Conception et analyse d'algorithmes

Robert Cori, Guillaume Hanrot, Claire Kenyon et Jean-Marc Steyaert

Table des matières

Introduction

Ce cours est principalement consacré à l'étude de la complexité des algorithmes de nature combinatoire et à quelques applications. Ce domaine est particulièrement vivant depuis les années 70 après la mise en évidence de la classe des algorithmes **NP** et de la célèbre conjecture $\mathbf{P} \neq \mathbf{NP}$. En effet, on peut dire que dans l'état actuel de nos connaissances théoriques et étant donné l'état de l'art en matière de conception et de réalisation de machines, seuls les algorithmes dont le temps d'exécution est majoré par une fonction polynomiale de la taille des données sont utilisables dans la pratique : les problèmes solubles par un algorithme dont le temps d'exécution est polynomial *dans le pire cas* constituent la classe **P**. Cette définition, très robuste, ne dépend pas des détails du modèle de calcul (Machine de Turing, PRAM, machie parallèle ...), lequel n'influence que les facteurs constants et éventuellement le degré du polynôme.
Les progrès de la vitesse des processeurs ne se traduisent en des gains d'efficacité des algorithmes, et donc en une amélioration tangible de la plus grande taille de problème soluble en une seconde, que pour les algorithmes de complexité polynomiale, comme l'illustre le tableau ci-dessous. On suppose que la vieille machine peut exécuter 10000 opérations en une seconde, et que la nouvelle macine est dix fois plus rapide. La première colonne donne la complexité de l'algorithme, la deuxième la plus grande taille de problème soluble en une seconde sur la vieille machine, la suivante donne la plus grande taille de problème soluble en une seconde sur la nouvelle machine. Il est clair que si l'algorithme utilisé est de complexité exponentielle, peu importe les progrès en architecture des machines : le problème restera de toute façon au-delà des capacités du programme.

$f(n)$	n	n'	Changement	n'/n
$10n$	1000	10000	$n' = 10n$	10
$20n$	500	5000	$n' = 10n$	10
$5n \log n$	250	1842	$\sqrt{10}n \leq n' \leq 10n$	7.37
$2n^2$	70	223	$n' = \sqrt{10}n$	3.16
2^n	13	16	$n' = n + 3$	–

Le fait de ne pas connaître d'algorithmes en temps polynomial pour résoudre certains problèmes d'optimisation combinatoire est à la fois une question intrigante sur le plan conceptuel, et cruciale sur le plan pratique puisque de nombreux problèmes de cette famille se posent tous les jours dans les applications et que l'on espère pouvoir les résoudre effectivement : un grand nombre de ces problèmes sont réunis dans la classe **NP**.

Lorsqu'il est placé devant l'impossibilité de répondre à la question théorique posée et qu'il n'a pas à sa disposition des algorithmes exacts efficaces, l'utilisateur se tourne souvent vers des algorithmes probabilistes permettant en un temps polynomial d'obtenir une solution ayant une probabilité raisonable d'être optimale ou des algorithmes approchés qui donnent, avec une quasi certitude, une solution située à une faible distance de l'optimum.

Au long des neuf chapitres qui composent ce cours on s'intéresse d'abord à des méthodes qui donnent naissance à des algorithmes efficaces : la stratégie gloutonne et la programmation dynamique permettent ainsi de résoudre bon nombre de problèmes de façon élégante ; ces deux procédés font l'objet des premiers chapitres. On s'intéresse ensuite à des problèmes classiques d'algorithmique des graphes et réseaux. Plusieurs de ces problèmes, comme la recherche d'un flot optimal dans un réseau, sont des cas particuliers de programmation linéaire. Vu son importance, on consacrera un chapitre entier au problème de la programmation linéaire.
Les questions générales de complexité sont abordées sous l'angle de classification des problèmes pratiques : on illustrera sur des exemples des propriétés partagées par tous les systèmes de calcul et on accordera un peu de temps au problème fameux $\mathbf{P} \neq \mathbf{NP}$ On abordera rapidement le concept d'algorithme probabiliste, qui constitue l'un des plus riches développements du domaine.
Pour terminer on s'interessera à deux domaines d'applications des techniques mises en oeuvre : d'une part les algorithmes d'approximation pour l'optimisation combinratoire, et d'autre part l'analyse de chaînes de Markov de type Monte-Carlo qui sont utilisées en physique statistique.

Nous pensons que le lecteur aura ainsi développé un point de vue sur une problématique riche et utile, qui fait aujourd'hui l'objet de nombreux travaux de recherche et de développements industriels.

Terminologie de théorie des graphes

Un *graphe non-orienté* $G = (X, E)$ (on dit aussi symétrique) est constitué d'un ensemble (fini) X de *sommets* et d'un ensemble E *d'arêtes*, chaque arête étant une paire de sommets. On note en général n le nombre de sommets et m le nombre d'arêtes d'un graphe. Un *graphe orienté* est donné par un ensemble X de *sommets* et un ensemble A d'*arcs* ; chaque arc est un couple de sommets. On peut aussi considérer un graphe symétrique comme un graphe orienté (X, A) tel que pour tout arc $a \in A$, il existe un arc opposé $\bar{a}$ dont l'origine est l'extrémité de a et dont l'extrémité est l'origine de a. Le couple $\{a, \bar{a}\}$ forme alors une arête.

Un *chemin* dans un graphe non orienté est une suite d'arêtes $e_1, e_2, \ldots, e_k$ telle qu'il existe une suite de sommets $x_1, x_2, \ldots, x_k, x_{k+1}$ vérifiant $e_i = \{x_i, x_{i+1}\}$; on dit que le chemin *joint* x_1 à x_{k+1} ; k est la longueur du chemin. Le chemin est dit *élémentaire* si les $x_1, x_2, \ldots, x_k$ sont distincts. Un chemin est un *cycle* si $x_1 = x_{k+1}$ (on dit aussi parfois *circuit* mais nous reservons ce mot aux graphes orientés). Un graphe est *connexe* si pour tout couple de sommets $x, y \in X$, il existe un chemin qui joint x à y. Le *graphe complet* sur n sommets est le graphe tel que toute paire de sommets soit une arête.

Un *arbre* est un graphe non orienté connexe et sans cycle. Dans un arbre on a $m = n-1$; d'autre part pour tout couple de sommets il existe un chemin élémentaire unique qui les joint.

LEMME 0.1. *Les conditions suivantes sont équivalentes pour un graphe non-orienté* $G = (X, T)$ *:*

(1) $G = (X, T)$ *est connexe et sans cycle*

(2) $G = (X, T)$ *est connexe et* $X = |T| + 1$

(3) $G = (X, T)$ *est sans cycle et* $X = |T| + 1$

(4) $G = (X, T)$ *est tel que tout couple de sommets est relié par un chemin unique.*

Les arbres utilisés en structure de données sont légèrement différents de ceux considérés ici dans la mesure où un sommet appelé *racine* est distingué, ce qui induit la notion de parent et d'enfant qui est absente ici.

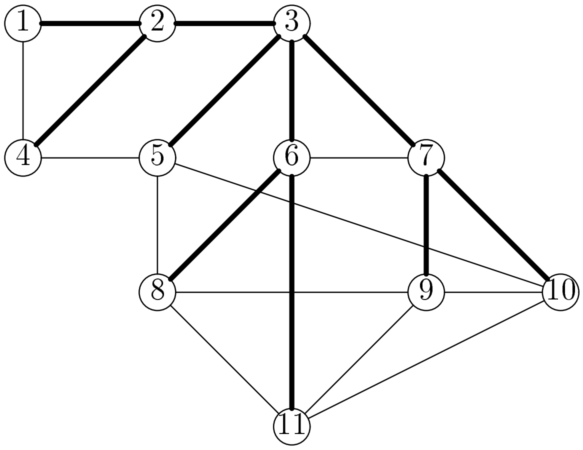


Fig. 1. Un graphe et un de ses arbres recouvrants en traits gras

Pour un graphe non-orienté $G = (X, E)$ quelconque, un *arbre recouvrant* est donné par un sous-ensemble de l'ensemble des arêtes qui forme un arbre ayant X pour ensemble de sommets (voir figure). G posséde un arbre recouvrant si et seulement s'il est connexe. Les algorithmes de parcours de graphes vus en Algorithmique et Programmation permettent de construire des arbres recouvrants. Le nombre d'arbres recouvrants d'un graphe est en général une fonction exponentielle de n.

Théorème 0.1. *Le graphe complet possède n^{n-2} arbres recouvrants.*

Si $G = (X, E)$ est un graphe symétrique et $A = (X, T)$ un arbre recouvrant de G, toute arête de G qui n'est pas dans T crée un cycle unique avec des arêtes de T, on l'appelle parfois *cycle principal*. Il y a $m - n + 1$ telles arêtes et donc $m - n + 1$ cycles principaux. On peut montrer que ces cycles permettent de construire tous les autres cycles du graphe par différence symétrique.

