc  $(B, D)$
- Chemin vers le sommet B : pas d'amélioration par l'arc  $(A, B)$
- Chemin vers le sommet C : pas d'amélioration par l'arc  $(A, C)$



	P[A]	D[A]	P[B]	D[B]	P[C]	D[C]	P[D]	D[D]	P[E]	D[E]
	n/a	0	n/a	inf	n/a	inf	n/a	inf	n/a	inf
Etape 1	n/a	0	A	-1	A	4	n/a	inf	n/a	inf
Etape 2	n/a	0	A	-1	B	2	B	1	B	1
Etape 3	n/a	0	A	-1	B	2	E	-2	B	1
Etape 4	n/a	0	A	-1	D	-1	E	-2	B	1
Etape 5	n/a	0	A	-1	D	-1	E	-2	B	1

### 5.3 Comparaison des deux algorithmes

Ces deux algorithmes, Dijkstra et Bellman-Ford, nous permettent d'illustrer les choix que doivent réaliser les informaticiens. En effet, confronté à une situation modelisée par un graphe, quel algorithme sera le plus adapté ? Cela dépend de plusieurs critères que nous allons étudier :

#### 5.3.1 Le temps d'exécution.

Soit  $n$  le nombre de sommets du graphe et  $m$  le nombre d'arcs du graphe. Avec une certaine implémentation, il est possible d'obtenir la complexité (ordre de grandeur du temps d'exécution) suivante pour chacun des deux algorithmes :

- $O((m + n)\log(n))$  pour l'algorithme de Dijkstra.
- $O(nm)$  pour l'algorithme de Bellman-Ford.

Concrètement, si le temps d'exécution est crucial pour l'application, alors il convient d'utiliser l'algorithme de Dijkstra.

#### 5.3.2 Graphes admissibles.

Cependant, le simple critère du temps d'exécution n'est pertinent que si le graphe est admissible pour les deux algorithmes :

- L'algorithme de Dijkstra sélectionne de manière gloutonne le nœud de poids minimum qui n'a pas encore été traité, et effectue ce processus de relaxation sur tous ses arcs sortants.
- L'algorithme de Bellman-Ford effectue la relaxation sur tous les arcs, et fait cela  $n$  fois. A chacune de ces itérations, le nombre de sommets avec des distances correctement calculées augmente. Il en résulte que tous les sommets auront finalement leurs distances correctes.

L'algorithme de Bellman-Ford peut donc s'appliquer à une classe d'entrées plus large que l'algorithme de Dijkstra.

### 5.3.3 Conclusion.

- Si le graphe contient au moins une arête de poids négatif, il est nécessaire d'utiliser l'algorithme de Bellman-Ford.
- Si ce n'est pas le cas, il est préférable d'utiliser l'algorithme de Dijkstra.

## Chapitre 6

# Problème de transports et flots

De nombreux problèmes peuvent être modélisés par un problème de flot, c'est-à-dire calculer la quantité maximum d'un fluide pouvant transiter sur un réseau dont les capacités sont limitées. Par exemple :

1. Distribution d'eau dans un réseau de canalisations, transport de pétrole sur un réseau de pipelines.
2. Systèmes de transports en commun dans une grande ville.
3. Transport d'énergie électrique sur le réseau EDF.
4. Débit de données sur le réseau d'un opérateur (neutralité du net).

Nous allons voir comment modéliser de tels problèmes, comment les résoudre au moyen de l'algorithme de Ford-Fulkerson, puis un exemple illustrant le lien entre théorie des flots et un problème classique de théorie des graphes.

### 6.1 Réseaux de transport et flots

Un réseau de transport peut-être vu comme un graphe orienté, chaque arc étant associé à une quantité maximum de flot pouvant le traverser à un instant donné :

**Définition 34 (Réseau de transport)** *Un réseau de transport est un graphe  $G = (V, A, c, s, t)$  tel que chaque arc  $(u, v)$  possède une capacité  $c(u, v)$ . Un réseau de transport possède également deux sommets particuliers notés  $s$  (la source) et  $t$  (le puit) tels que  $N^+(s) = 0$  et  $N^-(t) = 0$ .*

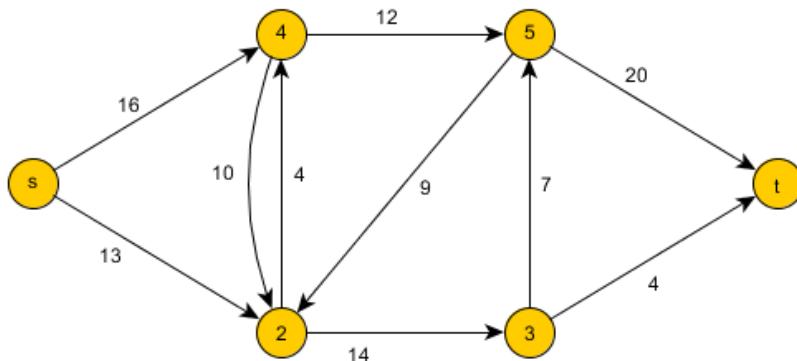


FIGURE 6.1 – Exemple de réseau de transport

Pour plus de simplicité, on suppose que :

1. Le graphe est connexe
2. Tout sommet du graphe se trouve sur un chemin reliant  $s$  à  $t$ .

**Définition 35 (Flot)** Un *flot* est une fonction  $f$ , appliquée à un réseau de transport  $G = (V, A, c, s, t)$ , qui associe à chaque arc  $(u, v)$  une valeur  $f(u, v)$  qui respecte les trois propriétés suivantes :

1. *Contrainte de capacité* : Pour tout arc  $(u, v)$ ,  $f(u, v) \leq c(u, v)$ .
2. *Symétrie* : Pour tout couple de sommets  $(u, v)$ ,  $f(u, v) = -f(v, u)$ .
3. *Conservation du flot* : Pour tout sommet autre que  $s$  ou  $t$ , la somme des flots entrants est égale à la somme des flots sortants.

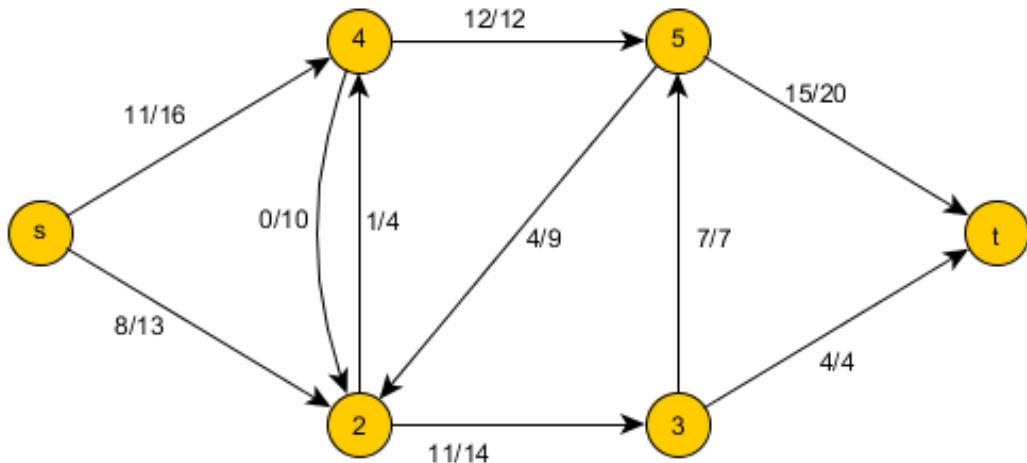
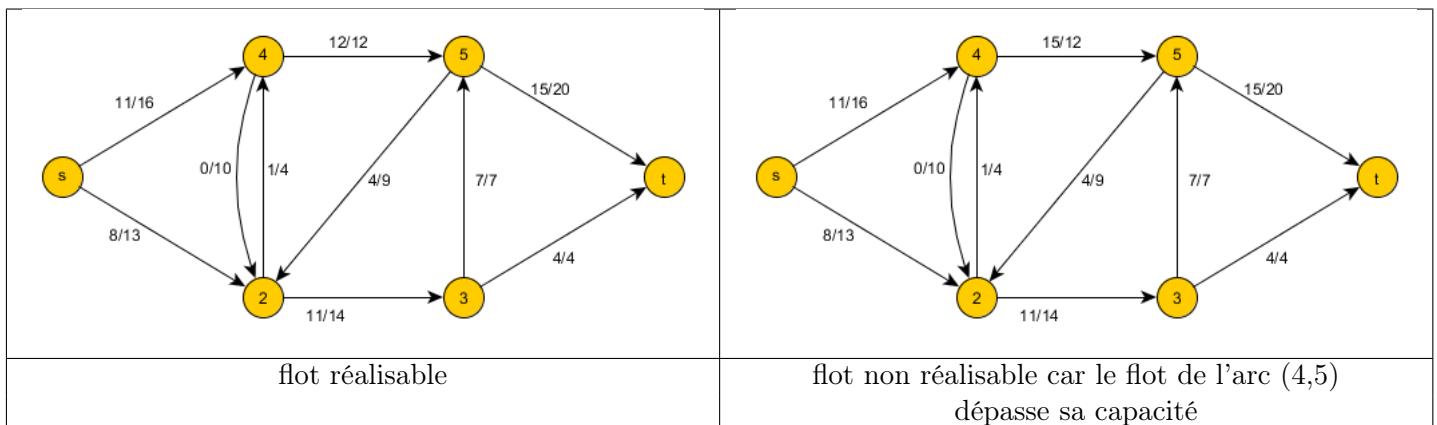


FIGURE 6.2 – Exemple de flot sur le réseau de transport de la figure 6.1

**Définition 36 (Flot réalisable)** Si pour tout arc du réseau de transport, la valeur du flot ne dépasse pas la capacité, alors on dit que le flot est réalisable.



**Remarque :** Quand deux arcs en sens inverse relient deux sommets, on peut toujours annuler la fonction flot sur l'un des deux arcs.

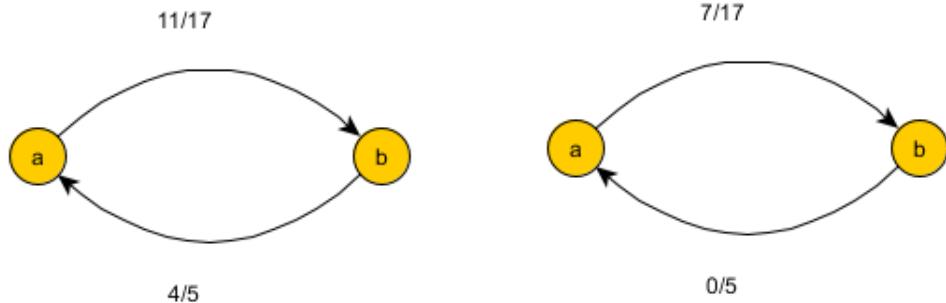


FIGURE 6.3 – Exemple de deux arcs en sens inverse, l'un avec un flot de 11, l'autre avec un flot de 4.

FIGURE 6.4 – Les deux mêmes arcs après annulation du flot sur l'un d'entre eux.

**Définition 37 (Saturation)** *Lorsqu'un flot passant par un arc  $(u, v)$  atteint la capacité, on dit alors que cet arc est saturé.*

**Définition 38 (Valeur du flot)** *La valeur d'un flot, est égale à la somme des flots sortants du sommet  $s$ , ou bien, de manière équivalente, à la somme des flots entrants au sommet  $t$ .*

En particulier, le flot présent sur la figure 6.2 a une valeur de 19.

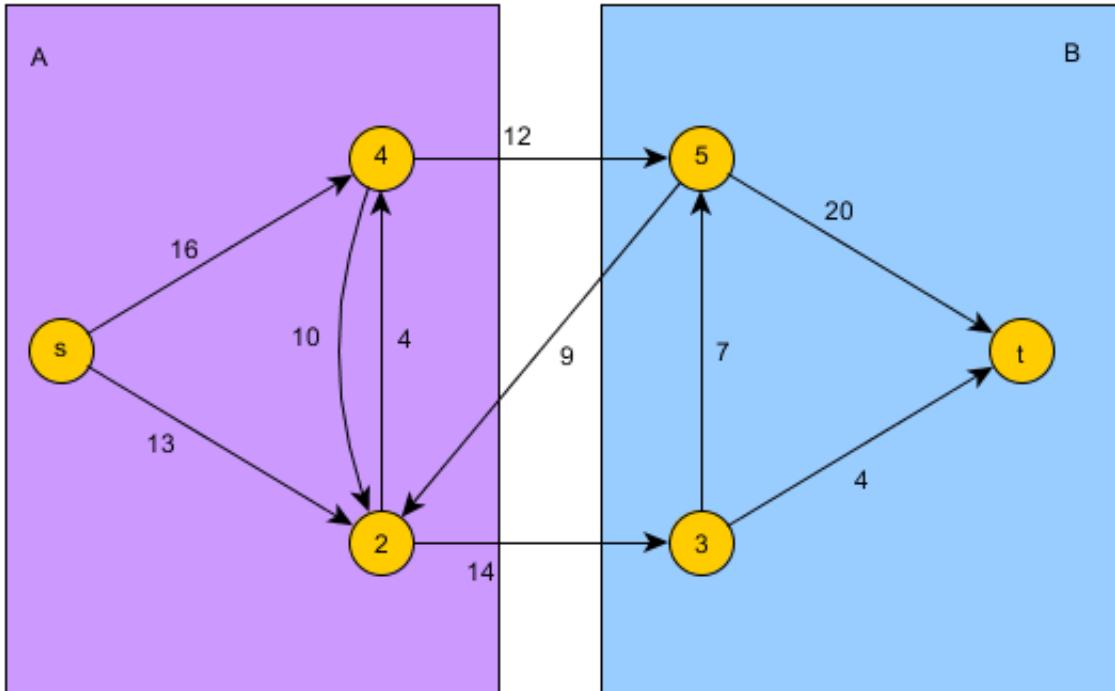
## 6.2 Coupe minimum

**Définition 39 (Coupe)** *une coupe est une séparation des sommets en deux sous-ensembles  $A$  et  $B$ , tels que  $s \in A$ ,  $t \in B$  et  $B = V - A$ .*

**Définition 40 (Capacité d'une coupe)** *La capacité d'une coupe  $(A, B)$ , notée  $C(A, B)$ , est égale à la somme des capacités des arcs allant de  $A$  vers  $B$  :*

$$C(A, B) = \sum_{i \in A, j \in B} c(i, j)$$

**Exemple.** La capacité de la coupe  $(A, B)$  dans l'exemple suivant vaut  $12 + 14 = 26$ .



**Définition 41 (Flot net)** Étant donnée une coupe séparant les sommets en deux ensembles  $A$  et  $B$ . La somme des valeurs du flot sur les arcs allant de  $A$  vers  $B$  moins la somme des valeurs du flot sur les arcs allant de  $B$  vers  $A$  est appelée flot net traversant la coupe. Le flot net traversant une coupe ne dépend pas de la coupe.