Plusieurs sous-ensembles de sommets ou d'arêtes d'un graphe symétrique jouent un rôle particulier. Un *ensemble indépendant* (on dit aussi parfois ensemble stable) est un sous-ensemble S de X tel que deux sommets de S ne sont pas joints par une arête. Soit formellement

$$(x \in S \text{ et } y \in S) \Rightarrow \{x, y\} \notin E.$$

Un ensemble *couvrant* Y de sommets est un sous-ensemble de X tel que toute arête possède au moins une extremité dans Y, soit :

$$\{x, y\} \in E \Rightarrow (x \in Y \text{ ou } y \in Y)$$

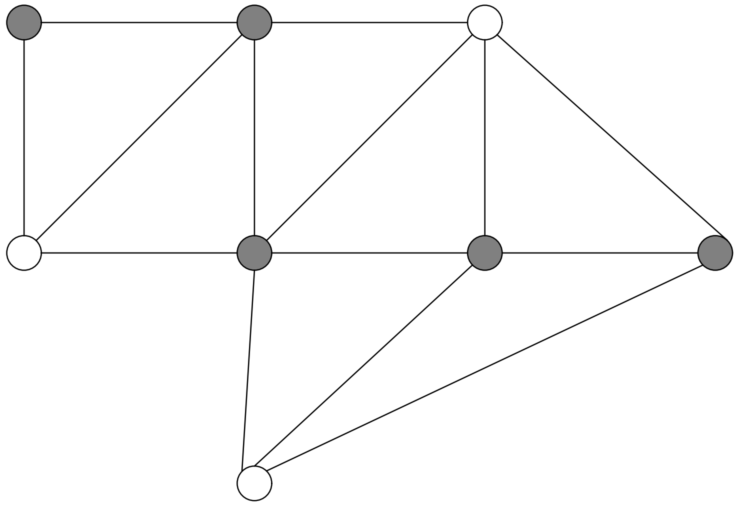


FIG. 2. Un des ensembles indépendants en clair et couvrants en plus foncé.

Un *couplage* est un sous-ensemble C de E tel que deux arêtes de C ne possèdent pas d'extrémité commune, soit :

$$e, f \in C \quad \Rightarrow \quad e \cap f = \emptyset$$

La notion de *coloration* joue un rôle important en théorie des graphes, il s'agit d'affecter des couleurs aux sommets de façon telle que deux sommets reliés par une arête aient des couleurs différentes. Formellement une k-coloration est une application c de X dans un ensemble $\{1, 2, \ldots, k\}$ de couleurs telle que :

$$\{x, y\} \in E \quad \Rightarrow \quad c(x) \neq (y).$$

Le plus petit entier k tel qu'il existe une k-coloration de G est appelé *nombre chromatique* de G.

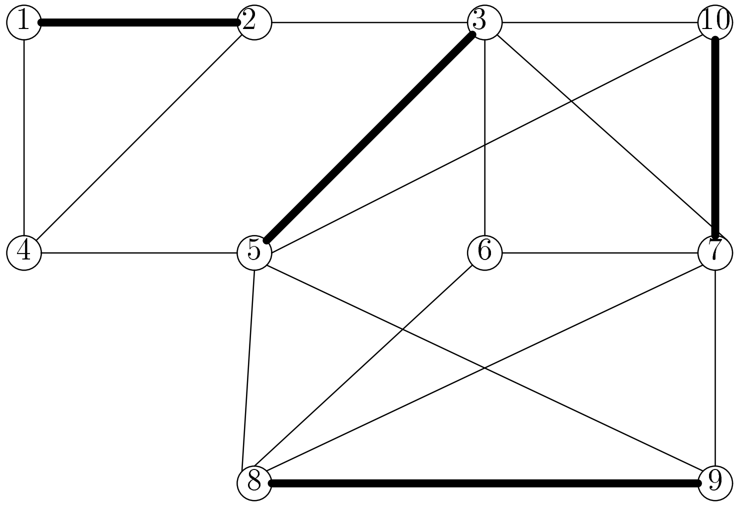


FIG. 3. Un graphe et un de ses couplage en traits gras

Probabilités discrètes

Etant donné une variable aléatoire X, deux paramètres donnent des informations particulièrement importantes sur X : sa moyenne et sa variance. La moyenne est linéaire : si $X = X_1 + \cdots + X_n$, alors $E(X) = E(X_1) + \cdots + E(X_n)$ (si ces moyennes existent, bien entendu). Ceci est vrai même si les variables aléatoires X_i ne sont pas indépendantes.

La variance de X est définie par $V(X) = E((X - E(X))^2) = E(X^2) - (E(X))^2$. Sa racine carrée est l'écart-type de X.

Si X et Y sont deux variables aléatoires indépendantes, alors $E(XY) = E(X)E(Y)$ et $V(X+Y) = V(X) + V(Y)$.

Si X est une variable aléatoire positive, alors l'inégalité de Markov dit que pour tout $t \geq 0$, on a

$$\Pr(X \geq t) \leq E(X)/t.$$

L'inégalité de Chebyshev en est un corollaire :

$$\Pr(|X - E(X)| \geq a) \leq V(X)/a^2.$$

Il est souvent nécessaire en analyse d'algorithmes de faire appel à des inégalités un peu plus sophistiquées, de type grande déviation. Voici l'une des plus simples et des plus couramment utilisées. Soient $X_1, X_2, \ldots X_n$ n variables aléatoires indépendantes à valeurs dans $\{0,1\}$, avec $\Pr(X_i = 1) = p_i$, et soit $X = X_1 + \cdots + X_n$. Soit $\mu = E(X) = \sum_i p_i$. Alors :

$$\Pr(X < (1-\delta)\mu) < e^{(-\mu\delta^2/2)}, \text{ et}$$
$$\Pr(X > (1+\delta)\mu) < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu,$$

pour $\delta \in (0,1]$.

Première partie

Algorithmes

CHAPITRE 1

Algorithmes gloutons et matroïdes

Les algorithmes de type glouton sont les plus simples de l'algorithmique. De nombreux problèmes d'optimisation combinatoire simples se présentent sous la forme suivante :

> On se donne un ensemble E fini et à chaque élément e de E est affectée une valeur $v(e)$ qui est un réel positif ; on se donne de plus une condition C sur l'ensemble des parties de E. Le problème consiste à construire un sous-ensemble F de E tel que :
>
> $\sum_{e \in F} v(e)$ est maximal parmi les parties F satisfaisant C.

Dans plusieurs cas la condition C est fermée par sous-ensemble : si F satisfait C alors il en est de même pour tout sous-ensemble de F. Il est alors naturel de proposer un algorithme très simple consistant à initialiser F par $F = \emptyset$, puis à ajouter successivement des éléments à F suivant un certain critère (par exemple par valeurs décroissantes), jusqu'à obtenir un ensemble auquel on ne peut plus ajouter d'élément sans contredire C. On appelle *algorithme glouton* ce type d'algorithme, ce qualificatif décrit le fait qu'on se précipite sur l'élément le plus intéressant en premier sans se préoccuper des conséquences ultérieures de ce choix.

Cet algorithme donne très rapidement un résultat, en revanche ce résultat n'est pas toujours la solution optimale. Nous allons voir que ce n'est le cas que pour des familles de parties satisfaisant une condition particulière : on appelle ces familles *matroïdes*. L'étude de la structure algébrique des matroïdes a fait l'objet de plusieurs travaux en combinatoire car elle donne un cadre mathématique unificateur à des résultats de théorie des graphes et d'algorithmique. Plusieurs ouvrages lui ont été consacrés, nous recommandons en particulier celui de D. Welsh [**21**] ; une bonne introduction au sujet est aussi donnée par le chapitre 17 du manuel [**4**].

Les questions relatives à l'affectation d'une ou plusieurs ressources à des utilisateurs (clients, processeurs, etc.) constituent une classe importante de problèmes qu'il est tentant de résoudre par un algorithme glouton. Nous allons montrer que dans certains cas très simples, l'algorithme glouton donne une solution optimale. Dans des cas plus complexes, l'algorithme donne une solution approchée, dont on se contente parfois, vu le temps de calcul prohibitif de la recherche de l'optimum exact. Après avoir traité ce cas simple nous considérons un problème classique sur les graphes : la recherche d'un arbre recouvrant de poids maximal (ou de manière équivalente minimal). La résolution de celui-ci par l'algorithme glouton amène à la structure de matroïde dont on donne quelques propriétés.

1. Stable maximum d'un graphe d'intervalles

Le problème décrit précisément ci-dessous peut être résolu par l'algorithme glouton (mais, comme on le verra, l'algorithme glouton ne donne pas la solution optimale pour d'autres formulations du problème, pourtant proches de celle-ci). Il s'agit d'affecter une ressource unique, non partageable, successivement à un certain nombre d'utilisateurs qui en font la demande en précisant la période exacte pendant laquelle ils souhaitent en disposer.

On peut illustrer ceci en considérant la question de la location d'une voiture. Des clients formulent un ensemble de demandes de location et, pour chaque demande sont donnés le jour du début de la location et le jour de restitution du véhicule ; le but est d'affecter le véhicule de façon à satisfaire le maximum de clients (et non pas de maximiser la somme des durées de location). On vérifie que ce problème rentre dans le cadre général considéré plus haut. L'ensemble E est celui des demandes de location, pour chaque élément e de E, on note $d(e)$ la date du début de la location et $f(e) > d(e)$ la date de fin. La valeur $v(e)$ de tout élément e de E est égale à 1 et la contrainte à respecter pour le sous-ensemble F à construire est la suivante, (cette contrainte exprime que deux clients ne peuvent disposer en même temps du véhicule) :

$$\forall e_1, e_2, \quad e_1 \in F \text{ et } e_2 \in F \Rightarrow [d(e_1), f(e_1)] \cap [d(e_2), f(e_2)] = \emptyset.$$

L'algorithme glouton est décrit comme suit :

– *Etape 1 :*

Classer les éléments de E par ordre de dates de fin croissantes. Les éléments de E forment alors une suite $e_1, e_2, \ldots, e_n$ telle que $f(e_1) \leq f(e_2) \leq \ldots \leq f(e_n)$

– *Etape 2 :*

Initialiser $F := \emptyset$

Pour i variant de 1 à n, ajouter la demande e_i à F si celle-ci ne chevauche pas la dernière demande appartenant à F.

THÉORÈME 1.1. *L'algorithme ci-dessus donne une solution optimale.*

Soit $F = \{x_1, x_2, \ldots, x_p\}$ la solution obtenue par l'algorithme glouton et soit $G = \{y_1, y_2, \ldots, y_q\}, q \geq p$ une solution optimale telle que le cardinal de $F \cap G$ soit maximum. Dans G et dans F nous supposons que les x_i et y_j sont classées par dates de fin croissantes. Nous allons montrer que $p = q$. Soit k le plus petit entier tel que $x_k \neq y_k$; ainsi $\forall i < k$, on a $x_i = y_i$. Par construction de F on a alors $f(y_k) \geq f(x_k)$. On peut alors remplacer G par $G' = \{y_1, y_2, \ldots y_{k-1}, x_k, y_{k+1}, y_q\}$ tout en satisfaisant à la contrainte de non chevauchement des demandes, ainsi G' est une solution optimale, de même cardinalité que G, ayant plus d'éléments en commun avec F que n'en avait G. En répétant cette opération suffisamment de fois on trouve un ensemble H de même cardinalité que G et qui contient F. L'ensemble H ne peut contenir plus d' éléments que F car ceux-ci auraient été ajoutés à F par l'algorithme glouton, ceci montre bien que $p = q$.

Remarques

1. Noter que le choix de classer les demandes par dates de fin croissantes est important. Si on les avait classées, par exemple, par dates de début croissantes, on n'aurait pas obtenu le résultat. On le voit sur l'exemple suivant avec trois demandes e_1, e_2, e_3 dont les dates de début et de fin sont données par le tableau suivant :

	e_1	e_2	e_3
d	2	3	5
f	8	4	8

Bien entendu, pour des raisons évidentes de symétrie, le classement par dates de début décroissantes donne aussi un résultat optimal.

2. On peut noter aussi que si le but est de maximiser la durée totale de location du véhicule l'algorithme décrit ci-dessus ne donne pas l'optimum. En particulier, il ne considérera pas comme prioritaire une demande de location de durée très importante. Une idée consiste à le modifier en classant les demandes par durées décroissantes et d'appliquer l'algorithme glouton, malheureusement cette technique ne donne pas non plus le bon résultat (il suffit de considérer une demande de location de 3 jours et deux demandes qui ne se chevauchent pas mais qui sont incompatibles avec la première chacune de durée égale à 2 jours).

3. S'il y a plus d'une ressource à affecter, par exemple deux voitures à louer, l'algorithme glouton consistant à classer les demandes suivant les dates de fin et à affecter la première ressource disponible, ne donne pas l'optimum, comme le montre l'exemple suivant :

	e_1	e_2	e_3	e_4
d	1	2	6	4
f	3	5	7	8

En effet les deux premières demandes sont satisfaites par des véhicules différents si on attribue le premier véhicule à la troisième demande, il n'est plus possible de satisfaire la quatrième.
4. Voici la justification du titre de cette section. Un graphe d'intervalles est un graphe dont chaque sommet correspond à un intervalle, et tel que deux sommets sont joints par une arête si et seulement si les deux intervalles ont une intersection non vide. Les demandes de location sont des intervalles, les contraintes sont représentées par le graphe d'intervalles associé, et une solution optimale correspond à un stable maximum de ce graphe.

2. Arbre de poids maximal d'un graphe

2.1. Exposé du problème. Nous renvoyons aux préliminaires sur les graphes pour la terminologie utilisée ici.

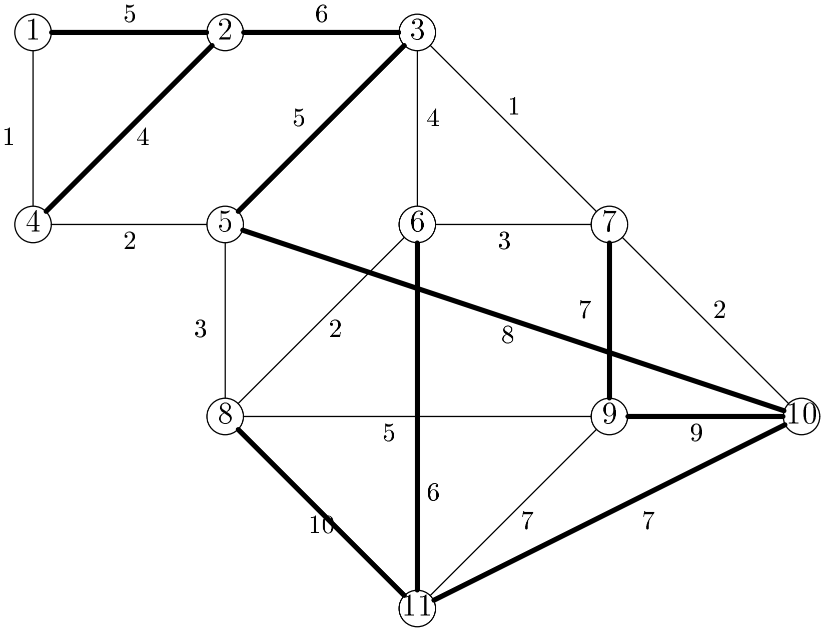


FIG. 1. Un arbre recouvrant de poids maximum

Un exemple classique d'utilisation de l'algorithme glouton est la recherche d'un arbre recouvrant de poids maximal dans un graphe symétrique, il prend dans ce cas particulier le nom d'algorithme de Kruskal. Ce problème se rencontre de fait en pratique sous la forme de la recherche d'un arbre de coût minimal pour réaliser une connexion de tous les points d'un réseau ; nous avons choisi de conserver une cohérence générale à ce chapitre en présentant tous les problèmes sous forme de recherche d'un objet qui rend maximal une certaine fonction, ceci n'a toutefois pas une grande importance car il est facile dans ce cas de transformer une recherche d'un poids maximal en celle d'un coût minimal par une transformation simple. Décrivons brièvement le problème et l'algorithme.

On affecte aux arêtes d'un graphe symétrique un poids qui est un entier positif, le poids d'un arbre est égal à la somme des poids des arêtes qui le composent ; on cherche un arbre recouvrant de poids maximal. Une illustration de ce problème est la réalisation d'un réseau électrique ou informatique entre différents points, deux points quelconques doivent toujours être reliés entre eux (connexité) et on doit minimiser le coût de la réalisation. Le poids d'une arête est, dans ce contexte, la différence entre un entier constant assez grand et le coût de construction de la portion du réseau reliant ses deux extrémités. L'arbre de poids maximal va ainsi donner le réseau dont le coût de construction est minimal.

On peut facilement formuler le problème dans le cadre général donné en début de chapitre : E est l'ensemble des arêtes du graphe, la condition C à satisfaire par F est de former un graphe sans cycle, enfin il faut maximiser la somme des poids des éléments de F.

Un exemple d'arbre recouvrant de poids maximal est donné sur la figure 1.

2.2. Algorithme de Kruskal. Ce problème peut être résolu très efficacement par l'algorithme glouton suivant :

– *Etape 1 :*

Classer les arêtes par ordre de poids décroissants. Elles constituent alors une suite

$$e_1, e_2, \ldots, e_n$$

telle que

$$v(e_1) \geq v(e_2) \geq \ldots \geq v(e_n)$$

– *Etape 2 :*

Initialiser $F := \emptyset$

Pour i variant de 1 à n, ajouter l'arête e_i à F si celle-ci ne crée pas de cycle avec celles appartenant à F.

On montre que l'algorithme glouton donne l'arbre de poids maximal en utilisant la propriété suivante des arbres recouvrants d'un graphe :

LEMME 1.1. *Soient T et U deux arbres recouvrants d'un graphe G et soit a une arête de U qui n'est pas dans T. Alors il existe une arête b du cycle γ créé par a dans T telle que $U \setminus \{a\} \cup \{b\}$ soit aussi un arbre recouvrant de G.*

Preuve : La suppression de a dans l'arbre U divise cet arbre en deux composantes connexes U_1 et U_2. Le cycle γ créé par a dans T contient nécessairement un nombre pair d'arêtes reliant ces deux composantes, comme a est une telle arête il en existe une autre dans T. On note b cette arête et on vérifie qu'elle satisfait bien aux conditions du lemme.