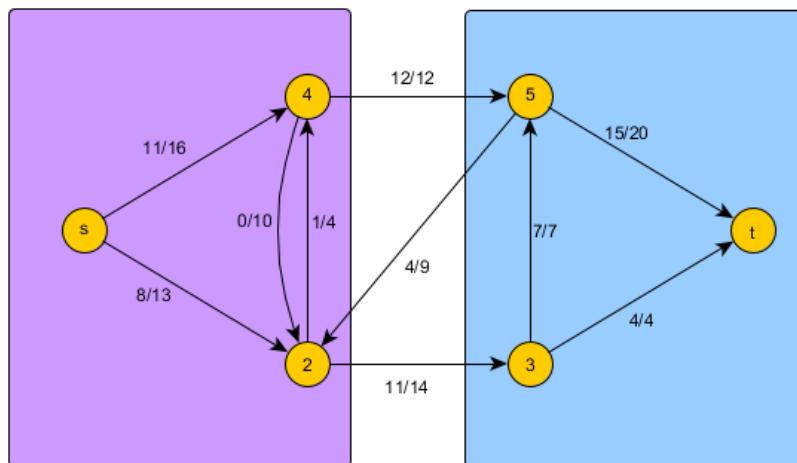


FIGURE 6.5 – Exemple de coupe pour laquelle le flot net vaut  $12 - 4 + 11 = 19$ .

**Théorème 2 (Ford-Fulkerson)** Pour tout flot  $F$  de valeur  $V(F)$ , et pour toute coupe  $(A, B)$  de capacité  $C(A, B)$  on a :

$$V(F) \leq C(A, B)$$

**Définition 42 (Coupe minimale)** Une coupe  $(A, B)$  est dite minimale pour un flot  $F$  si tous les arcs allant de  $A$  vers  $B$  sont saturées et aucun arc de  $B$  vers  $A$  n'est saturé.

**Théorème 3** Si il existe une coupe minimale pour un flot  $F$  alors ce flot est maximal.

## 6.3 Flot maximum

Il est trivial de trouver un flot réalisable, puisqu'il suffit d'appliquer un flot nul à chaque arc. Le problème qui nous intéresse ici est de trouver un flot dont la valeur est maximum.

### 6.3.1 Graphe résiduel

Étant donnés un réseau de transport et un flot associé, le graphe résiduel correspond au graphe représentant la quantité de flot pouvant être ajouté ou retiré sur chaque arc. On peut le construire de la manière suivante :

1. On utilise le même ensemble de sommets du graphe.
2. Pour chaque arc  $f(u, v) \leq c(u, v)$ , on peut augmenter le flot de  $c(u, v) - f(u, v)$  et on peut le diminuer de  $f(u, v)$ , ce qui revient à faire passer un flot  $-f(u, v)$  sur un arc  $(v, u)$ .

### 6.3.2 Chemin augmentant

**Définition 43 (Chemin augmentant)** Pour un réseau de transport associé à un flot, un chemin augmentant est un chemin allant de  $s$  à  $t$  dans le graphe résiduel.

Un chemin augmentant  $p$  déterminé d'un graphe résiduel  $G_r$  calculé à partir d'un réseau de transport  $G$  associé à un flot  $f$  peut être vu comme la possibilité d'augmentation du flot  $f$  en modifiant la valeur du flot sur les arcs de  $p$  dans  $f$ . Afin de conserver un flot réalisable, ces arcs ne peuvent être augmentés que de la valeur minimum des capacités résiduelle de  $p$ .

### 6.3.3 Algorithme de Ford-Fulkerson

---

**Algorithm 10:** Ford-Fulkerson( $G = (V, A, c, s, t)$ )

---

**Data:**  $G = (V, A, c, s, t)$  un réseau de transport

**Réultat:** Un flot maximum pour ce réseau

**foreach**  $(u, v) \in A$  **do**

|  $f(u, v) = 0;$

**end**

**while**  $\exists$  un chemin  $p$  de  $s$  à  $t$  dans le graphe résiduel **do**

|  $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\};$

| **foreach**  $(u, v) \in p$  **do**

| |  $f(u, v) = f(u, v) + c_f(p);$

| **end**

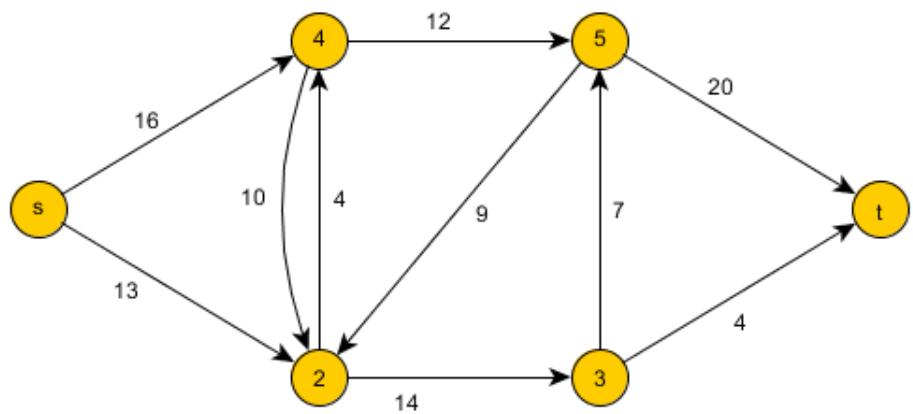
**end**

**return** Vrai;

---

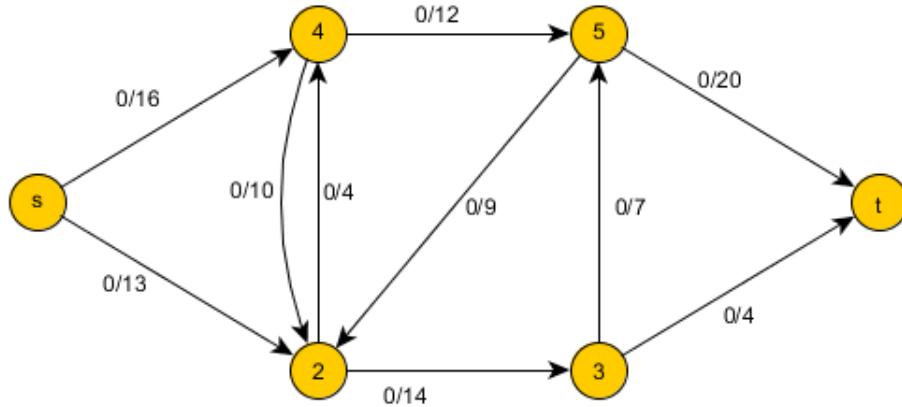
### 6.3.4 Déroulement de l'algorithme

Nous allons dérouler l'algorithme de Ford-Fulkerson sur le réseau de transport suivant :



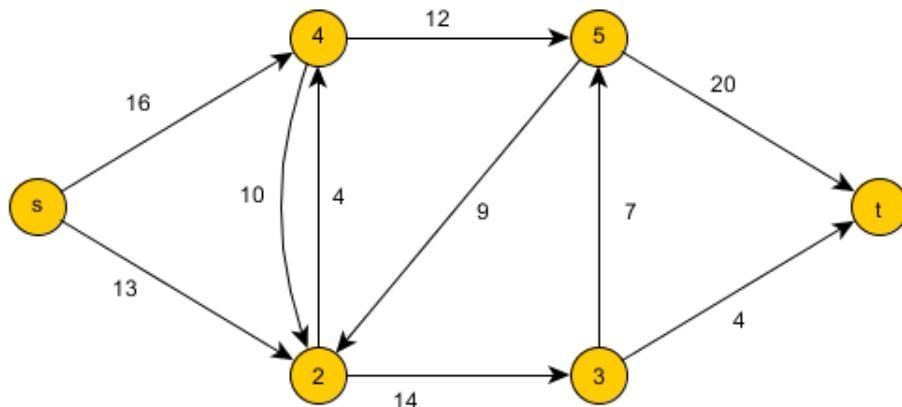
#### 6.3.4.1 Étape d'initialisation

**Initialisation avec un flot nul.** Pour débuter, on initialise avec un flot nul, qui est nécessairement un flot réalisable.



#### 6.3.4.2 Étape 1.

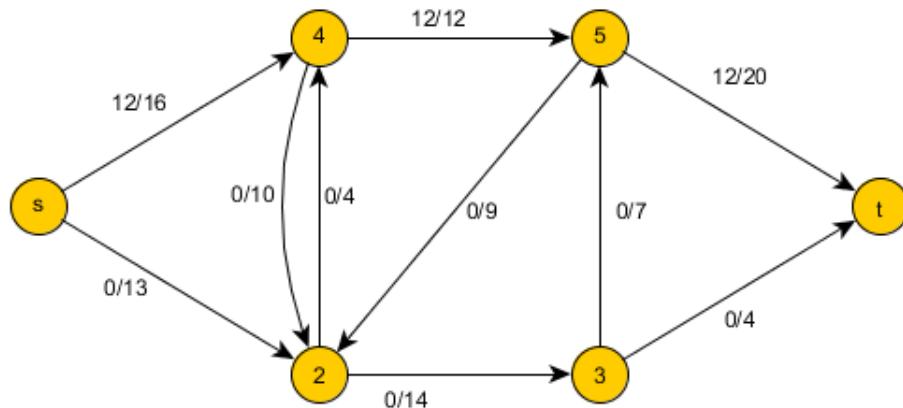
**Calcul du graphe résiduel.** A partir de ce flot réalisable, nous calculons son graphe résiduel.



**Sélection d'un chemin allant de  $s$  à  $t$  dans le graphe résiduel.** Prenons le chemins  $s \rightarrow 4 \rightarrow 5 \rightarrow t$ . L'arc de poids minimum sur ce chemin vaut 12. Nous allons donc augmenter de 12 le flot des arcs  $(s, 4), (4, 5), (5, t)$ .

**Important.** Le chemin est sélectionné de manière totalement arbitraire. Si on souhaite implémenter cet algorithme, il est possible d'utiliser n'importe quel algorithme, par exemple celui du parcours en largeur ou celui de Dijkstra.

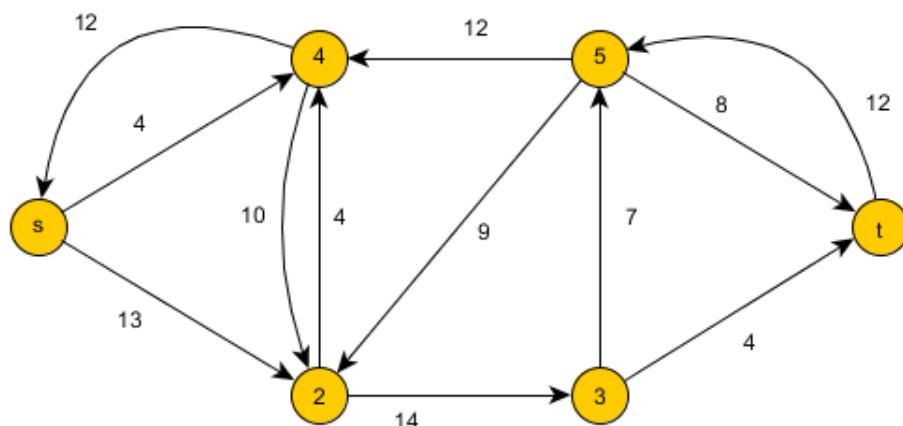
On obtient le flot réalisable suivant :



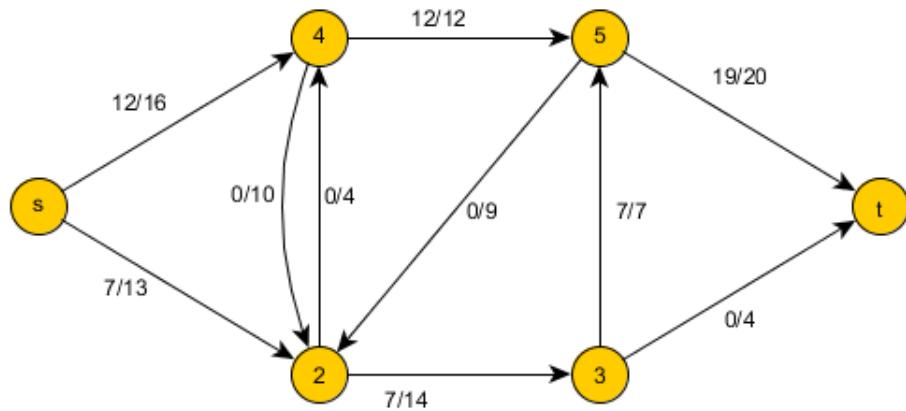
On peut remarquer directement qu'on ne pourra donc plus augmenter le flot parcourant l'arc  $(4, 5)$  puisqu'il est saturé.