Preuve de validité de l'algorithme : Soit T l'arbre donné par l'algorithme glouton. Si l'algorithme ne donne pas l'optimum c'est qu'il existe des arbres recouvrants de poids supérieurs à celui de T. Notons U celui parmi ceux-ci dont le nombre d'arêtes communes avec T est maximum. Soit a l'arête de U de plus grand poids qui n'est pas dans T, et soit b l'arête de T donnée par le lemme. Comme a n'a pas été choisie par l'algorithme glouton c'est qu'elle crée un cycle avec les éléments de T, formé d'arêtes toutes de poids supérieur ou égal à $v(a)$. Puisque b est dans ce cycle on a $v(b) \geq v(a)$, ainsi $v(U - \{a\} \cup \{b\}) \geq v(U) > v(T)$, mais $U - \{a\} \cup \{b\}$ est un arbre qui a une arête en plus en commun avec T que n'en a U ce qui contredit le choix de U. □

3. Condition d'optimalité de l'algorithme glouton

Les deux problèmes considérés ci-dessus peuvent être formulés dans un cadre général. On se donne un ensemble E fini et une famille non vide $\mathcal{F}$ de parties de E qui soit fermée par sous-ensemble :

$$F \in \mathcal{F} \text{ et } F' \subset F \Rightarrow F' \in \mathcal{F}$$

On se donne aussi une application v de E dans l'ensemble des entiers naturels. On cherche $F \in \mathcal{F}$ tel que $v(F) = \sum_{e \in F} v(e)$ soit maximal

L'algorithme glouton s'exprime alors par :

- *Etape 1 :* Classer les éléments de E par ordre des $v(e)$
  décroissants. Ils constituent alors une suite $e_1, e_2, \ldots e_n$, telle que :

$$v(e_1) \geq v(e_2), \ldots \geq v(e_n)$$

- Initialiser $F := \emptyset$
- *Etape 2 :* Pour i variant de 1 à n, ajouter l'élément e_i à F si $F \cup \{e_i\} \in \mathcal{F}$.

On a vu que l'algorithme glouton donne le résultat optimal quelles que soient les valeurs des $v(e)$ pour l'arbre recouvrant, par contre pour celui de l'affectation de ressources on n'obtient l'optimum que lorsque la fonction v est constante ($\forall e, v(e) = 1$). Il est ainsi naturel de rechercher un critère sur la famille $\mathcal{F}$ qui permette de déterminer les cas où l'algorithme glouton donne la solution optimale quelles que soient les valeurs des $v(e)$. On obtient alors le résultat suivant :

THÉORÈME 1.2. *Si l'algorithme glouton donne une solution optimale quelle que soit la fonction v alors la famille $\mathcal{F}$ vérifie la propriété suivante :*

Si F et G sont deux ensembles de $\mathcal{F}$ tels que G a un élément de plus que F, alors il existe un élément x de G qui n'est pas dans F et tel que $F \cup \{x\} \in \mathcal{F}$.

Preuve : Pour démontrer ce résultat on suppose donnés F et G et on construit une fonction de poids v telle que l'algorithme glouton donne une solution qui contient strictement F.

Notons p le nombre d'éléments communs à F et G et q le nombre d'éléments de F qui ne sont pas dans G. On définit la fonction v suivante :

- Pour tout élément f de F, $v(f) = 2q + 2$
- Pour tout élément g de G qui n'est pas dans F, $v(g) = 2q + 1$
- Pour tout élément h qui n'est ni dans F ni dans G, $v(h) = 0$

L'algorithme glouton appliqué pour ces valeurs des poids donne un ensemble F' qui contient nécessairement F puisque les éléments de F se présentent en premier. Or on a $v(F) = (p + q)(2q + 2)$ et $v(G) = p(2q + 2) + (q + 1)(2q + 1)$ Ainsi $v(G) = v(F) + q + 1$, comme l'algorithme glouton donne un ensemble F' ayant un poids maximal le poids de F' doit être supérieur ou égal à celui de G donc strictement supérieur à celui de F. Ainsi F' contient un élément qui n'est pas dans F et qui a un poids non nul, c'est donc un élément de G qui est le x cherché. □

En fait cette condition est aussi suffisante comme le montre la réciproque ci-dessous.

Réciproque Si $\mathcal{F}$ est une famille de sous-ensembles d'un ensemble E fini, satisfaisant les 2 conditions suivantes :

- $F \in \mathcal{F}, F' \subset F, \ \Rightarrow F' \in \mathcal{F}$
- Si F et G sont deux ensembles de $\mathcal{F}$ tels que G contient plus d'éléments que F alors il existe un élément x de $G \setminus F$ tel que $F \cup \{x\} \in \mathcal{F}$.

Alors l'algorithme glouton appliqué à la famille $\mathcal{F}$ donne l'optimum.

Preuve : Supposons que l'algorithme glouton construise l'ensemble F et notons G un des ensembles de la famille $\mathcal{F}$.

On montre, par récurrence sur le nombre p d'éléments de G qui ne sont pas dans F, que

$$v(F) \geq v(G)$$

Si $p = 0$ c'est immédiat. Soit $p > 0$, il existe un $x \in F$ tel que $x \notin G$, sinon $F \subset G$ et l'algorithme glouton aurait sélectionné les éléments de G lorsqu'ils se seraient présentés.

Ainsi soit x le premier élément sélectionné par l'algorithme glouton qui n'est pas dans G. Les éléments y de G qui ne sont pas dans F sont tels que $v(y) \leq v(x)$, sinon ils auraient été retenus par l'algorithme glouton. Soit l'ensemble $F' = (F \cap G) \cup \{x\}$, comme c'est un sous-ensemble de F, il appartient à $\mathcal{F}$. La deuxième condition ci-dessus implique que l'on peut y ajouter successivement des éléments de G pour obtenir un ensemble G' de même cardinal que G et appartenant à $\mathcal{F}$. Comme x est le seul élément de G' qui n'est pas dans G, on a alors $G' = G \cup \{x\} \setminus \{y\}$ et :

$$v(G') = v(G) + v(x) - v(y) \geq v(G).$$

De plus G' a un élément de plus en commun avec F que n'en a G, l'hypothèse de récurrence implique donc $v(F) \geq v(G') \geq v(G)$. □

4. Matroïdes : définition et exemples

Les propriétés figurant dans le théorème ci-dessus et sa réciproque suggèrent l'introduction de la notion de matroïde qui est une structure définie par une famille de sous-ensembles d'un ensemble E, famille qui vérifie certaines conditions. Cette démarche est assez courante en mathématiques, citons par exemple la topologie définie par des ouverts, la théorie de la mesure et ses ensembles mesurables, les espaces vectoriels et leurs ensembles indépendants de vecteurs. Dans le cadre de la combinatoire, principalement concernée par des structures mathématiques *finies*, on a développé ces dernières années des outils unificateurs qui tendent à présenter sous une forme générale des résultats qui paraissent dans un premier abord très épars. Une de ces structures est celle de matroïde, il n'est pas étonnant de voir apparaître le nombre d'éléments d'un sous-ensemble dans la définition puisque la combinatoire s'intéresse aux ensembles finis.

Définition Un matroïde est un couple $\mathcal{M} = (E, \mathcal{F})$ où E est un ensemble fini et où $\mathcal{F}$ est une famille de parties (sous-ensembles) de E satisfaisant les conditions suivantes (on note dans la suite $|F|$ le nombre d'éléments de l'ensemble F) :

(I1) $\mathcal{F}$ contient au moins une partie de E

(I2) $F' \subset F$ et $F \in \mathcal{F} \Rightarrow F' \in \mathcal{F}$

(I3) $F, G \in \mathcal{F}, |G| > |F| \Rightarrow \exists x \in G \setminus F, \quad F \cup \{x\} \in \mathcal{F}$

Les éléments d'une telle famille $\mathcal{F}$, sont appelés ensembles *indépendants* du matroïde $\mathcal{M}$.

Les matroïdes ont fait l'objet d'études mathématiques importantes sans référence à la raison pour laquelle ils ont été introduits dans ce cours. Pour ce qui concerne l'algorithmique il est intéressant de reconnaître si un problème de recherche d'un optimum rentre dans le cadre de la recherche d'un ensemble indépendant maximal d'un matroïde, auquel cas il existe un algorithme simple permettant de trouver cet optimum.

Nous nous proposons de donner quelques exemples de matroïdes :

4.1. Familles de vecteurs indépendants dans un espace vectoriel. Soit E un espace vectoriel sur un corps K, et soit $\mathcal{F}$ la famille des ensembles de vecteurs linéairement indépendants de E. Il est facile de vérifier que cette famille satisfait les conditions (I1), (I2) et (I3), pour les deux premières c'est immédiat. Vérifions la dernière; soit $F = \{u_1, u_2, \ldots, u_k\}$ et $G = \{v_1, v_2, \ldots, v_k, v_{k+1}\}$ deux ensembles chacun formé de vecteurs indépendants. Si chaque v_i dépendait linéairement des u_j l'ensemble G serait de dimension k ce qui contredirait le fait qu'il est formé de $k+1$ vecteurs indépendants, il existe donc un v_i qui ne dépend pas des u_j, et $F \cup \{v_i\} \in \mathcal{F}$.

4.2. Familles des parties sans cycle dans un graphe. Soit $\Gamma = (X, E)$ un graphe non-orienté, on définit un matroïde formé par des sous-ensembles de l'ensemble E des arêtes. Un ensemble indépendant est un sous-ensemble de E qui ne contient pas de cycle. De manière équivalente on peut définir un ensemble indépendant comme une union d'arbres $F = \cup F_i$ $(i = 1, \ldots, p)$ dont les ensembles de sommets X_i sont disjoints et dont l'union est égale à X. Les conditions I1 et I2 sont faciles à vérifier par contre I3 est un peu plus complexe. Si G et F sont deux parties indépendantes de E telles que G possède une arête de plus que $F = \cup F_i$ on divise G en $G = \cup G_i \cup G'$ où G_i est constitué des arêtes de G ayant leurs deux extrémités dans X_i et G' contient les arêtes qui ont des extrémités dans deux X_i distincts. Pour démontrer I3, il suffit de vérifier que G' n'est pas vide. Or on remarque que G_i ne peut pas contenir plus d'éléments que F_i (sinon il y aurait un cycle dans X_i, G_i). Comme $|G| = |F| + 1$ et $|G| = \sum |G_i| + |G'|$ on a $|G'| > 0$ et le résultat.

4.3. Graphes bipartis. Un graphe $G = (X, E)$ est dit *biparti* si l'ensemble X des sommets est réunion de deux sous-ensembles disjoints Y et Z tels que toute arête $e \in E$ a une extrémité dans Y et l'autre dans Z. On définit un matroïde en prenant comme ensembles indépendants les sous-ensembles F de E tels que deux arêtes de F ne sont pas incidentes à un même sommet appartenant à Y. On peut par symétrie définir un autre matroïde en remplaçant Y par Z.

Un *couplage* dans un graphe biparti est un sous-ensemble M de l'ensemble des arêtes tels que tout couple d'arêtes de M n'a pas de sommet commun. On vérifie que l'ensemble des couplages ne forme pas un matroïde, par contre un *couplage* est un élément qui appartient à la fois aux deux matroïdes définis ci-dessus, ainsi l'ensemble des couplages est l'intersection de deux matroïdes.

4.4. Transversal d'une famille d'ensembles. Soit $\mathcal{A} = A_1, A_2, \ldots, A_k$ une famille de sous-ensembles de E, les *transversaux* de $\mathcal{A}$ sont les parties de $T = \{x_1, x_2 \ldots, x_l\}$ de E telles qu'il existe des entiers distincts $i_1, i_2, \ldots, i_l$ satisfaisant $x_j \in A_{i_j}$.

Un exemple simple est le suivant : $E = \{1, 2, 3, 4, 5, 6\}$,

$$A_1 = \{1, 3, 4\},\ A_2 = \{3, 5, 6\},\ A_3 = \{1, 4\},\ A_4 = \{2, 5\}, A_5 = \{3, 4\}, A_6 = \{1, 3\}$$

On vérifie alors facilement que tout sous-ensemble de E qui ne contient pas à la fois $2, 5$ et 6 est un transversal.

Un exemple de transversal est donné par la constitution d'un ensemble de représentants s pour différentes associations dans une population. Ainsi si E est l'ensemble des habitants d'un village, soit A_1 l'ensemble des parents d'élèves, A_2 l'ensemble des pécheurs à la ligne, A_3 les membres du cinéclub, A_4 les joueurs de pétanque etc. On peut alors interpréter un transversal comme un ensemble d'individus tel que chacun d'entre eux représente une association : une même personne ne peut représenter deux groupes, par contre, il se peut que certains groupes ne soient pas représentés.

La détermination d'un représentant pour chacune des associations de façon telle qu'un nombre maximum d'entre elles soient représentées relève ainsi de la recherche d'un transversal de taille maximale.

On peut vérifier les axiomes de matroïdes pour ces ensembles transversaux mais ce n'est pas immédiat, ceci pourra être fait au chapitre 3 après l'étude des couplages.

4.5. Réalisation de taches sur un processeur séquentiel. Voici un exemple de matroïde issu d'une situation pratique. Des tâches $T_1, T_2, \ldots, T_k$, sont à réaliser sur un processeur séquentiel, la réalisation de chacune de celles-ci prend une unité de temps et on donne pour chaque T_i un délai d_i à respecter (en supposant que l'on commence à l'instant 0, la tâche T_i doit être réalisée avant l'instant d_i). De plus une pénalité p_i est fixée pour le cas où la tache T_i ne serait pas réalisée dans les délais impartis (cette pénalité est fixe et ne dépend pas de l'instant de la réalisation). Il s'agit de déterminer l'ordre sur les tâches qui minimise la somme des pénalités. L'exemple suivant illustre le problème : il y a 7 tâches $T_1, \ldots, T_7$ dont les délais et pénalités sont données par le tableau suivant :

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
d	4	3	2	2	6	3	5
p	70	80	90	50	30	40	20

Si on effectue les tâches dans l'ordre donné par T_1, T_2, T_3, T_4, T_5, T_6, T_7 les tâches T_3, T_4, T_6, T_7 ne sont pas effectuées dans les délais et la pénalité est de $90+50+40+20 = 200$. Pour l'ordre : T_3, T_4, T_2, T_5, T_7, T_1, T_6 seules les tâches T_1 et T_6 sont hors délais et la pénalité est 110, noter que l'optimum est atteint pour T_4, T_3, T_2, T_1, T_7, T_5, T_6 avec une pénalité de 40.

On considère la famille $\mathcal{F}$ de sous-ensembles F de $T = \{T_1, T_2, \ldots, T_k\}$ donnée par $F \in \mathcal{F}$ si les tâches contenues dans F peuvent être toutes réalisées dans les délais. On montre que cette famille satisfait les conditions sur les matroïdes. (Laissé en exercice, considérer les nombres $U(i, F)$ de tâches de délai inférieur ou égal à i). Ainsi l'algorithme glouton consistant à ajouter les tâches par ordre de pénalités décroissantes donne la solution optimale au problème posé.

5. Définitions équivalentes des matroïdes

Nous avons défini les matroïdes par des ensembles indépendants mais il y a d'autres ensembles remarquables dans un matroïde : les bases et les circuits. Ceux-ci peuvent être définis à partir des ensembles indépendants mais on peut aussi inversement retrouver les ensembles indépendants à l'aide des bases ou à l'aide des circuits. Une notion intéressante est aussi celle de rang d'un sous-ensemble qui associe un entier à tout sous-ensemble de E. L'équivalence entre ces différents concepts peut alors être utilisée pour vérifier qu'une famille constitue bien un matroïde, il est en effet parfois plus simple de retrouver une famille d'ensembles indépendants, d'autres fois une famille de bases, ou de circuits. Les caractérisations sont données par 3 théorèmes dont les preuves techniques sont omises.

Soit $\mathcal{M} = (E, \mathcal{F})$ un matroïde, on appelle base de $\mathcal{M}$ une partie maximale de $\mathcal{F}$. Ainsi la famille $\mathcal{B}$ des bases de $\mathcal{M}$ est composée des ensembles B tels que

$$B \in \mathcal{F}, B \subset F, F \in \mathcal{F} \quad \Rightarrow B = F$$

On vérifie le théorème suivant :

Théorème 1.3. *La famille $\mathcal{B}$ des bases d'un matroïde $\mathcal{M}$ satisfait la condition :*

$$(Ba) \quad B_1, B_2 \in \mathcal{B}, x \in B_1 \setminus B_2, \Rightarrow \exists y \in B_2 \setminus B_1 \quad \text{tel que} \quad (B_1 \cup y) \setminus x \in \mathcal{B}$$

Réciproquement Si $\mathcal{B}$ satisfait la condition (Ba), alors la famille $\mathcal{F}$ des parties F de E telles qu'il existe $B \in \mathcal{B}$ satisfaisant $F \subset B$ vérifie les conditions $(I1), (I2), (I3)$.

La notion de *circuit* joue aussi un rôle important dans cette théorie. Soit $\mathcal{F}$ une famille de parties indépendantes de $\mathcal{M}$, on appelle circuit de $\mathcal{M}$ une partie minimale qui n'appartient pas à $\mathcal{F}$, ainsi la famille $\mathcal{C}$ des circuits de $\mathcal{M}$ est composée des ensembles C tels que

$$C \notin \mathcal{F}, \quad \text{et} \qquad F \subset C, F \neq C \;\Rightarrow F \in \mathcal{F}$$

On vérifie alors la caractérisation suivante :

Théorème 1.4. *La famille $\mathcal{C}$ des circuits d'un matroïde $\mathcal{M}$ vérifie les conditions :*
- *Si C_1 et C_2 sont deux ensembles distincts de $\mathcal{C}$ alors C_1 n'est pas inclus dans C_2.*
- *Si C_1 et C_2 sont deux ensembles distincts de $\mathcal{C}$ et si $x \in C_1 \cap C_2$, alors il existe C_3 tel que $C_3 \subset (C_1 \cup C_2) \setminus x$.*

Réciproquement, soit $\mathcal{C}$ une famille de parties de E satisfaisant les deux conditions ci-dessus, on construit une famille $\mathcal{F}$ d'ensembles indépendants. Un ensemble est indépendant si et seulement s'il ne contient pas de circuit. On vérifie sans trop de peine que $\mathcal{F}$ satisfait les conditions I1, I2 et I3.