#### 6.3.4.3 Étape 2.

##### Calcul du graphe résiduel

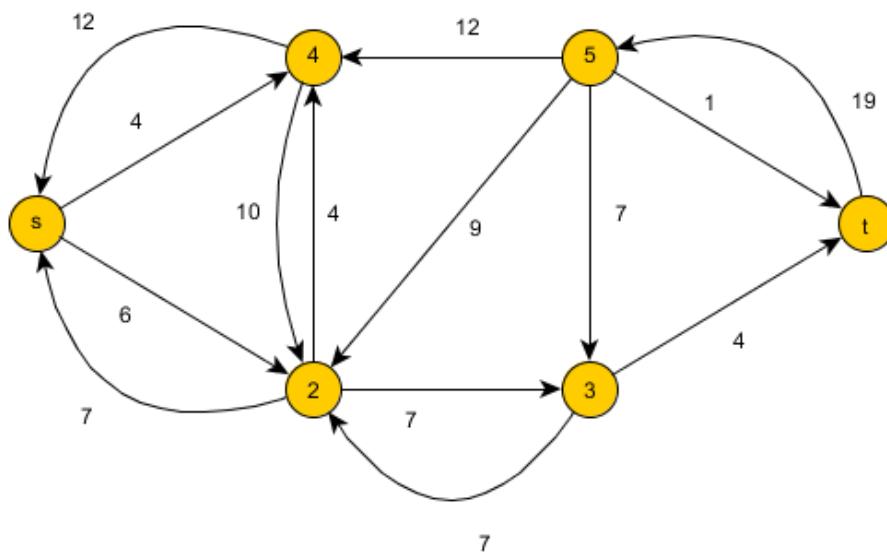


**Sélection d'un chemin allant de  $s$  à  $t$  dans le graphe résiduel.** Prenons le chemins  $s \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow t$ . L'arc de poids minimum vaut 7. Nous allons donc augmenter de 7 le flot des arcs  $(s, 2), (2, 3), (3, 5), (5, t)$ .

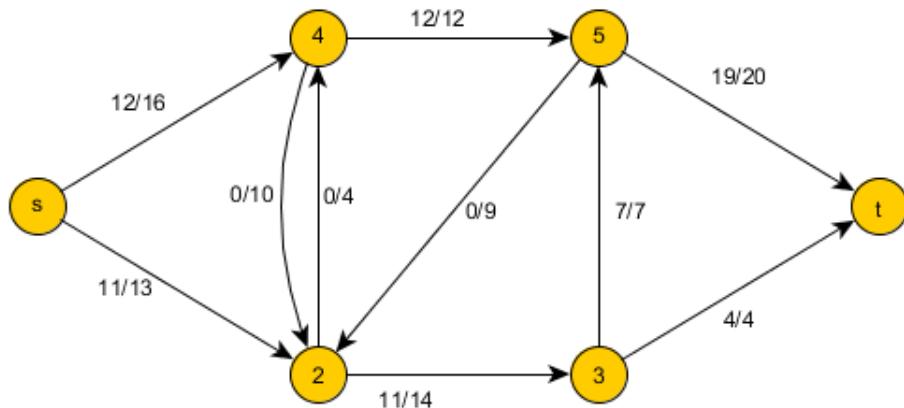


#### 6.3.4.4 Étape 3.

##### Calcul du graphe résiduel

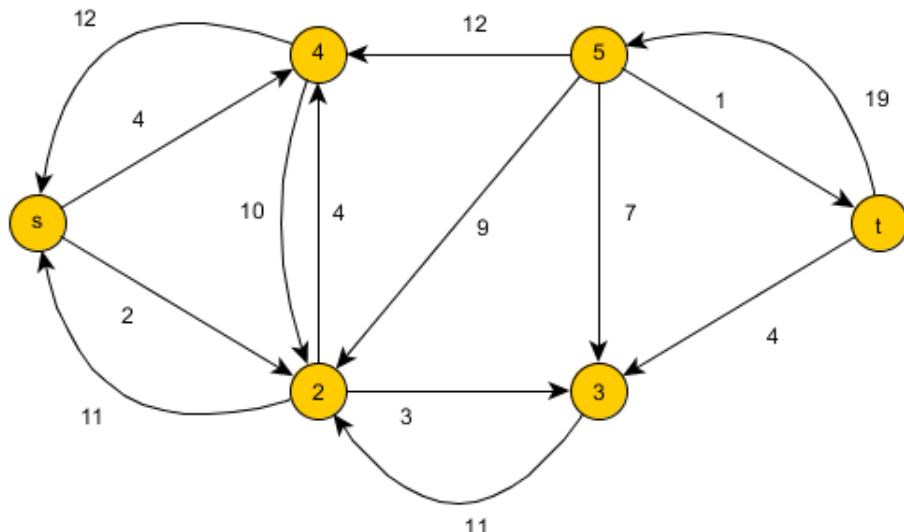


**Sélection d'un chemin allant de  $s$  à  $t$  dans le graphe résiduel** Prenons le chemins  $s \rightarrow 2 \rightarrow 3 \rightarrow t$ . L'arc de poids minimum vaut 4. Nous allons donc augmenter de 4 le flot des arcs  $(s, 2), (2, 3), (3, t)$ .



#### 6.3.4.5 Étape 4.

##### Calcul du graphe résiduel



**Sélection d'un chemin allant de  $s$  à  $t$  dans le graphe résiduel** Il n'existe pas de chemin permettant de relier le sommet  $s$  au sommet  $t$  dans le graphe résiduel.

**Fin de l'algorithme.** Le flot maximum a pour valeur 23.

## 6.4 Exemples d'applications des flots

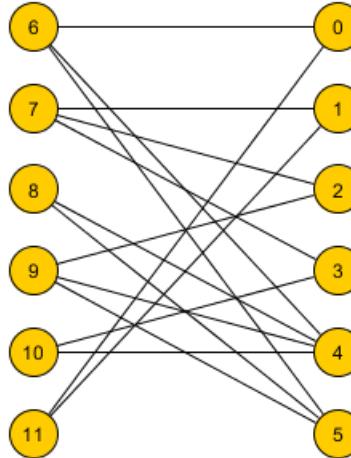
### 6.4.1 Déterminer des chemins indépendants

Lorsque la capacité de chaque arc du réseau est de 1, alors un flot correspond à un ensemble de chemins n'ayant aucun arc en commun. La capacité de ce flot est égale au nombre de ces chemins.

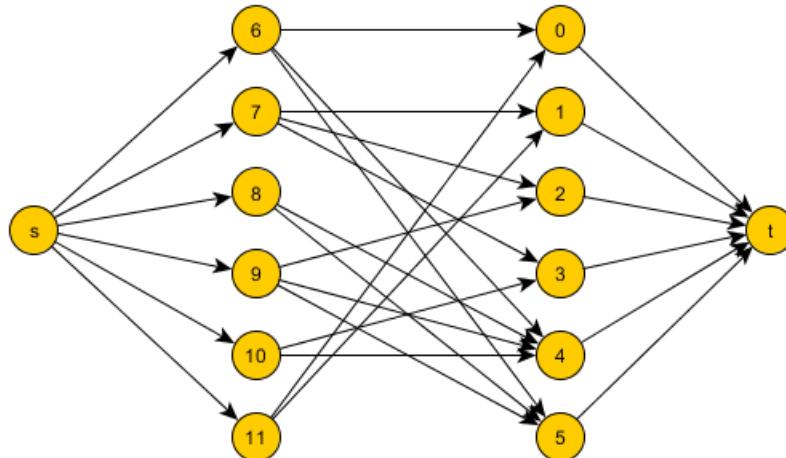
### 6.4.2 Couplage maximum dans un graphe biparti

**Définition 44 (Couplage)** Un couplage d'un graphe est un ensemble d'arêtes deux à deux non adjacentes. Un couplage est dit maximal (au sens de l'inclusion) lorsqu'on ne peut plus ajouter d'arête dans le couplage. Un couplage est dit maximum lorsqu'il n'existe pas d'autre couplage contenant plus d'arêtes.

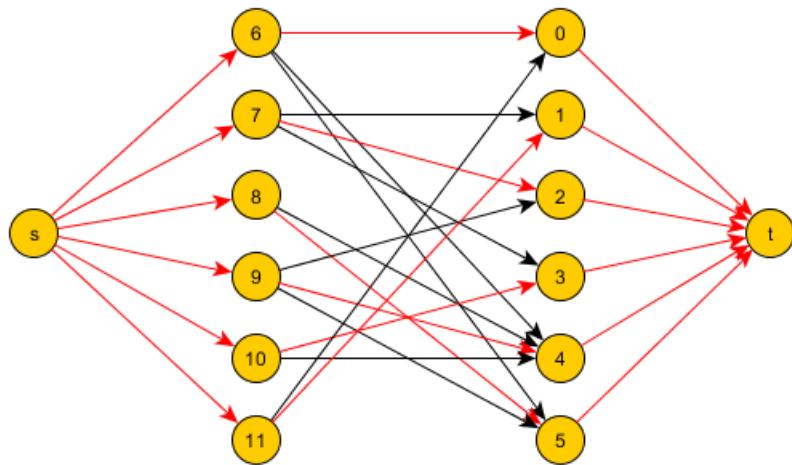
Trouver un couplage maximum dans un graphe biparti revient à un problème de flot maximum. Considérons le graphe suivant :



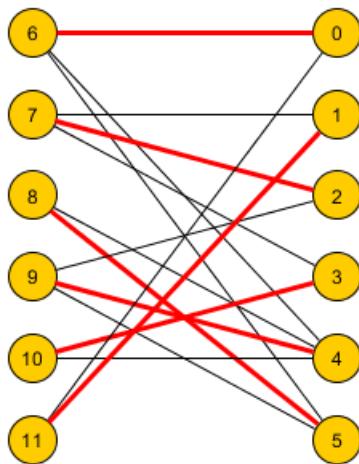
Il suffit d'ajouter une source reliée aux sommets du premier ensemble, et un puits relié aux sommets du second ensemble. Ensuite, chaque arête du graphe est transformée en arc allant de la source vers le puits.



En considérant que la capacité de chaque arc est de 1, on calcule ensuite un flot maximum.



Chaque arc saturé entre les deux ensembles appartient ainsi au couplage, qui est maximum.



# Chapitre 7

## Programmation linéaire

Dans cette partie, nous allons voir un autre outil de la recherche opérationnelle : la programmation linéaire. Cet outil permet à la fois de modéliser un grand nombre de problèmes, mais aussi de les résoudre. En particulier, il existe de nombreux solveurs très performants qui rendent la programmation linéaire utilisable en pratique pour des problèmes de grande ou petite taille.

### 7.1 Exemple introductif

Afin de présenter cet outil, nous allons considérer un exemple concret.

#### 7.1.1 Un problème de production

Un fabricant produit 2 types de yaourts à la fraise A et B à partir de Fraise, de Lait et de Sucre. Chaque pot de yaourt doit respecter les proportions suivantes de matières premières :

	A	B
Fraise	2 kg	1 kg
Lait	1 kg	2 kg
Sucre	0 kg	1 kg

On dispose de 800 Kg de Fraises, 700 Kg de Lait et 300 Kg de sucre. La vente de 1 pot de yaourts *A* et *B* rapporte respectivement 4 euros et 5 euros. Le fabricant cherche à maximiser son profit.

#### 7.1.2 Modélisation

Pour modéliser un tel problème, il convient de se poser un certain nombre de questions :

- Sur quelles quantités peut-on travailler ?
- Que cherche-t-on à optimiser ?
- Quelles sont les contraintes du problème ?

##### **7.1.2.1 Sur quelles quantités peut-on travailler ?**

1. Seules valeurs non constantes : les quantités de yaourts A et B produites
2. On parle de variables
3. On les notera  $X_A$  et  $X_B$

### 7.1.2.2 Que cherche-t-on à optimiser ?

1. Le profit  $Z$
2. Calculé à partir de  $X_A$  et  $X_B$
3. On parle de fonction objectif
4.  $Z = 4X_A + 5X_B$

### 7.1.2.3 Quelles sont les contraintes du problème ?

Optimiser une fonction objectif est une chose, a priori, simple. Cependant, il ne faut pas négliger les contraintes induites par les quantités limitées de ressources à notre disposition.

1. Première contrainte : 800 Kg de fraises disponibles
2. La quantité utilisée dépend de la production :  $2X_A + X_B$
3. D'où la contrainte :  $2X_A + X_B \leq 800$

En suivant le même raisonnement, on obtient les trois contraintes suivantes :

1.  $2X_A + X_B \leq 800$  (fraises)
2.  $X_A + 2X_B \leq 700$  (lait)
3.  $X_B \leq 300$  (sucre)

De plus, le nombre de pots de yaourts A et B est forcément positif. D'où les deux contraintes supplémentaires :

1.  $X_A \geq 0$
2.  $X_B \geq 0$

### 7.1.2.4 Le programme linéaire

Ainsi, avec les différentes informations, on obtient le programme linéaire suivant :

$$\begin{array}{rcl} \text{Max} & 4X_A & + \quad 5X_B \\ & 2X_A & + \quad X_B \quad \leq \quad 800 \\ & X_A & + \quad 2X_B \quad \leq \quad 700 \\ & & \quad X_B \quad \leq \quad 300 \\ & X_A, X_B & \geq \quad 0 \end{array}$$

## 7.1.3 Évolution du modèle

Ce type de modèle peut évoluer très facilement :

1. Si on rajoute un produit alors il suffit de rajouter une variable.
2. Si on ajoute une ressource critique il suffit d'ajouter une contrainte.

## 7.2 Modélisation par un programme linéaire

Un programme linéaire possède trois composantes : Les variables, une fonction économique (aussi appelée fonction objectif) et un ensemble de contraintes.

### 7.2.1 Les variables

Les variables correspondent aux quantités variables (des produits fabriqués, par exemple). Il s'agit des éléments sur lesquels nous allons avoir un impact en ayant la possibilité de modifier leurs valeurs.

### 7.2.2 La fonction économique

La fonction objectif correspond à une combinaison linéaire des différentes variables. C'est cette fonction que l'on souhaite optimiser, en la maximisant ou en la minimisant.

### 7.2.3 Les contraintes

L'optimisation de la fonction objectif se fait en tenant compte des différentes contraintes liées aux variables. Chaque contrainte peut être représentée par une équation ou une inéquation dont les coefficients ne doivent pas contenir de variables du problème.