La notion de *rang* permet, lorsqu'on a défini un matroïde $\mathcal{M}$ sur E, d'associer un entier à toute partie de l'ensemble de base E. Soit $\mathcal{F}$ une famille de parties indépendantes définissant $\mathcal{M}$, la fonction rang, notée ρ de $\mathcal{M}$ est telle que pour tout A, $A \subset E$, $\rho(A)$ est le cardinal maximal d'un ensemble F de $\mathcal{F}$ inclus dans A, soit :

$$\rho(A) \;=\; \max_{F \subset A, F \in \mathcal{F}} |F|$$

On a alors le théorème suivant :

THÉORÈME 1.5. *Une application ρ de l'ensemble des parties de E dans l'ensemble $\mathcal{N}$ des entiers naturels est la fonction rang d'un matroïde $\mathcal{M}$ si et seulement si :*
- $\rho(\emptyset) = 0$,
- $A \subset B \quad \Rightarrow \quad \rho(A) \leq \rho(B)$,
- $\rho(A \cup \{x\}) = \rho(A \cup \{y\}) = \rho(A) \quad \Rightarrow \quad \rho(A \cup \{x, y\}) = \rho(A)$.

Dans les exemples donnés plus haut on retrouve ces différentes notions :

Pour les ensembles indépendants de vecteurs, les bases correspondent à la notion habituelle de base d'un sous-espace vectoriel, la fonction rang correspond à la dimension, enfin les circuits sont les ensembles de vecteurs dépendants tels que la suppression de l'un quelconque d'entre eux donne une famille indépendante.

Pour les graphes connexes, on vérifie facilement que les cycles élémentaires de G satisfont les conditions sur les circuits d'un matroïde. Les bases sont alors les arbres recouvrants, et le rang d'un sous-ensemble A de l'ensemble E des arêtes est égal à $n - c$. Ici n est le nombre total de sommets du graphe, c est le nombre de composantes connexes du graphe ayant X comme ensemble de sommets et A comme ensemble d'arêtes un sommet sans arête dans A compte pour une composante connexe).

6. Jeu de Shannon

6.1. Présentation. Le jeu de Shannon illustre l'importance de la notion d'arbre recouvrant sur un graphe, et donc celle de base sur un matroïde. On voit aussi apparaître les opérations de suppression et de contraction d'arête qui jouent un rôle crucial en théorie algébrique des graphes et qui ont aussi une généralisation intéressante dans le cadre des matroïdes. Les règles de ce jeu sont les suivantes :

Soit G un graphe symétrique donné par un ensemble X de sommets, où deux sommets u et v sont distingués, et un ensemble E d'arêtes. Un jeu de Shannon sur $G = (X, E, u, v)$ se pratique à deux joueurs. Au départ toute les arêtes sont incolores ; chaque joueur à tour de rôle colorie une arête incolore, l'un *(le conquérant)* en vert, l'autre *(le défenseur)* en rouge. Le conquérant a gagné s'il a réussi à construire un chemin composé d'arêtes vertes reliant u à v ; par contre si toutes les arêtes sont coloriées sans qu'un tel chemin existe c'est le défenseur qui est déclaré gagnant. La victoire du défenseur peut se constater avant la fin du jeu s'il a réussi à disconnecter le graphe par des arêtes rouges, en deux composantes dont l'une contient u et l'autre v.

6.2. Classification. On classe les graphes où deux sommets sont distingués en trois catégories :
- Les graphes *faibles* pour lesquels il existe une stratégie du conquérant qui lui permet de gagner contre toute défense et ceci quel que soit le joueur qui joue en premier.
- Les graphes *résistants* pour lesquels il existe une stratégie permettant au résistant de vaincre et ceci même dans le cas où le conquérant joue en premier.

– Les graphes *neutres* pour lesquels il existe une stratégie gagnante pour tout joueur qui a le trait en premier.

À tout graphe $G = (X, E, u, v)$ on associe le graphe $G_1 = (X, E', u, v)$ où $E' = E \cup \{u, v\}$; on vérifie alors que G est résistant si et seulement si G_1 est neutre et que G est neutre si et seulement si G_1 est faible et G ne l'est pas. Ainsi un algorithme qui permet de vérifier si un graphe est faible peut être utilisé sur G_1 pour tester si G est résistant ou neutre.

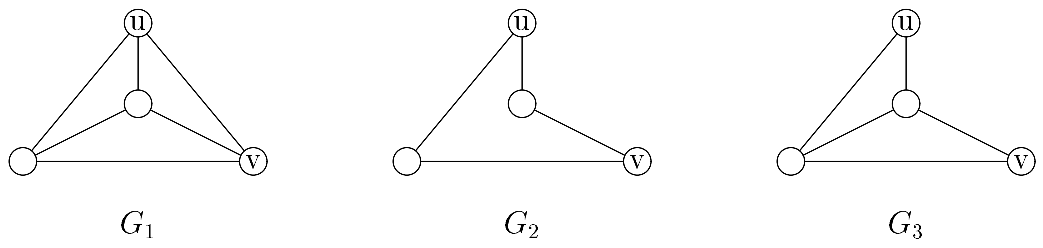


FIG. 2. Trois graphes, G_1 faible, G_2 résistant, G_3 neutre

6.3. Caractérisation des situations gagnantes. On a le résultat suivant dû à A. Lehman :

THÉORÈME 1.6. *Le graphe G avec les sommets u et v est faible si et seulement s'il existe deux arbres $A_1 = (Y, T_1)$, $A_2 = (Y, T_2)$ tels que :*

$$Y \subset X, \quad T_1 \subset E, \quad T_2 \subset E, \quad T_1 \cap T_2 = \emptyset, \quad u, v \in Y$$

On ne démontre pas ce théorème complètement ; on se contente de donner une stratégie pour le conquérant, dans un graphe qui satisfait les conditions ci-dessus, lorsqu'il joue en second. L'idée générale est que les arêtes coloriées en rouge par le défenseur sont supprimées du graphe alors que celles coloriées en vert sont contractées. On donne le sens suivant à la contraction :

Soit $G = (X, E)$ un graphe symétrique et $e = \{x, y\}$ une arête de G, le contracté G_e de G suivant e est le graphe obtenu en supprimant le sommet y et remplaçant chaque arête $\{y, z\}$ dont l'une des extrémités est y par une arête $\{x, z\}$, une telle opération peut amener à construire des arêtes multiples.

La stratégie de l'attaquant consiste lorsque le défenseur supprime une arête qui disconnecte un des arbres T_1 ou T_2 à choisir une arête e dans l'autre arbre de telle façon que que G_e contienne deux arbres disjoints pour les arêtes ayant même ensemble de sommets, ce qui se traduit par l'algorithme suivant :

Si l'arête rendue rouge par le défenseur n'appartient à aucun des deux arbres T_1 ou T_2, colorier en vert n'importe quelle arête.

Si l'arête rendue rouge appartient à T_1, on considère les deux parties connexes Y_1 et Y_2 de T_1 ainsi créées, on choisit une arête f de T_2 qui relie un sommet de Y_1 à un sommet de Y_2, et l'on colorie$f$ en vert.

Si l'arête rendue rouge appartient à T_2, on effectue un choix similaire en échangeant T_1 et T_2.

On poursuit alors le jeu dans le graphe obtenu en supprimant l'arête rouge et en contractant l'arête verte, ce graphe possède encore deux arbres qui satisfont les mêmes conditions que T_1 et T_2, le jeu se termine lorsque les deux sommets u et v sont reliés par une arête, lorsque c'est le tour du conquérant de jouer.

Ce jeu sur des graphes se généralise sur des matroïdes et on montre un résultat du même type car la preuve qui est faite sur les graphes utilise des propriétés des arbres recouvrants qui sont celles d'un matroïde.

CHAPITRE 2

Plus courts chemins, programmation dynamique

Dans ce chapitre on s'intéresse à la recherche de chemins de longueur minimale dans des graphes orientés ou symétriques en supposant que chaque arc (ou chaque arête) est muni d'une longueur. On suppose connus les algorithmes sur les parcours de graphes qui construisent des chemins entre les sommets sans la contrainte de minimalité, ceux-ci figurent dans le cours d'algorithme et programmation du tronc commun.

On verra que la recherche de chemins minimaux partant d'un sommet distingué donné fait appel à des techniques du style de l'algorithme glouton. Par contre si l'on recherche l'ensemble des chemins minimaux entre tous les couples de sommets une nouvelle technique est mise en œuvre. Celle-ci se rencontre dans des problèmes de nature différente on l'appelle pour des raisons historiques *programmation dynamique*, le terme programmation est employé ici dans le sens de méthode et non pas dans celui de réalisation d'un programme informatique.

La recherche de plus courts chemins peut être vue aussi comme la résolution d'un système d'équations linéaires dans un anneau particulier pour lequel cette résolution se limite au calcul de la puissance d'une matrice. Une introduction à cette théorie qui est largement développée par ailleurs est donnée.

La méthode de la programmation dynamique est expliquée dans ses grandes lignes au paragraphe suivant, elle est appliquée ensuite pour résoudre plusieurs problèmes d'optimisation dont l'un intervient dans le cadre de l'analyse du génome.