### 7.2.4 Écriture générale d'un programme linéaire

On peut écrire ainsi un programme linéaire avec  $n$  variables  $X_1, \dots, X_n$  et  $m$  contraintes.

$$\text{Min ou Max } \sum_{i=1}^n C_i X_i$$

sous les contraintes

$$\sum_{i=1}^n A_{ij} X_i \leq B_j, (j = 1 \dots m)$$

$$X_i \in \mathbb{R}, (i = 1 \dots n)$$

1. **Linéarité** : Objectif et contraintes sont des fonctions linéaires des variables de décision (les coefficients  $C_i$  et  $A_{ij}$  des variables sont constants)
2. **Continuité** : Les variables peuvent prendre n'importe quelle valeur réelle respectant les contraintes linéaires

La modélisation par un Programme Linéaire est une méthode assez générique, elle permet de capturer de nombreux problèmes concrets. Un programme linéaire peut être résolu par des algorithmes dédiés efficaces en théorie et/ou en pratique. Ces algorithmes sont plus efficaces que des algorithmes génériques de Programmation mathématique.

**Exemple :** algorithme du simplexe, méthode des points intérieurs, etc.

Il existe un certain nombre de solveurs rapides, basés sur ces algorithmes et permettant une intégration aisée dans tout type d'applications (Java, C++, C, python ...)

**Exemple d'outils :** CPLEX, LPSolve, Xpress, Excel ...

## 7.3 Linéarité et non-linéarité

Notez bien que toutes les fonctions, équations et inéquations que nous considérons doivent être linéaires. Considérons la formule suivante :

$$A \cdot X + B \cdot Y = C$$

La linéarité implique que les coefficients  $A$ ,  $B$  et  $C$  ne peuvent contenir les variables  $X$  ou  $Y$ .

### 7.3.1 Exemples de programmes linéaires

Le programme obtenu dans notre exemple introductif est, bien sur, linéaire :

$$\begin{array}{lllll} \text{Max} & 4X_A & + & 5X_B & \\ & 2X_A & + & X_B & \leq 800 \\ & X_A & + & 2X_B & \leq 700 \\ & & & X_B & \leq 300 \\ & X_A, X_B & \geq & 0 & \end{array}$$

### 7.3.2 Exemples de programmes non linéaires

**Premier exemple.** Dans cet exemple, la fonction objectif n'est pas linéaire puisqu'elle contient le coefficient suivant :  $4X_A \cdot X_B$ .

$$\begin{array}{lllll} \text{Max} & 4X_A \cdot X_B & + & 5X_B & \\ & 2X_A & + & X_B & \leq 800 \\ & X_A & + & 2X_B & \leq 700 \\ & & & X_B & \leq 300 \\ & X_A, X_B & \geq & 0 & \end{array}$$

**Deuxième exemple.** Dans cet exemple, l'une des contraintes n'est pas linéaire puisqu'elle contient le coefficient suivant :  $X_A \cdot X_A$ .

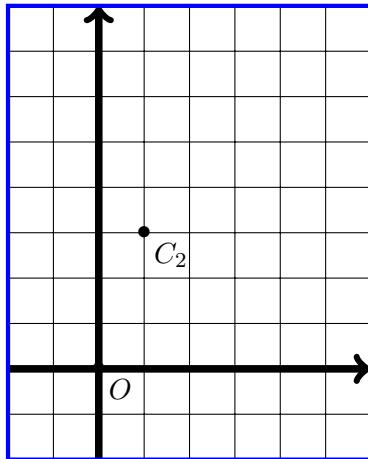
$$\begin{array}{lllll} \text{Max} & 4X_A & + & 5X_B & \\ & 2X_A & + & X_B & \leq 800 \\ & X_A \cdot X_A & + & 2X_B & \leq 700 \\ & & & X_B & \leq 300 \\ & X_A, X_B & \geq & 0 & \end{array}$$

## 7.4 Résolution graphique

Un programme linéaire à 2 variables peut être résolu graphiquement. Reprenons notre exemple introductif :

$$\begin{array}{lllll} \text{Max} & 4X_A & + & 5X_B & \text{(Fonction objectif)} \\ & 2X_A & + & X_B & \leq 800 \quad (\text{C1}) \\ & X_A & + & 2X_B & \leq 700 \quad (\text{C2}) \\ & & & X_B & \leq 300 \quad (\text{C3}) \\ & X_A, X_B & \geq & 0 & \text{(C4) et (C5)} \end{array}$$

**Repère orthonormé.** Dans un repère orthonormé chaque point possède une abscisse et une ordonnée. En considérant que la valeur de la première variable représentera l'abscisse d'un point et la valeur de la seconde représentera son ordonnée, il devient possible de représenter l'espace des solutions admissibles. Chaque point du plan correspondant une solution, éventuellement non réalisable.



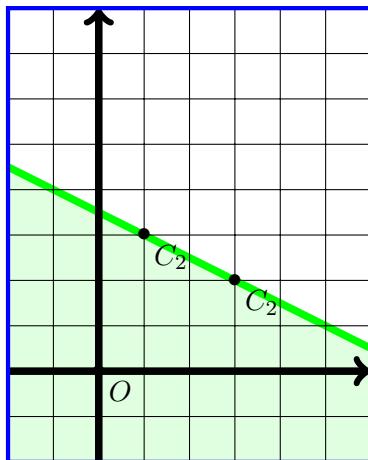
La fonction objectif que nous souhaitons maximiser vaut :

$$4X_A + 5X_B$$

Le point C<sub>2</sub> a pour coordonnées (1, 3). Son abscisse va correspondre à la variable X<sub>A</sub> tandis que son ordonnée va correspondre à la variable X<sub>B</sub>. Ainsi, le point C<sub>2</sub> correspond à la solution

$$4 \cdot 1 + 5 \cdot 3 = 19$$

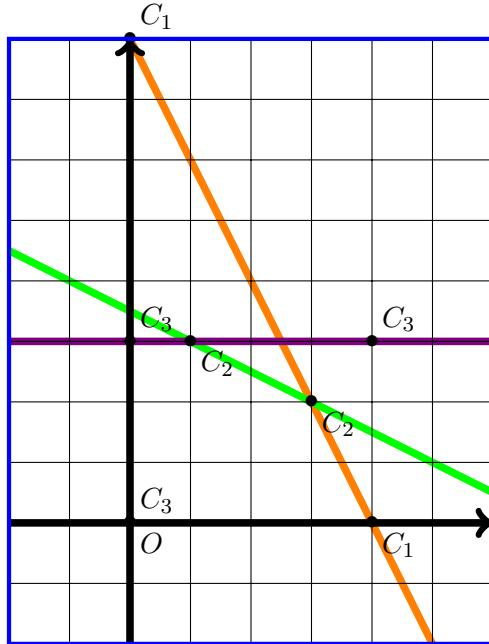
**Demi plan.** Chaque contrainte étant de la forme  $A \cdot X + B \cdot Y \leq C$ , on peut en déduire un demi-plan contenant tous les points respectant cette contrainte. La frontière de ce demi-plan étant déterminée par l'équation de droite suivante :  $A \cdot X + B \cdot Y = C$ . Ainsi, chaque contrainte (C<sub>1</sub>, C<sub>2</sub>, C<sub>3</sub>, C<sub>4</sub> et C<sub>5</sub>) peut être vu comme un demi-plan limitant l'espace des solutions admissibles.



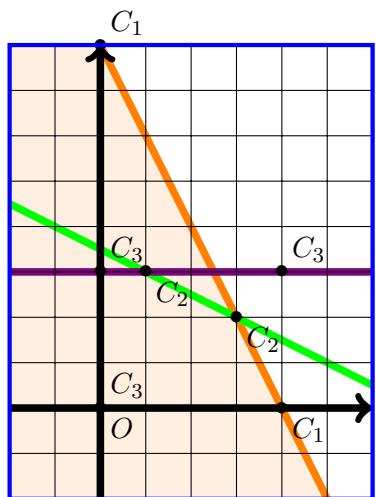
Prenons par exemple la contrainte C<sub>2</sub> :  $X_A + 2X_B \leq 700$ . Chaque unité de la grille vaut 100. On considère la frontière de l'inéquation, c'est à dire la droite  $X_A + 2X_B = 700$ . Il suffit de calculer deux points de la droite :

1. Posons  $X_A = 100$ . On en déduit que  $X_B$  vaut  $X_B = \frac{700 - X_A}{2} = 300$ .
2. Posons  $X_A = 300$ . On en déduit que  $X_B$  vaut  $X_B = \frac{700 - X_A}{2} = 200$ .

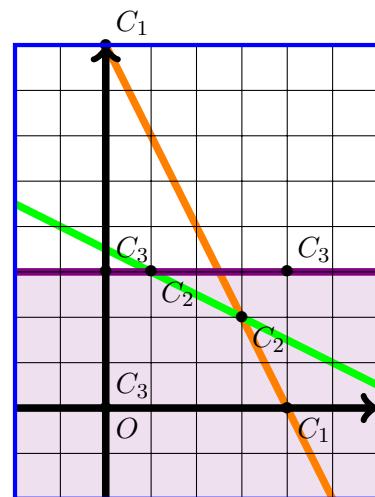
**Frontières des contraintes.** Chacune des différentes contraintes peut être transformée en une équation. Cette équation représente la frontière de la zone admissible induite par la contrainte.



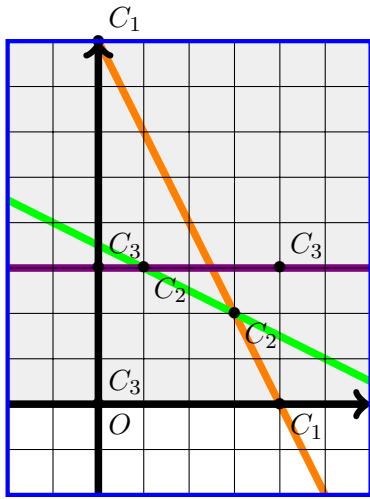
**Demi-plan (suite).** Pour chacune des contraintes, voici la zone d'admissibilité induite.



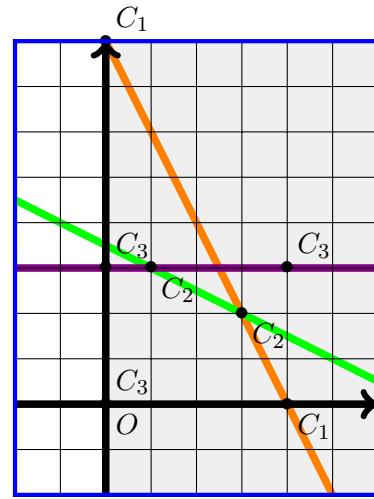
Zone admissible induite par  
la contrainte :  $C_1$ .



Zone admissible induite par  
la contrainte :  $C_3$ .

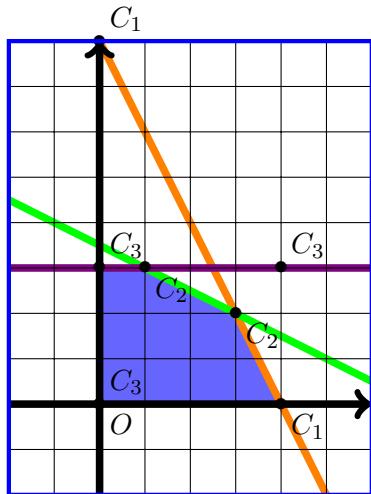


Zone admissible induite par la contrainte :  $X_B \geq 0$ .



Zone admissible induite par la contrainte :  $X_A \geq 0$ .

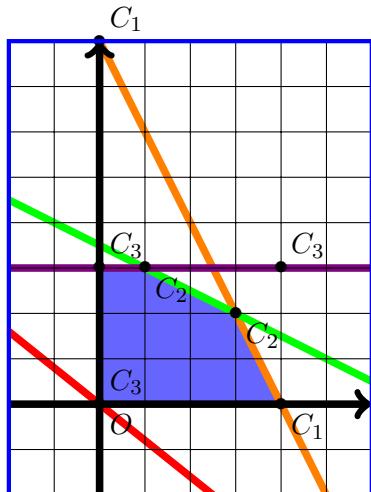
**Polyèdre des solutions.** L'intersection de ces demi-plans va former un polyèdre contenant l'ensemble des points qui respectent toutes les contraintes. Il ne reste plus qu'à déterminer lequel de ces point permet d'optimiser notre fonction objectif.



Zone admissible induite par l'ensemble des contraintes. Cette zone est aussi appelée polyèdre des solutions.

Chaque point situé à l'intérieur ou sur les frontières correspond à une solution admissible.

**Fonction objectif.** Pour cela, il suffit de remarquer que la fonction objectif, étant une combinaison linéaire des deux variables, peut être vu comme une équation de droite. En déterminant une valeur triviale (généralement en mettant la valeur de la fonction objectif à 0), il est possible de tracer cette droite.

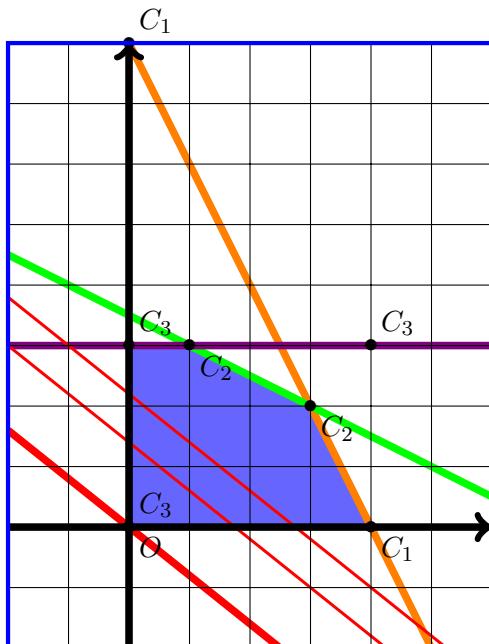


Droite représentant la fonction objectif initialisée à une valeur triviale :

$$4X_A + 5X_B = 0$$

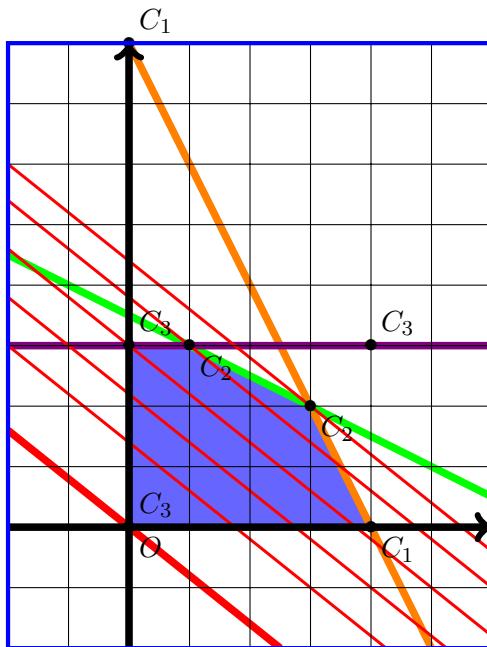
1.  $(X_A = 0, X_B = 0)$
2.  $(X_A = -200, X_B = 160)$

Changer la valeur de la fonction objectif n'aura aucun impact sur les coefficients des variables. Ainsi, toutes les droites générées pour différentes valeurs de la fonction objectif seront parallèles.



- Augmentation de la valeur de la fonction objectif :  $4X_A + 5X_B = 700$ 
  1.  $(X_A = -200, X_B = 300)$
  2.  $(X_A = 425, X_B = -200)$
- Augmentation de la valeur de la fonction objectif :  $4X_A + 5X_B = 1100$ 
  1.  $(X_A = -200, X_B = 380)$
  2.  $(X_A = 525, X_B = -200)$

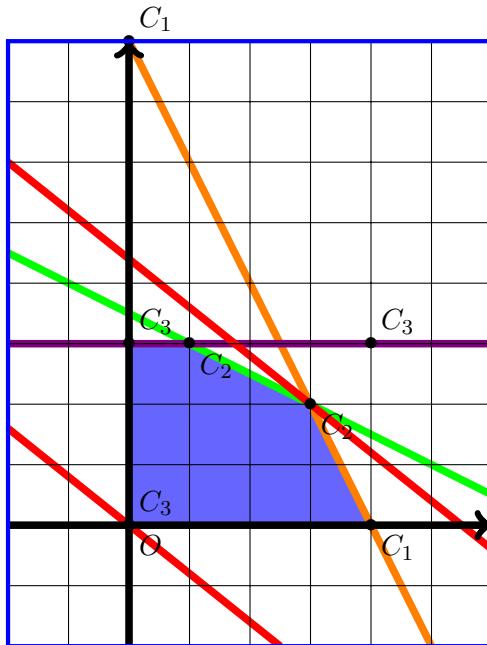
C'est ici que l'idée de résolution graphique intervient. On va dessiner plusieurs fois la droite correspondant à la fonction objectif pour des valeurs l'optimisant de plus en plus, tout en la faisant traverser (au moins par un point) l'espace des solutions réalisables. Enfin, n'importe quel point situé sur la dernière droite ET appartenant à l'espace des solutions réalisables correspond à une solution optimale.



On augmente successivement la valeur de la fonction objectif jusqu'à tracer la droite représentant la fonction objectif et ayant la valeur 2200 :  $4X_A + 5X_B = 2200$

1.  $(X_A = -200, X_B = 600)$
2.  $(X_A = 600, X_B = -40)$

Il s'agit de la valeur maximum pouvant être atteinte par la fonction objectif tout en restant en intersection avec au moins un sommet du polyèdre des solutions.



Droite représentant la fonction objectif :  $4X_A + 5X_B = 2200$

1.  $(X_A = -200, X_B = 600)$
2.  $(X_A = 600, X_B = -40)$

La solution optimale est atteinte au point  $(300, 200)$ .

### Pour résumer :

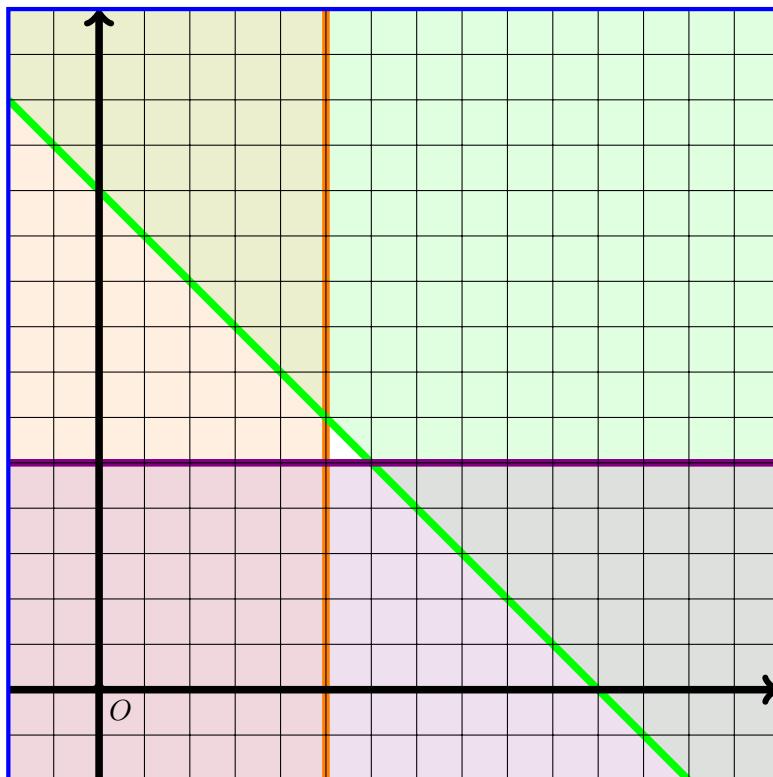
1. Chaque axe représente une des deux variables.
2. Chaque contrainte représente un demi-plan.
3. Si il existe une solution, l'ensemble des contraintes détermine une zone, un polyèdre, représentant l'espace des solutions possibles du problème, c'est-à-dire le domaine réalisable.
4. La fonction objectif, valant initialement 0 et dont la valeur va changer peut être représentée par une succession de droites parallèles.
5. La droite parallèle représentant la valeur la plus importante (dans le cas d'un problème de maximisation) correspond à une solution optimale.

### 7.4.1 problème infaisable

Voici un exemple de programme linéaire ne possédant pas de solution :

$$\begin{array}{lll}
 \text{Max} & X_1 + X_2 & \text{(Fonction objectif)} \\
 & X_2 \leq 1 & \text{(C1)} \\
 & X_1 \leq 1 & \text{(C2)} \\
 & X_1 + X_2 \geq 2.2 & \text{(C3)} \\
 & X_1, X_2 \geq 0 & \text{(C4) et (C5)}
 \end{array}$$

Si vous observez attentivement le graphique, vous pourrez constater que les trois demi-plans ne s'intersectent pas en même temps.

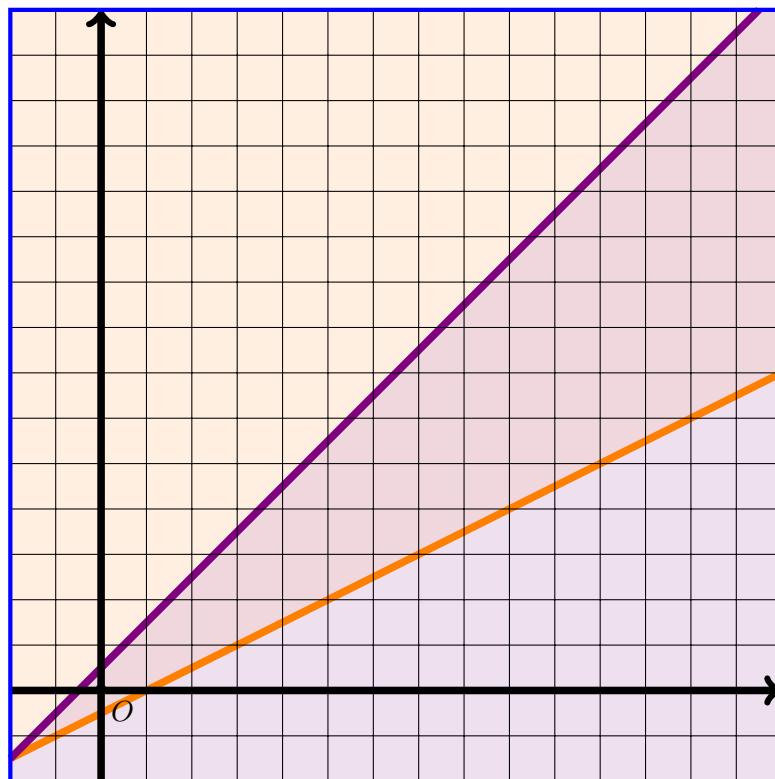


Échelle du graphique : 1 carreau = 0.2.

### 7.4.2 Solution non-bornée

Il existe des problèmes non bornés. Cela signifie qu'il n'est pas possible de les optimiser, puisque pour toute solution admissible, il en existe une autre dont la valeur est plus grande (ou plus petite dans le cas d'un problème de minimisation).

$$\begin{array}{lll}
 \text{Max} & X_1 + X_2 & \text{(Fonction objectif)} \\
 & -2 \cdot X_1 + 2 \cdot X_2 \leq 1 & \text{(C1)} \\
 & X_1 - 2 \cdot X_2 \geq 1 & \text{(C2)} \\
 & X_1, X_2 \geq 0 & \text{(C4) et (C5)}
 \end{array}$$



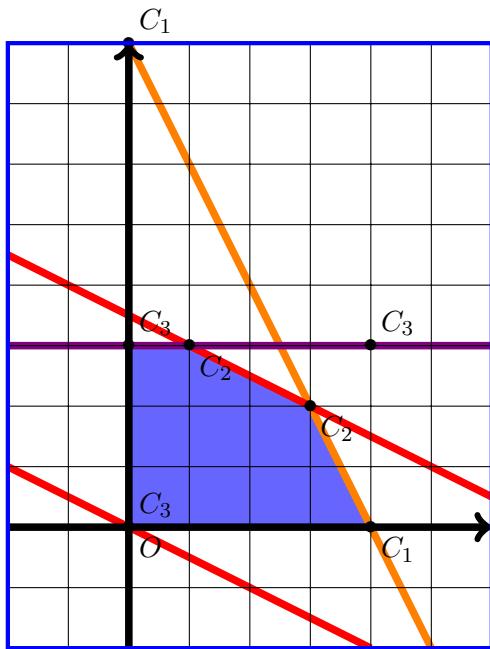
Échelle du graphique : 1 carreau = 1.

### 7.4.3 Solutions multiples

Certains problèmes possèdent une infinité de solutions optimales. Par exemple, lorsque la fonction objectif n'intersecte pas le polyèdre en un point mais sur un segment. Considérons le programme linéaire suivant (la fonction objectif a changé par rapport à notre exemple introduc-tif) :

$$\begin{array}{lll}
 \text{Max} & 2X_A + 4X_B & \text{(Fonction objectif)} \\
 & 2X_A + X_B \leq 800 & \text{(C1)} \\
 & X_A + 2X_B \leq 700 & \text{(C2)} \\
 & X_B \leq 300 & \text{(C3)} \\
 X_A, X_B & \geq 0 & \text{(C4) et (C5)}
 \end{array}$$

Et la résolution graphique associée :



La droite représentant notre fonction objectif, une fois optimisée, se superpose avec la droite représentant la frontière de la contrainte  $C_2$ . Tous les points, présents sur le segment situé entre les deux points  $C_2$ , à savoir  $(100, 300)$  et  $(300, 200)$ , correspondent à une solution optimale. Il y a donc une infinité de solutions.

#### 7.4.4 Sommets et solutions.

Comme nous venons de le voir, si le programme linéaire possède une solution optimale, il ne peut y avoir que deux cas :

1. La droite de la fonction objectif intersecte le polyèdre en un point.
2. La droite de la fonction objectif intersecte le polyèdre sur un segment.

Dans le second cas, on peut remarquer que, comme tous les points du segment correspondent à une solution optimale, alors les extrémités de ce segment forment, en particulier, une solution optimale.

**Conclusion :** Si le programme linéaire possède une solution optimale, alors il existe au moins un sommet du polyèdre qui correspond à cette solution optimale.

### 7.5 Une évolution naïve de la résolution graphique

#### 7.5.1 Principe de l'algorithme

Lorsqu'il y a plus de deux variables, la résolution graphique n'est plus possible, puisqu'il y a autant de dimensions que de variables. Il est donc nécessaire d'utiliser une autre méthode.

**Lien entre géométrie et algèbre.** La notion géométrique de sommet du polyèdre correspond à la notion algébrique de solution réalisable. Puisque il y a un lien entre solution et sommet, on peut imaginer l'algorithme suivant :

 **Un algorithme naïf**

Énumérer tous les sommets du polyèdre. Pour chaque point, évaluer la fonction objectif. Conserver le point qui l'optimise.

Cet algorithme fonctionne lorsque le nombre de sommets est limité. Mais, le nombre de sommets peut être très important :

$$C_n^m = \frac{n!}{m!(n-m)!}$$

Et comme chaque sommet ne correspond pas nécessairement à une solution admissible, il est nécessaire de vérifier l'intégralité des contraintes pour vérifier qu'un sommet correspond à une solution admissible. Il est donc impératif d'utiliser un algorithme plus fin, plus efficace.

#### 7.5.1.1 Bases voisines et pivotage

Le polyèdre des solutions étant convexe, soit un sommet correspond à une solution optimale, soit il existe un voisin de ce sommet permettant d'améliorer la valeur de la fonction objectif. On en déduit l'algorithme suivant :



##### Principe de l'algorithme.

Passer de point extrême en point extrême, en augmentant la valeur de la fonction objectif, jusqu'à atteindre l'optimum.



##### Squelette de l'algorithme

1. Commencer avec une solution de base réalisable : un point extrême
2. Trouver un autre point extrême permettant d'augmenter la fonction objectif
3. Arrêt lorsqu'il n'existe plus de point extrême augmentant la fonction objectif

L'algorithme du simplexe utilise cette méthode. Mais pour cela, il est nécessaire de placer le programme linéaire sous forme normale (aussi appelée forme standard). La mise sous forme normale est elle-même très simple lorsque l'algorithme est placé sous forme canonique. Nous allons donc voir comment placer un programme linéaire sous forme canonique puis sous forme normale.

### 7.6 Forme canonique

La forme canonique d'un programme linéaire est une manière de représenter un programme linéaire de manière systématique. Cela permet de simplifier la résolution en diminuant le nombre de cas à traiter. En particulier, cela simplifie grandement la programmation d'un algorithme de résolution. Concrètement, il s'agit de transformer un programme linéaire  $P_1$  en un autre programme linéaire équivalent  $P_2$ . La résolution de  $P_2$  permettra de retrouver automatiquement une solution pour le programme  $P_1$  en effectuant les transformations inverses. La forme canonique choisie dans ce cours est la suivante :

1. La fonction objectif doit être maximisée.
2. Toutes les contraintes doivent être du type  $\leq$ .
3. Toutes les variables sont positives.