1. Plus courts chemins d'origine donnée

Dans la suite, on considère un graphe orienté $G = (X, A)$ ayant X comme ensemble de sommets et A comme ensemble d'arcs. Les applications *or* et ext associent à tout arc a de A son origine et son extrémité respectivement. On se donne une application ℓ de A dans l'ensemble des entiers naturels, $\ell(a)$ est la *longueur* de l'arc a. La longueur d'un chemin est égale à la somme des longueurs des arcs qui le composent. Étant donné un sommet x_0, le problème consiste à déterminer pour chaque sommet $x_i \in X$, le plus court chemin, s'il existe, qui joint x_0 à x_i. Par exemple dans le graphe donné ci-dessous les plus courts chemins à partir du sommet x_2 sont dessinés en traits gras.

Le caractère très simple des algorithmes de recherche de plus courts chemins partant de x_0 repose sur la propriété suivante : si un chemin f de longueur minimale joint x_0 à x tout facteur gauche de ce chemin se terminant par l'arc a est un chemin minimal qui joint x_0 à $ext(a)$.

On utilise dans les deux algorithmes donnés ci-dessous une application d de l'ensemble des sommets dans l'ensemble des entiers. L'image par d du sommet x est notée $d[x]$ car dans la réalisation d'un programme celle-ci sera représentée par un tableau. D'autre part la représentation des plus courts chemins s'effectue par un vecteur `pere` tel que `pere[x]` est le sommet situé immédiatement avant `x` dans un chemin de longueur minimale qui joint x_0 à x. Une opération élémentaire utilisée pour modifier d est l'opération dite de *relaxation*, elle est décrite par l'algorithme suivant :

```
 Relaxation (Arc a) {
      si (d[ext[a]] > d[or[a]] + l(a)){
           d[ext[a]] = d[or[a]] + l(a);
           pere[ext[a]] = or[a]; }
}
```

1.1. Algorithme de Dijkstra. Cet algorithme commence par initialiser l'application d en posant $d[x_0] = 0$ et pour $x \neq x_0$, $d[x] = \infty$. Ensuite, pour chaque sommet tel que $d[x]$ est minimum, il effectue la relaxation de tous les arcs issus de x. On construit une file de priorité `F` qui contient au départ tous les sommets et sur laquelle on effectue deux opérations :

– recherche et suppression de l'élément x tel que $d[x]$ soit minimum ;
– modification de la valeur de certains $d[x]$.

Il s'exprime de façon plus détaillée comme suit :

```
Pour tout sommet x faire {
    pere[x] = x; d[x] = Maxint;
    Ajouter (x, F);
    }
d[x0] = 0;
Tant que (F non vide) {
    Rechercher x de F tel que d[x] minimum;
    Supprimer x de F;
    Pour tout (a tel que (or[x] == a)) faire Relaxation (a);
    }
```

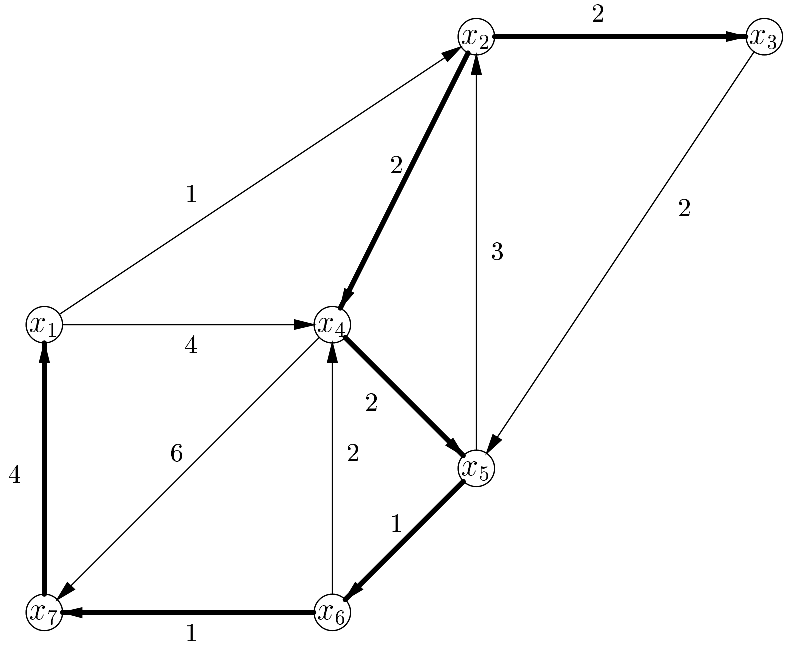


FIG. 1. Plus courts chemins issus du sommet x_2

On peut considérer l'algorithme de Dijkstra comme un algorithme glouton (on traite à chaque étape le sommet le plus intéressant) mais qui remet à jour la valeur de chaque élément après la réalisation d'une étape.

Remarques :

1. Une réalisation efficace de `F` consiste à utiliser la notion de *tas* introduite dans le cours d'algorithmes. Pour un graphe ayant n sommets et m arcs, on effectue n suppressions et m modifications de valeurs de d, chaque suppression demande $\log n$ opérations de même que chaque modification de valeur, puisuqe celle-ci revient à une suppression suivie d'une insertion, la complexité de l'algorithme est en $O(n \log n + m \log n)$.

2. On peut terminer l'algorithme dès que tous les éléments x situés dans `F` sont tels que $d[x] = \infty$; aucun de ces éléments ne sera atteint par un chemin d'origine x.

3. L'algorithme a été décrit pour des graphes orientés ; si le graphe est symétrique il faut considérer une arête comme étant formée de deux arcs opposés.

4. L'algorithme ne donne pas le bon résultat si certains arcs sont négatifs. Il est clair qu'on ne peut s'attendre à obtenir un résultat s'il existe des circuits dont la somme des longueurs des arcs est négative ; par contre en l'absence de tels circuits certains algorithmes donnent les plus courts chemins. C'est le cas de l'algorithme de Ford ci-dessous.

1.2. Algorithme de Ford. Cet algorithme est particulièrement simple : il consiste à effectuer pour un graphe à n sommets, $n-1$ fois la procédure de relaxation sur tous les arcs. Soit plus précisément :

```
        Pour tout sommet x {
            Pere[x] = x; d[x] = Maxint;
            }
        d[x0] = 0;
        Pour i=1 jusqu'a n-1 {
             Pour tout arc a du graphe G faire Relaxation(a)
};
```

On peut démontrer que cet algorithme donne le chemin minimal d'origine x_0. Plus exactement, on vérifie qu'après l'exécution de la i-ème itération de la boucle on a obtenu tous les chemins minimaux composés de i arcs. Ce résultat est valide même en présence d'arcs de longueur négative, la seule condition est que tout circuit soit de longueur positive ou nulle. Ce critère se vérifie une fois l'algorithme de Ford exécuté en testant la présence d'un arc `a` tel que `d[ext[a]] > d[or[a]] + l[a]`. On trouve un tel arc si et seulement si le graphe possède un circuit de longueur négative.

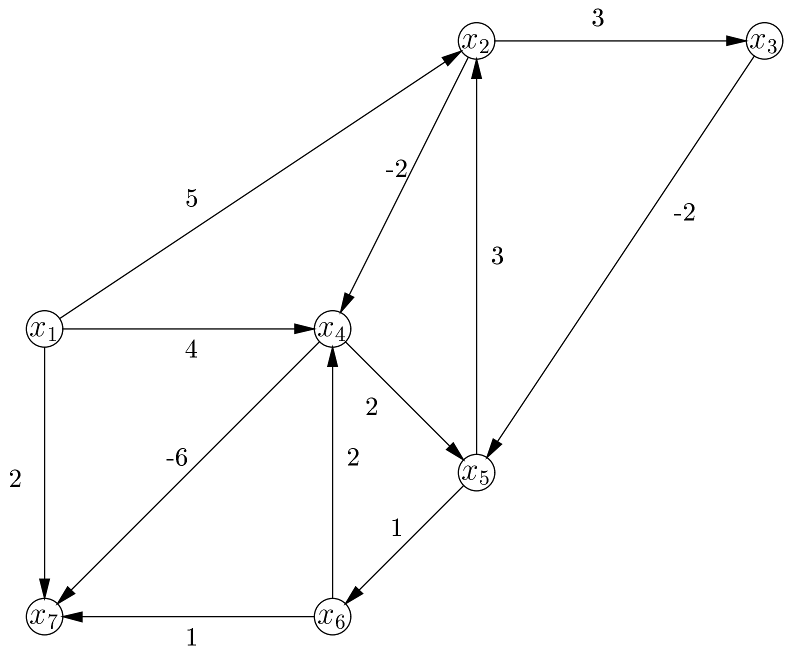


FIG. 2. Un graphe avec des arcs négatifs

Sur l'exemple ci-dessus on obtient les plus courts chemins depuis x_1 à tous les sommets ils sont de longueurs respectives 5, 8,3,5,6,-3 pour $x_2, x_3, x_4, x_5, x_6, x_7$ on vérifie que toute relaxation sur ces valeurs ne modifie pas le résultat trouvé.

La complexité de l'algorithme est en $O(nm)$, ce qui est moins bon que celui de Dijkstra.