#### 7.6.1 Les règles de transformation

Pour obtenir la forme canonique à partir d'un programme linéaire, il suffit d'appliquer un ensemble de règles tant qu'il est possible d'en appliquer une. Suivant l'ordre d'application des règles, il est possible d'obtenir plus ou moins rapidement la forme canonique.

## [Règle 1] Fonction objectif

Si le programme linéaire doit minimiser sa fonction objectif  $f(x)$ , alors on peut le transformer en un problème de maximisation qui sera équivalent. Pour cela, il suffit de maximiser  $-f(x)$ .

Exemple :

### 💡 Programme linéaire avant application de la règle 1

$$\begin{array}{lll} \text{Min} & 3X_1 - 2X_2 + 8X_3 \\ \text{T.Q.} & \\ & 5X_1 - 2X_2 + 4X_3 \leq 8 \\ & X_1 + 3X_2 + 8X_3 \leq 25 \\ & 9X_1 + 6X_2 - 3X_3 \leq 17 \\ \\ & X_1, X_2, X_3 \geq 0 \end{array}$$

Le programme linéaire doit minimiser la fonction objectif  $3X_1 - 2X_2 + 8X_3$ . Cela revient donc à maximiser la fonction objectif  $-3X_1 + 2X_2 - 8X_3$ . Ce qui nous donne :

### 💡 Programme linéaire après application de la règle 1

$$\begin{array}{lll} \text{Max} & -3X_1 + 2X_2 - 8X_3 \\ \text{T.Q.} & \\ & 5X_1 - 2X_2 + 4X_3 \leq 8 \\ & X_1 + 3X_2 + 8X_3 \leq 25 \\ & 9X_1 + 6X_2 - 3X_3 \leq 17 \\ \\ & X_1, X_2, X_3 \geq 0 \end{array}$$

## [Règle 2] Équation

Si le programme contient une équation, on la remplace par deux inéquations équivalentes, l'une de supériorité, l'autre d'infériorité.

Exemple :

### 💡 Programme linéaire avant application de la règle 2

$$\begin{array}{lll} \text{Max} & 3X_1 - 2X_2 + 8X_3 \\ \text{T.Q.} & \\ & 5X_1 - 2X_2 + 4X_3 \leq 8 \\ & X_1 + 3X_2 + 8X_3 = 25 \\ & 9X_1 + 6X_2 - 3X_3 \leq 17 \\ \\ & X_1, X_2, X_3 \geq 0 \end{array}$$

La deuxième contrainte du programme linéaire est une équation :

$$X_1 + 3X_2 + 8X_3 = 25$$

On peut la remplacer par deux contraintes qui auront exactement le même impact sur le résultat :

$$X_1 + 3X_2 + 8X_3 \leq 25$$

et

$$X_1 + 3X_2 + 8X_3 \geq 25$$

En effet, ces deux contraintes imposent que  $X_1 + 3X_2 + 8X_3$  soit inférieur ou égal à 25 et supérieur ou égal à 25, ce qui implique que  $X_1 + 3X_2 + 8X_3 = 25$ .



### Programme linéaire après application de la règle 2

$$\begin{array}{lllll} \text{Max} & 3X_1 & - & 2X_2 & + & 8X_3 \\ \text{T.Q.} & 5X_1 & - & 2X_2 & + & 4X_3 & \leq & 8 \\ & X_1 & + & 3X_2 & + & 8X_3 & \geq & 25 \\ & X_1 & + & 3X_2 & + & 8X_3 & \leq & 25 \\ & 9X_1 & + & 6X_2 & - & 3X_3 & \leq & 17 \\ \\ & X_1, X_2, X_3 & \geq & 0 \end{array}$$

### [Règle 3] Variable non contrainte

Si une variable  $X_1$  n'a pas de contrainte de signe, et que l'on arrive à déterminer que  $X_1 \geq -300$ , il suffit de poser  $X'_1 = X_1 + 300$  et l'on aura  $X'_1 \geq 0$ . Mais, si on ne connaît pas de borne inférieure pour  $X_1$ , on peut poser  $X_1 = X'_1 - X''_1$  avec  $X'_1 \geq 0$  et  $X''_1 \geq 0$  car tout nombre peut être représenté comme la différence de deux nombres positifs ou nuls.

Si une variable  $X_1$  n'a pas de contrainte de signe, on la remplace par deux variables positives  $X'_1$  et  $X''_1$  telles que  $X_1 = X'_1 - X''_1$ .

**Exemple :**



### Programme linéaire avant application de la règle 3

$$\begin{array}{lllll} \text{Max} & 3X_1 & - & 2X_2 & + & 8X_3 \\ \text{T.Q.} & 5X_1 & - & 2X_2 & + & 4X_3 & \leq & 8 \\ & X_1 & + & 3X_2 & + & 8X_3 & \leq & 25 \\ & 9X_1 & + & 6X_2 & - & 3X_3 & \leq & 17 \\ \\ & X_1, X_2 & \geq & 0 \end{array}$$

La variable  $X_3$  n'a pas de contrainte de signe. On la remplace donc par deux variables  $X'_3 \geq 0$  et  $X''_3 \geq 0$  positives telles que :  $X_3 = X'_3 - X''_3$ . Le programme linéaire devient donc :



### Programme linéaire après application de la règle 3

$$\begin{array}{ll}
 \text{Max} & 3X_1 - 2X_2 + 8X'_3 - 8X''_3 \\
 \text{T.Q.} & \\
 & 5X_1 - 2X_2 + 4X'_3 - 4X''_3 \leq 8 \\
 & X_1 + 3X_2 + 8X'_3 - 8X''_3 \leq 25 \\
 & 9X_1 + 6X_2 - 3X'_3 - 3X''_3 \leq 17 \\
 \\ 
 & X_1, X_2, X'_3, X''_3 \geq 0
 \end{array}$$

### [Règle 4] Variable négative

Si une variable  $X_1$  est négative, on la remplace par une variable positive  $X'_1 = -X_1$ .

**Exemple :**



### Programme linéaire avant application de la règle 4

$$\begin{array}{ll}
 \text{Max} & 3X_1 - 2X_2 + 8X_3 \\
 \text{T.Q.} & \\
 & 5X_1 - 2X_2 + 4X_3 \leq 8 \\
 & X_1 + 3X_2 + 8X_3 \leq 25 \\
 & 9X_1 + 6X_2 - 3X_3 \leq 17 \\
 \\ 
 & X_1, X_2 \geq 0 \\
 & X_3 \leq 0
 \end{array}$$

La variable  $X_3$  est négative. On la remplace par une variable  $X'_3$  positive telle que :  $X'_3 = -X_3$ . Le programme linéaire devient donc :



### Programme linéaire après application de la règle 4

$$\begin{array}{ll}
 \text{Max} & 3X_1 - 2X_2 - 8X'_3 \\
 \text{T.Q.} & \\
 & 5X_1 - 2X_2 - 4X'_3 \leq 8 \\
 & X_1 + 3X_2 - 8X'_3 \leq 25 \\
 & 9X_1 + 6X_2 + 3X'_3 \leq 17 \\
 \\ 
 & X_1, X_2, X'_3 \geq 0
 \end{array}$$

### [Règle 5] Contrainte de type $\geq$

Si une contrainte est une inéquation de type  $\geq$ , on la remplace par une inéquation de type  $\leq$  en inversant les signes de tous les membres de l'inéquation.

**Exemple :**



### Programme linéaire avant application de la règle 5

$$\begin{array}{lll}
 \text{Max} & 3X_1 - 2X_2 - 8X'_3 \\
 \text{T.Q.} & \\
 & 5X_1 - 2X_2 - 4X'_3 \geq 8 \\
 & X_1 + 3X_2 - 8X'_3 \leq 25 \\
 & 9X_1 + 6X_2 + 3X'_3 \leq 17 \\
 \\ 
 & X_1, X_2, X'_3 \geq 0
 \end{array}$$

La première contrainte est une inéquation de type  $\geq$  :  $5X_1 - 2X_2 - 4X'_3 \geq 8$ . Cette inéquation est équivalente à :  $-5X_1 + 2X_2 + 4X'_3 \leq -8$ .



### Programme linéaire après application de la règle 5

$$\begin{array}{lll}
 \text{Max} & 3X_1 - 2X_2 - 8X'_3 \\
 \text{T.Q.} & \\
 & -5X_1 + 2X_2 + 4X'_3 \leq -8 \\
 & X_1 + 3X_2 - 8X'_3 \leq 25 \\
 & 9X_1 + 6X_2 + 3X'_3 \leq 17 \\
 \\ 
 & X_1, X_2, X'_3 \geq 0
 \end{array}$$

#### 7.6.2 Un exemple complet

En partant du programme linéaire suivant, nous allons appliquer l'ensemble des règles pour obtenir un programme linéaire équivalent sous forme canonique :



### Programme linéaire initial

$$\begin{array}{lll}
 \text{Min} & -3X_1 + 2X_2 - 8X_3 \\
 \text{T.Q.} & \\
 & 5X_1 - 2X_2 + 4X_3 \leq 8 \\
 & X_1 + 3X_2 + 8X_3 = 25 \\
 & -9X_1 - 6X_2 + 3X_3 \geq 17 \\
 \\ 
 & X_1 \geq 0 \\
 & X_2 \leq 0
 \end{array}$$

Nous pouvons voir qu'il n'est pas sous forme canonique pour les raisons suivantes :

1. Il s'agit d'un problème de minimisation
2. La deuxième contrainte est une équation
3. La troisième contrainte est une inéquation de type  $\geq$
4. La variable  $X_3$  n'a pas de contrainte de signe
5. La variable  $X_2$  est négative

L'ordre dans lequel nous allons appliquer les différentes règles est arbitraire. Il suffit d'appliquer l'une des règles tant que le programme n'est pas sous forme canonique. Cependant, l'ordre d'application de ces règles peut avoir un impact sur le nombre d'étapes.

### 7.6.2.1 Étape 1 : application de la règle 1

Le programme a pour but de minimiser la fonction objectif. Nous allons le transformer pour obtenir un programme dont le but est de maximiser la fonction objectif. On obtient :

#### Programme linéaire après application de la règle 1

$$\begin{array}{lllll} \text{Max} & 3X_1 & - & 2X_2 & + & 8X_3 \\ \text{T.Q.} & 5X_1 & - & 2X_2 & + & 4X_3 & \leq & 8 \\ & X_1 & + & 3X_2 & + & 8X_3 & = & 25 \\ & -9X_1 & - & 6X_2 & + & 3X_3 & \geq & 17 \\ \\ & X_1 & \geq & 0 \\ & X_2 & \leq & 0 \end{array}$$

### 7.6.2.2 Étape 2 : application de la règle 4

La variable  $X_2$  est négative. Nous allons procéder à un changement de variable pour obtenir une variable positive : on pose  $X'_2 = -X_2$ , ce qui nous donne :

#### Programme linéaire après application de la règle 4

$$\begin{array}{lllll} \text{Max} & 3X_1 & + & 2X'_2 & + & 8X_3 \\ \text{T.Q.} & 5X_1 & + & 2X'_2 & + & 4X_3 & \leq & 8 \\ & X_1 & - & 3X'_2 & + & 8X_3 & = & 25 \\ & -9X_1 & + & 6X'_2 & + & 3X_3 & \geq & 17 \\ \\ & X_1, X'_2 & \geq & 0 \end{array}$$

### 7.6.2.3 Étape 3 : application de la règle 3

La variable  $X_3$  ne possède pas de contrainte de signe. Nous allons procéder à un changement de variable pour obtenir deux variables positives : on pose  $X_3 = X'_3 - X''_3$ , ce qui nous donne :

#### Programme linéaire après application de la règle 3

$$\begin{array}{lllll} \text{Max} & 3X_1 & + & 2X'_2 & + & 8X'_3 & - & 8X''_3 \\ \text{T.Q.} & 5X_1 & + & 2X'_2 & + & 4X'_3 & - & 4X''_3 & \leq & 8 \\ & X_1 & - & 3X'_2 & + & 8X'_3 & - & 8X''_3 & = & 25 \\ & -9X_1 & + & 6X'_2 & + & 3X'_3 & - & 3X''_3 & \geq & 17 \\ \\ & X_1, X'_2, X'_3, X''_3 & \geq & 0 \end{array}$$

#### 7.6.2.4 Étape 4 : application de la règle 2

La deuxième contrainte est une équation. On la remplace par un système équivalent de deux inéquations :

 Programme linéaire après application de la règle 2

$$\begin{array}{llllll} \text{Max} & 3X_1 & + & 2X'_2 & + & 8X'_3 - 8X''_3 \\ \text{T.Q.} & 5X_1 & + & 2X'_2 & + & 4X'_3 - 4X''_3 \leq 8 \\ & X_1 & - & 3X'_2 & + & 8X'_3 - 8X''_3 \leq 25 \\ & X_1 & - & 3X'_2 & + & 8X'_3 - 8X''_3 \geq 25 \\ & -9X_1 & + & 6X'_2 & + & 3X'_3 - 3X''_3 \geq 17 \\ \\ & X_1, X'_2, X'_3, X''_3 & \geq & 0 \end{array}$$

#### 7.6.2.5 Étape 5 : application de la règle 5

La troisième contrainte ainsi que la quatrième sont des inéquations de type  $\geq$ . On inverse le signe de chacun de leurs membres pour obtenir deux inéquations de type  $\leq$  :

 Programme linéaire après application de la règle 5

$$\begin{array}{llllll} \text{Max} & 3X_1 & + & 2X'_2 & + & 8X'_3 - 8X''_3 \\ \text{T.Q.} & 5X_1 & + & 2X'_2 & + & 4X'_3 - 4X''_3 \leq 8 \\ & X_1 & - & 3X'_2 & + & 8X'_3 - 8X''_3 \leq 25 \\ & -X_1 & + & 3X'_2 & - & 8X'_3 + 8X''_3 \leq -25 \\ & 9X_1 & - & 6X'_2 & - & 3X'_3 + 3X''_3 \leq -17 \\ \\ & X_1, X'_2, X'_3, X''_3 & \geq & 0 \end{array}$$

#### 7.6.2.6 Étape 6 : bilan de la transformation

Le programme linéaire obtenu après l'application des différentes règles est le suivant :

 Programme linéaire final

$$\begin{array}{llllll} \text{Max} & 3X_1 & + & 2X'_2 & + & 8X'_3 - 8X''_3 \\ \text{T.Q.} & 5X_1 & + & 2X'_2 & + & 4X'_3 - 4X''_3 \leq 8 \\ & X_1 & - & 3X'_2 & + & 8X'_3 - 8X''_3 \leq 25 \\ & -X_1 & + & 3X'_2 & - & 8X'_3 + 8X''_3 \leq -25 \\ & 9X_1 & - & 6X'_2 & - & 3X'_3 + 3X''_3 \leq -17 \\ \\ & X_1, X'_2, X'_3, X''_3 & \geq & 0 \end{array}$$

Après vérification, on peut voir que :

1. Il s'agit d'un problème de maximisation
2. Toutes les contraintes sont des inéquations de type  $\leq$
3. Toutes les variables sont positives

Le programme est donc sous forme canonique. Trouver une solution à ce problème revient à trouver une solution pour le problème initial.



### Solution finale

Supposons que l'on trouve la solution suivante :

$$X_1 = a, X_2' = b, X_3' = c, X_3'' = d$$

On va retrouver les valeurs des variables  $X_2$  et  $X_3$  à partir de  $X_2'$ ,  $X_3'$  et  $X_3''$

$$X_2 = -X_2' = -b$$

$$X_3 = X_3' - X_3'' = c - d$$

Nous obtenons donc la solution suivante pour le programme linéaire initial :

$$X_1 = a, X_2 = -b, X_3 = c - d$$

On peut ensuite calculer la valeur de la fonction objectif du programme linéaire initial :

$$\begin{aligned} -3X_1 + 2X_2 - 8X_3 &= -3 \cdot (a) + 2 \cdot (-b) - 8 \cdot (c - d) \\ &= -3 \cdot a - 2 \cdot b - 8 \cdot c + 8 \cdot d \end{aligned}$$

## 7.7 Forme normale

L'algorithme du simplexe a besoin que le programme linéaire soit représenté sous forme standard pour pouvoir le résoudre :

1. La fonction objectif doit être maximisée.
2. Toutes les contraintes sont des équations.
3. Les seconds membres sont des nombres positifs ou nuls.

Or, tout programme linéaire en forme canonique peut s'écrire en forme standard, et tout programme linéaire sous forme standard peut s'écrire sous forme canonique. Ainsi, le plus simple est de placer le programme linéaire sous forme canonique, puis d'appliquer les règles suivantes :

### 7.7.1 Règles de transformation de la forme canonique vers la forme standard

#### [Règle 6] Passage d'une inéquation à une équation

Si une contrainte est une inéquation, on la remplace par une équation en ajoutant une variable d'écart au premier membre. Une variable d'écart est la quantité qui, ajoutée au membre gauche d'une inéquation permet de transformer la contrainte en équation.

Exemple :



### Programme linéaire avant application de la règle 6

$$\begin{aligned}
 \text{Max} \quad & 3X_1 - 2X_2 - 8X_3 \\
 \text{T.Q.} \quad & 5X_1 - 2X_2 - 4X_3 \leq 8 \\
 & X_1 + 3X_2 - 8X_3 \leq 25 \\
 & 9X_1 + 6X_2 + 3X_3 \leq 17 \\
 \\ 
 & X_1, X_2, X_3 \geq 0
 \end{aligned}$$

Il y a trois contraintes représentées par des inéquations. Il est donc nécessaire d'introduire trois variables d'écart  $e_1, e_2$  et  $e_3$ .



### Programme linéaire après application de la règle 6

$$\begin{aligned}
 \text{Max} \quad & 3X_1 - 2X_2 - 8X_3 \\
 \text{T.Q.} \quad & 5X_1 - 2X_2 - 4X_3 + e_1 = 8 \\
 & X_1 + 3X_2 - 8X_3 + e_2 = 25 \\
 & 9X_1 + 6X_2 + 3X_3 + e_3 = 17 \\
 \\ 
 & X_1, X_2, X_3 \geq 0 \\
 & e_1, e_2, e_3 \geq 0
 \end{aligned}$$

### [Règle 7] Positivité du second membre des équations

Si une contrainte comporte un second membre négatif, on multiplie par  $-1$  chaque membre de cette contrainte

Exemple :



### Programme linéaire avant application de la règle 7

$$\begin{aligned}
 \text{Max} \quad & 3X_1 - 2X_2 - 8X_3 \\
 \text{T.Q.} \quad & 5X_1 - 2X_2 - 4X_3 = 8 \\
 & X_1 + 3X_2 - 8X_3 = -25 \\
 & 9X_1 + 6X_2 + 3X_3 = 17 \\
 \\ 
 & X_1, X_2, X_3 \geq 0
 \end{aligned}$$

La deuxième contrainte possède un second membre négatif :

$$X_1 + 3X_2 - 8X_3 = -25$$

En multipliant chaque membre par  $-1$  on obtient :

$$-X_1 - 3X_2 + 8X_3 = 25$$



### Programme linéaire après application de la règle 7

$$\begin{array}{llll} \text{Max} & 3X_1 & - & 2X_2 & - & 8X_3 \\ \text{T.Q.} & 5X_1 & - & 2X_2 & - & 4X_3 = 8 \\ & -X_1 & - & 3X_2 & + & 8X_3 = 25 \\ & 9X_1 & + & 6X_2 & + & 3X_3 = 17 \\ \\ X_1, X_2, X_3 & \geq & 0 \end{array}$$

### 7.7.2 Équivalence entre les deux formes

Voici un programme linéaire sous forme canonique, puis le programme linéaire transformé en forme standard :



### Programme linéaire sous forme canonique

$$\begin{array}{llll} \text{Max} & 3X_1 & + & 5X_2 \\ \text{T.Q.} & X_1 & \leq & 4 \\ & 2X_2 & \leq & 12 \\ & 3X_1 & + & 2X_2 \leq 18 \\ \\ X_1, X_2 & \geq & 0 \end{array}$$



### Programme linéaire sous forme standard

$$\begin{array}{llll} \text{Max} & 3X_1 & + & 5X_2 \\ \text{T.Q.} & X_1 & + e_1 & = 4 \\ & 2X_2 & + e_2 & = 12 \\ & 3X_1 & + 2X_2 & + e_3 = 18 \\ \\ X_1, X_2 & \geq & 0 \end{array}$$

Il y a une équivalence totale entre les deux formes (canonique et standard) :

1. Toute solution réalisable pour la forme canonique peut être augmentée en une solution réalisable pour la forme standard.
2. Toute solution réalisable pour la forme standard peut être tronquée en une solution réalisable pour la forme canonique.
3. De plus, les fonctions objectifs étant identiques, il y a bien équivalence entre les deux problèmes.

Illustrons cette correspondance entre solutions :

- Forme canonique. Solution :  $(3, 2)$
- Forme normale. Solution :  $(3, 2, 1, 8, 5)$

## 7.8 L'algorithme du simplexe

L'algorithme du simplexe est l'un des algorithmes les plus utilisés pour résoudre les problèmes de programmation linéaire. Cet algorithme a été conçu en 1947 par George Dantzig. Il existe plusieurs versions de cet algorithme. Nous verrons la méthode algébrique et la méthode des tableaux.

### 7.8.1 Présentation de l'algorithme

Voici une version simplifiée de l'algorithme du simplexe. Cette version intègre la préparation du problème et considère qu'il existe toujours une solution initiale que l'on peut déterminer de manière triviale :

---

#### Algorithm 11: Algorithme du simplexe

---

**Data:** Un programme linéaire

**Result:** Une solution optimale pour ce programme linéaire

Placer le problème sous forme canonique;

Placer le problème sous forme normale;

Déterminer une solution de base réalisable initiale grâce aux variables d'écart;

Reformuler les équations :

- membre gauche : les variables en base
- membre droit : expression n'utilisant que des variables hors base

**while** La solution optimale n'est pas atteinte **do**

    Déterminer la variable entrante;

    Déterminer la variable sortante;

    Réaliser l'opération de pivotage;

**end**

**return** La solution optimale;

---

### 7.8.2 Méthode du dictionnaire

Nous allons dérouler cette méthode directement sur un exemple. On considère que le programme linéaire de notre exemple se trouve déjà sous forme normale :



#### Programme linéaire sous forme normale

$$\begin{array}{lllllll} z = & 7X_1 + 9X_2 + 18X_3 + 17X_4 + 0e_1 + 0e_2 + 0e_3 \\ \text{t.q. } & 2X_1 + 4X_2 + 5X_3 + 7X_4 + e_1 & = & 42 \\ & X_1 + X_2 + 2X_3 + 2X_4 + e_2 & = & 17 \\ & X_1 + 2X_2 + 3X_3 + 3X_4 + e_3 & = & 24 \\ & X_1 & X_2 & X_3 & X_4 & e_1 & e_2 & e_3 & \geq & 0 \end{array}$$

On cherche une solution de base réalisable. Le plus simple étant de placer les  $n$  variables du problème initial hors base (on leur affecte donc la valeur 0) et de placer les variables d'écart en base. Cela permet de déterminer directement leur valeur. Par exemple :

- $2 \cdot X_1 + 4 \cdot X_2 + 5 \cdot X_3 + 7 \cdot X_4 + e_1 = 42$
- $\Rightarrow 2 \cdot 0 + 4 \cdot 0 + 5 \cdot 0 + 7 \cdot 0 + e_1 = 42$
- $\Rightarrow e_1 = 42$

On peut ainsi récupérer les valeurs des variables  $e_1, e_2$  et  $e_3$  lorsque  $X_1 = 0, X_2 = 0, X_3 = 0$  et  $X_4 = 0$ . Les variables  $X_1 = 0, X_2 = 0, X_3 = 0$  et  $X_4 = 0$  sont dites hors base, et les variables  $e_1, e_2$  et  $e_3$  sont dites en base. De plus, comme on a  $X_1 = 0, X_2 = 0, X_3 = 0$  et  $X_4 = 0$ , on en déduit que la valeur de la solution actuelle (la fonction objectif) est 0.

Pour résumer :



### Initialisation

- Variables hors base :  $X_1, X_2, X_3, X_4$
- Variables en base :  $e_1 = 42, e_2 = 17, e_3 = 24$
- Comme  $X_1 = X_2 = X_3 = X_4 = 0$ , on obtient directement que  $e_1 = 42, e_2 = 17, e_3 = 24$
- La solution de base réalisable obtenue est la suivante :

$$X_1 = 0, X_2 = 0, X_3 = 0, X_4 = 0, e_1 = 42, e_2 = 17, e_3 = 24$$

- On peut ainsi calculer la valeur de la fonction objectif :  $z = 0$

A partir de cette solution, on peut reformuler chaque équation en plaçant les variables en base comme membre gauche (et rien d'autre). Par exemple, la première équation nous permet d'exprimer la variable  $e_1$  :

- $2 \cdot X_1 + 4 \cdot X_2 + 5 \cdot X_3 + 7 \cdot X_4 + e_1 = 42$
- $\Rightarrow e_1 = 42 - 2 \cdot X_1 - 4 \cdot X_2 - 5 \cdot X_3 - 7 \cdot X_4$

On obtient ainsi le problème reformulé :



### Reformulation

$$\begin{aligned} z &= 7X_1 + 9X_2 + 18X_3 + 17X_4 \\ e_1 &= 42 - 2X_1 - 4X_2 - 5X_3 - 7X_4 \\ e_2 &= 17 - X_1 - X_2 - 2X_3 - 2X_4 \\ e_3 &= 24 - X_1 - 2X_2 - 3X_3 - 3X_4 \end{aligned}$$

Nous entrons maintenant dans la boucle de l'algorithme. Nous devons déterminer à chaque étape quelle variable va rentrer en base, et quelle variable va sortir de base. Il s'agit de l'opération de pivotage :



### Comment réaliser un pivotage

1. Opération de pivotage
  - Une variable hors base entre en base
  - Une variable en base sort de la base
2. Choix des variables
  - Variable entrante : max des coefficients dans la fonction objectif
  - Variable sortante : celle correspondant à la ligne contrignant le plus la variable entrante

### 7.8.2.1 Étape 1.

Le critère de sélection de la variable entrante est de prendre la variable hors base qui fournit le plus fort taux d'augmentation de la fonction objectif. Cela revient à dire que l'on va sélectionner la variable dont le coefficient est le plus élevé dans la fonction objectif. Il s'agit du **critère de Dantzig**. Dans notre exemple, la variable qui à le coefficient le plus élevé dans la fonction objectif est la variable  $X_3$  :



#### Choix de la colonne pivot

$$\begin{aligned} z &= 7X_1 + 9X_2 + 18X_3 + 17X_4 \\ e_1 &= 42 - 2X_1 - 4X_2 - 5X_3 - 7X_4 \\ e_2 &= 17 - X_1 - X_2 - 2X_3 - 2X_4 \\ e_3 &= 24 - X_1 - 2X_2 - 3X_3 - 3X_4 \end{aligned}$$

Le critère de Dantzig nous fait sélectionner  $X_3$

Une fois la variable entrante sélectionnée, il faut déterminer la variable sortante. Cette variable sortante est nécessairement l'une des variables en base, soit  $e_1, e_2$  ou  $e_3$ . Pour déterminer quelle variable va sortir de la base, nous allons vérifier pour chaque équation laquelle va contraindre le plus notre variable entrante, soit  $X_3$  dans notre cas. On sait que les variables en base ont une valeur supérieure ou égale à 0. Par exemple, grâce à la première équation on a :

$$\begin{aligned} \underbrace{e_1}_{\geq 0} &= \underbrace{42 - 2 \cdot X_1 - 4 \cdot X_2 - 5 \cdot X_3 - 7 \cdot X_4}_{\geq 0} \\ \Rightarrow 42 - 2 \cdot X_1 - 4 \cdot X_2 - 5 \cdot X_3 - 7 \cdot X_4 &\geq 0 \\ \Rightarrow 42 - 2 \cdot X_1 - 4 \cdot X_2 - 7 \cdot X_4 &\geq 5 \cdot X_3 \end{aligned}$$

Or  $X_1 = X_2 = X_4 = 0$  donc :

$$\begin{aligned} 42 &\geq 5 \cdot X_3 \\ \Rightarrow X_3 &\leq 8.4 \end{aligned}$$

On fait de même avec  $e_2$  et  $e_3$ . On trouve

- $e_1 \geq 0 \Rightarrow X_3 \leq 8.4$
- $e_2 \geq 0 \Rightarrow X_3 \leq 8.5$
- $e_3 \geq 0 \Rightarrow X_3 \leq 8$

On peut ainsi se rendre compte que  $e_3$  contraint le plus la valeur de  $X_3$ . C'est donc  $e_3$  que nous allons faire sortir de base.

**Ainsi :**  $X_3$  entre en base et  $e_3$  sort de base. Pour rappel, les équations du programme linéaire doivent avoir les variables en base comme membre gauche. Nous allons donc modifier l'équation qui possède  $e_3$  comme membre gauche pour exprimer  $X_3$  en fonction des variables hors base (y compris  $e_3$ ) :

$$\begin{aligned} e_3 &= 24 - X_1 - 2 \cdot X_2 - 3 \cdot X_3 - 3 \cdot X_4 \\ \Rightarrow 3 \cdot X_3 &= 24 - X_1 - 2 \cdot X_2 - 3 \cdot X_4 - e_3 \\ \Rightarrow X_3 &= 8 - \frac{1}{3} \cdot X_1 - \frac{2}{3} \cdot X_2 - X_4 - \frac{1}{3} \cdot e_3 \end{aligned}$$

Cette équation va remplacer la précédente dans le programme linéaire. De plus, nous allons remplacer  $X_3$  dans chacune des autres équations ainsi que dans la fonction objectif. Par exemple, la fonction objectif sera transformée de la manière suivante :

- $z = 7 \cdot X_1 + 9 \cdot X_2 + 18 \cdot X_3 + 17 \cdot X_4$
- $z = 7 \cdot X_1 + 9 \cdot X_2 + 18 \cdot (8 - \frac{1}{3} \cdot X_1 - \frac{2}{3} \cdot X_2 - X_4 - \frac{1}{3} \cdot e_3) + 17 \cdot X_4$
- $z = 144 + X_1 - 3 \cdot X_2 - X_4 - 6 \cdot e_3$

On fait de même pour les autres équations et on obtient :

Programme mis à jour									
$z =$	144	+	$X_1$	-	$3 \cdot X_2$	-	$X_4$	-	$6 \cdot e_3$
$X_3 =$	8	-	$\frac{1}{3} \cdot X_1$	-	$\frac{2}{3} \cdot X_2$	-	$X_4$	-	$\frac{1}{3} \cdot e_3$
$e_1 =$	2	-	$\frac{1}{3} \cdot X_1$	-	$\frac{2}{3} \cdot X_2$	-	$2 \cdot X_4$	+	$\frac{5}{3} \cdot e_3$
$e_2 =$	1	-	$\frac{1}{3} \cdot X_1$	+	$\frac{1}{3} \cdot X_2$			+	$\frac{2}{3} \cdot e_3$

### 7.8.2.2 Étape 2.

Dans notre exemple, la variable dont le coefficient est le plus élevé dans la fonction objectif est la variable  $X_1$  :

Choix de la colonne pivot									
$z =$	144	+	$X_1$	-	$3 \cdot X_2$	-	$X_4$	-	$6 \cdot e_3$
$X_3 =$	8	-	$\frac{1}{3} \cdot X_1$	-	$\frac{2}{3} \cdot X_2$	-	$X_4$	-	$\frac{1}{3} \cdot e_3$
$e_1 =$	2	-	$\frac{1}{3} \cdot X_1$	-	$\frac{2}{3} \cdot X_2$	-	$2 \cdot X_4$	+	$\frac{5}{3} \cdot e_3$
$e_2 =$	1	-	$\frac{1}{3} \cdot X_1$	+	$\frac{1}{3} \cdot X_2$			+	$\frac{2}{3} \cdot e_3$

Il faut maintenant déterminer la variable sortante :

- $X_3 \geq 0 \Rightarrow X_1 \leq 24$
- $e_1 \geq 0 \Rightarrow X_1 \leq 6$
- $e_2 \geq 0 \Rightarrow X_1 \leq 3$

La variable qui va sortir de base est donc la variable  $e_2$ . On va donc utiliser l'équation contenant  $e_2$  comme membre gauche pour exprimer  $X_1$  en fonction des variables hors base. Ainsi, on trouve :

$$X_1 = 3 + X_2 - 3 \cdot e_2 + 2 \cdot e_3$$

Cette équation va remplacer la précédente dans le programme linéaire. De plus, nous allons remplacer  $X_1$  dans chacune des autres équations ainsi que dans la fonction objectif. On obtient le programme linéaire suivant :

Programme mis à jour									
$z =$	147	-	$2 \cdot X_2$	-	$X_4$	-	$3 \cdot e_2$	-	$4 \cdot e_3$
$X_3 =$	7	-	$X_2$	-	$X_4$	+	$e_2$	-	$e_3$
$e_1 =$	1	-	$X_2$	-	$2 \cdot X_4$	-	$e_2$	+	$e_3$
$X_1 =$	3	+	$X_2$			-	$3 \cdot e_2$	+	$2 \cdot e_3$

### 7.8.2.3 Étape 3.

Il n'existe plus de variables dans la fonction objectif possédant un coefficient positif. C'est la fin de l'algorithme.  $X_2, X_4, e_2$  et  $e_3$  étant hors base, elles valent 0. On peut donc calculer la valeur de la fonction objectif, qui est de 147. On peut également calculer la valeur des différentes variables en base :  $X_3 = 7, e_1 = 1$  et  $X_1 = 3$ . La solution finale pour le problème sous forme normale peut donc s'exprimer comme ceci :

$$(X_1 = 3, X_2 = 0, X_3 = 7, X_4 = 0, e_1 = 1, e_2 = 0, e_3 = 0)$$

La solution pour la forme canonique est :

$$(X_1 = 3, X_2 = 0, X_3 = 7, X_4 = 0)$$

### 7.8.3 Méthode des tableaux

La méthode des tableaux

---

#### Algorithm 12: Algorithme du simplexe

---

**Data:** Un programme linéaire

**Résultat:** Une solution optimale pour ce programme linéaire

Placer le problème sous forme canonique;

Placer le problème sous forme normale;

Réécrire le problème en le plaçant dans un tableau de la manière suivante :

- La première colonne contient les noms des différentes contraintes
- La première ligne contient les noms des variables du programme linéaire
- une case du tableau corresponds au coefficient de la variable pour la contrainte considérée.
- La dernière colonne contient les membres gauche de chaque contrainte
- La dernière ligne contient les coefficients de la fonction objectif.

**while** *La solution optimale n'est pas atteinte* **do**

    Déterminer la variable entrante;

    Déterminer la variable sortante;

    Réaliser l'opération de pivotage;

**end**

**return** La solution optimale;

---

### 7.8.4 Exemple de déroulement de l'algorithme

#### 7.8.4.1 Réécriture du problème sous forme de tableau

	X1	X2	X3	X4	e0	e1	e2	second membre
C0	2.00	4.00	5.00	7.00	1.00	0.00	0.00	42.00
C1	1.00	1.00	2.00	2.00	0.00	1.00	0.00	17.00
C2	1.00	2.00	3.00	3.00	0.00	0.00	1.00	24.00
Z	7.00	9.00	18.00	17.00	0.00	0.00	0.00	<b>0.00</b>

#### 7.8.4.2 Étape 1

##### Détection du pivot

- Calcul de la colonne du pivot. Le coefficient le plus élevé est celui de la variable  $X_3$  et vaut 18.00
- Calcul de la ligne du pivot :
  - $(C0) \frac{42.00}{5.00} = 8.40$
  - $(C1) \frac{17.00}{2.00} = 8.50$
  - $(C2) \frac{24.00}{3.00} = 8.00$

	X1	X2	<b>X3</b>	X4	e0	e1	e2	second membre
C0	2.00	4.00	<b>5.00</b>	7.00	1.00	0.00	0.00	42.00
C1	1.00	1.00	<b>2.00</b>	2.00	0.00	1.00	0.00	17.00
<b>C2</b>	<b>1.00</b>	<b>2.00</b>	<b>3.00</b>	<b>3.00</b>	<b>0.00</b>	<b>0.00</b>	<b>1.00</b>	<b>24.00</b>
Z	7.00	9.00	<b>18.00</b>	17.00	0.00	0.00	0.00	<b>0.00</b>

#### Division de la ligne du pivot par la valeur du pivot

	X1	X2	<b>X3</b>	X4	e0	e1	e2	second membre
C0	2.00	4.00	<b>5.00</b>	7.00	1.00	0.00	0.00	42.00
C1	1.00	1.00	<b>2.00</b>	2.00	0.00	1.00	0.00	17.00
<b>C2</b>	<b>0.33</b>	<b>0.67</b>	<b>1.00</b>	<b>1.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.33</b>	<b>8.00</b>
Z	7.00	9.00	<b>18.00</b>	17.00	0.00	0.00	0.00	<b>0.00</b>

#### Multiplication et soustraction

	X1	X2	<b>X3</b>	X4	e0	e1	e2	second membre
C0	0.33	0.67	<b>0.00</b>	2.00	1.00	0.00	-1.67	2.00
C1	0.33	-0.33	<b>0.00</b>	0.00	0.00	1.00	-0.67	1.00
<b>C2</b>	<b>0.33</b>	<b>0.67</b>	<b>1.00</b>	<b>1.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.33</b>	<b>8.00</b>
Z	1.00	-3.00	<b>0.00</b>	-1.00	0.00	0.00	-6.00	<b>-144.00</b>

#### 7.8.4.3 Étape 2

##### Détection du pivot

- Calcul de la colonne du pivot. Le coefficient le plus élevé est celui de la variable  $X_1$  et vaut 1.00
- Calcul de la ligne du pivot :
  - $(C0) \frac{2.00}{0.33} = 6.00$
  - $(C1) \frac{1.00}{0.33} = 3.00$
  - $(C2) \frac{8.00}{0.33} = 24.00$

	<b>X1</b>	X2	X3	X4	e0	e1	e2	second membre
C0	<b>0.33</b>	0.67	0.00	2.00	1.00	0.00	-1.67	2.00
<b>C1</b>	<b>0.33</b>	<b>-0.33</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>1.00</b>	<b>-0.67</b>	<b>1.00</b>
C2	<b>0.33</b>	0.67	1.00	1.00	0.00	0.00	0.33	8.00
Z	<b>1.00</b>	-3.00	0.00	-1.00	0.00	0.00	-6.00	<b>-144.00</b>

### Division de la ligne du pivot par la valeur du pivot

	X1	X2	X3	X4	e0	e1	e2	second membre
C0	0.33	0.67	0.00	2.00	1.00	0.00	-1.67	2.00
C1	1.00	-1.00	0.00	0.00	0.00	3.00	-2.00	3.00
C2	0.33	0.67	1.00	1.00	0.00	0.00	0.33	8.00
Z	1.00	-3.00	0.00	-1.00	0.00	0.00	-6.00	-144.00

### Multiplication et soustraction

	X1	X2	X3	X4	e0	e1	e2	second membre
C0	0.00	1.00	0.00	2.00	1.00	-1.00	-1.00	1.00
C1	1.00	-1.00	0.00	0.00	0.00	3.00	-2.00	3.00
C2	0.00	1.00	1.00	1.00	0.00	-1.00	1.00	7.00
Z	0.00	-2.00	0.00	-1.00	0.00	-3.00	-4.00	-147.00

#### 7.8.4.4 Étape 3

**Détection du pivot** Le coefficient le plus élevé est  $\leq 0$ . Fin de l'algorithme.



## Chapitre 8

# Les outils de la recherche opérationnelle

8.1 Pour la théorie des graphes

8.2 Pour la programmation linéaire



## Chapitre 9

# Propositions de sujets de présentation

1. histoire de la recherche opérationnelle
2. La méthode de Chris McKinlay pour maximiser ses chances sur OkCupid
3. Vulgarisation du travail de Gaspard Monge sur la théorie des déblais et des remblais
4. Analyse de l'activité pédophile dans les réseaux pairs à pairs
5. AlphaGo, de Google DeepMind
6. Histoire des supercalculateurs et techniques
7. Tutoriel sur une bibliothèque de manipulation de graphes
8. Les outils de la théorie des graphes
9. Le problème de la coloration de graphes
10. Algorithme de visualisation de graphe multi-échelle
11. La complexité algorithmique
12. La programmation dynamique
13. Les algorithmes génétiques
14. La méthode du recuit simulé
15. Le problème du sac à dos
16. Le problème de la tournée de véhicules
17. L'algorithme du pivot de Gauss
18. Tutoriel sur une bibliothèque de résolution de programme linéaire
19. Déterminer une solution de départ pour l'algorithme du simplexe
20. La méthode du primal et du dual
21. Deep learning, machine learning etc



# Bibliographie

- [1] Valentin Bouquet, Kymble Christophe, François Delbot, Gaétan Le Chat, and Jean-François Pradat-Peyre. Les perspectives du brexit évaluées par les ensembles dominants. *20ème congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision*, 2019.
- [2] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction à l'algorithmique*. Dunod, Paris, France, 1994.
- [3] François Delbot and Christian Laforest. Analytical and experimental comparison of six algorithms for the vertex cover problem. *J. Exp. Algorithmics*, 15 :1.4 :1.1–1.4 :1.27, November 2010.
- [4] M. R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [5] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
- [6] Maurice W Kirby. *Operational Research in War and Peace :The British Experience from the 1930s to 1970*. Number p247 in World Scientific Books. World Scientific Publishing Co. Pte. Ltd., April 2003.
- [7] Rolf Niedermeier and Peter Rossmanith. Upper bounds for vertex cover further improved. In Christoph Meinel and Sophie Tison, editors, *STACS 99*, pages 561–570, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.