2. Plus courts chemins entre tous les couples de sommets

On cherche ici à déterminer tous les plus courts chemins et non pas uniquement ceux d'origine x_0. Nous commençons par donner un algorithme qui détermine les longueurs de ces plus courts chemins. La longueur du plus court chemin entre x_i et x_j est notée $\delta(x_i, x_j)$. On convient de poser $\delta(x_i, x_j) = \infty$ s'il n'existe pas de chemin entre x_i et x_j (en pratique il suffit dans la suite de remplacer ∞ par un nombre suffisamment grand par exemple la somme des longueurs de tous les arcs du graphe). La construction effective des chemins sera examinée ensuite, elle se déduit simplement du calcul de δ. On suppose qu'entre deux sommets il y a au plus un arc. En effet, s'il en existe plusieurs, il suffit de ne retenir que le plus court. On note $\ell(i, j)$ la longueur de l'arc a (le plus court) qui a pour origine x_i et pour extrémité x_j, $\ell(i, j) = \infty$ s'il n'existe pas de tel arc.

Les algorithmes de recherche de chemins les plus courts reposent sur l'observation très simple mais importante suivante :

Remarque *Si f est un chemin de longueur minimale joignant x à y et qui passe par z, alors il se décompose en deux chemins de longueur minimale l'un qui joint x à z et l'autre qui joint z à y.*

Dans la suite, on suppose les sommets numérotés $x_1, x_2, \ldots x_n$ et, pour tout $k > 0$ on considère la propriété P_k suivante pour un chemin f :

$P_k(f)$ Tous les sommets de f, autres que son origine et son extrémité, ont un numéro strictement inférieur à k.

On peut remarquer qu'un chemin vérifie P_1 si et seulement s'il se compose d'un unique arc, d'autre part la condition P_{n+1} est satisfaite par tous les chemins du graphe. Notons $\delta_k(x_i, x_j)$ la longueur du plus court chemin qui vérifie P_k et qui a pour origine x_i et pour extrémité x_j. Cette valeur est ∞ si un tel chemin n'existe pas. Ainsi $\delta_1(x_i, x_j) = \infty$ s'il n'y a pas d'arc entre x_i et x_j et vaut $\ell(a)$ si a est cet arc. D'autre part $\delta_{n+1} = \delta$. Le lemme suivant permet de calculer les δ_{k+1} connaissant les $\delta_k(x_i, x_j)$. On en déduira un algorithme itératif.

LEMME 2.1. *Les relations suivantes sont satisfaites par les δ_k :*

$$\delta_{k+1}(x_i, x_j) = \min(\delta_k(x_i, x_j), \delta_k(x_i, x_k) + \delta_k(x_k, x_j))$$

DÉMONSTRATION. Soit un chemin de longueur minimale satisfaisant P_{k+1}, ou bien il ne passe pas par x_k et on a

$$\delta_{k+1}(x_i, x_j) = \delta_k(x_i, x_j)$$

ou bien il passe par x_k et, d'après la remarque préliminaire, il est composé d'un chemin de longueur minimale joignant x_i à x_k et satisfaisant P_k et d'un autre minimal aussi joignant x_k à x_j. Il a donc pour longueur : $\delta_k(x_i, x_k) + \delta_k(x_k, x_j)$. □

L'algorithme pour la recherche du plus court chemin met à jour une matrice `delta[i,j]` qui a été initialisée à $\ell(i, j)$. À chaque itération de la boucle externe, on fait croître l'indice k du δ_k calculé.

```
Pour k = 1 jusqu'a n {
    Pour  i = 1 jusqu'a n {
        Pour  j = 1 jusqu'a n {
          Si delta[i][j] > (delta[i][k] + delta[k][j])

              delta[i][j]:=  delta[i][k] + delta[k][j]; }}}
```

Sur l'exemple du graphe donné sur la figure 4, on part de la matrice δ_1 donnée par

$$\delta_1 = \begin{pmatrix} 0 & 1 & \infty & 4 & \infty & \infty & \infty \\ \infty & 0 & 3 & 2 & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 0 & 2 & \infty & 6 \\ \infty & 3 & \infty & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & 2 & \infty & 0 & 1 \\ 4 & \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

Après le calcul on obtient :

$$\delta = \begin{pmatrix} 0 & 1 & 4 & 3 & 5 & 6 & 7 \\ 10 & 0 & 3 & 2 & 4 & 5 & 6 \\ 8 & 5 & 0 & 5 & 2 & 3 & 4 \\ 8 & 5 & 8 & 0 & 2 & 3 & 4 \\ 6 & 3 & 6 & 3 & 0 & 1 & 2 \\ 5 & 6 & 9 & 2 & 4 & 0 & 1 \\ 4 & 5 & 8 & 7 & 9 & 10 & 0 \end{pmatrix}$$

Pour le calcul effectif des chemins les plus courts, on utilise une matrice `suivant` telle que `suivant[i][j]` est le sommet qui suit i dans le chemin le plus court qui va de i à j. Les valeurs `suivant[i][j]` sont initialisées à j s'il existe un arc de i vers j et à -1 sinon, `suivant[i][i]` est lui initialisé à i. Le calcul précédent qui a donné δ peut s'accompagner de celui de `suivant` en ajoutant la ligne :

```
suivant[i][j] = suivant[i][k]
```

dans le cas où le test sur `delta[i][j]` est positif.

Une fois le calcul des deux matrices effectué on peut retrouver le chemin le plus court qui joint i à j par la procédure :

```
k = i;
while (k != j) k= suivant[k][j];
```

Sur l'exemple précédent on trouve pour `suivant` :

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

3. Généralisation de l'algorithme

L'algorithme précédent a une structure qui rappelle le calcul de la puissance p-ième d'une matrice. Cela se voit mieux sur l'algorithme ci-dessous qui donne la longueur $d^{(p)}(i,j)$ du plus court chemin de longueur inférieure ou égale à p entre les sommets x_i et x_j d'un graphe $G = (X, A, \ell)$. Dans ce qui suit $\ell(i,j)$ désigne la longueur de l'arc le plus court entre x_i et x_j, égal à ∞ s'il n'existe pas de tel arc.

$$\begin{array}{rcl} d^{(1)}(i,j) & = & \ell(i,j), \\ d^{(p)}(i,j) & = & \min[d^{(p-1)}(i,j), \min_{1\leq k\leq n}[d^{(p-1)}(i,k) + \ell(k,j)]]. \end{array}$$

Ceci est exactement le calcul de la puissance de la matrice contenant les $\ell(i,j)$ si on utilise des opérations particulières pour la somme et le produit des coefficients. Il faut en effet remplacer la somme par min et le produit par $+$. On est ainsi amené à introduire la notion de semi-anneau.

3.1. Semi-anneaux. Définition. Un *semi-anneau* est un ensemble S muni de deux lois de compositions notées ici $\oplus$ et $\odot$ telles que ;

(1) La loi $\oplus$ est associative, commutative et possède un élément neutre noté $\underline{0}$
(2) La loi $\odot$ est associative, et possède un élément neutre noté $\underline{1}$
(3) La loi $\odot$ est distributive (à gauche et à droite) par rapport à $\oplus$ et pour tout s dans S on a $s \odot \underline{0} = \underline{0}$

Tout anneau est bien entendu un semi-anneau, en effet en plus de ces conditions on a aussi dans un anneau l'existence d'un inverse pour $\oplus$. Les exemples de semi-anneaux sont nombreux en voici quelques-uns qui sont utiles en algorithmique.

1. Le semi-anneau de *Boole* $\mathcal{B}$ a deux éléments $\{0, 1\}$, $\oplus$ est l'opération `ou`, $\odot$ est l'opération `et`.

$\oplus$	0	1
0	0	1
1	1	1

$\odot$	0	1
0	0	0
1	0	1

2. L'ensemble des entiers naturels $\mathbb{N}$, pour les lois usuelles d'addition et de multiplication.

3. L'ensemble $\mathbb{N} \cup \{\infty\}$ des entiers naturels augmenté de ∞, pour les lois min et $+$. Les éléments neutres sont respectivement ∞ et 0 pour ces deux lois. On vérifie simplement que min est associative, commutative et possède ∞ comme élément neutre, d'autre part $+$ est associative d'élément neutre 0 ; enfin on a bien :

$$a + \min(b, c) = \min(a + b, a + c), \quad \forall a \in S,\ a + \infty = \infty$$

4. Si on remplace min par max dans l'exemple ci-dessus, on n'a pas tout à fait un semi-anneau, car l'élément neutre de max qui est 0, ne satisfait pas la relation voulue avec $+$, par contre c'est le cas pour max et la multiplication.

5. Le dernier exemple est un peu plus nouveau. On considère un alphabet A et l'ensemble A^* des mots sur A, S est alors l'ensemble des parties $\mathcal{P}(A^*)$ de A^*. L'opération $\oplus$ est la réunion et l'opération $\odot$ est définie à partir de la concaténation . Pour deux parties U et V de A^* :

$$U \odot V \quad = \{u.v | u \in U, v \in V\}$$

Les éléments neutres sont respectivement $\emptyset$ pour l'union et l'ensemble composé du seul mot vide ε pour $\odot$.

3.2. Puissances de matrices. On peut définir des matrices carrées $n \times n$ dont les coefficient sont dans un semi-anneau S ; le produit de deux telles matrices P et Q est alors la matrice M dont le coefficient $M_{i,j}$ est donné par :

$$M_{i,j} = \oplus_{k=1}^{n}(P_{i,k} \odot Q_{k,j})$$

Le calcul des plus courts chemins dans un graphe se généralise à celui de graphes dont les arcs sont valués par des éléments d'un semi-anneau S. Soit $G = (X, A, \lambda)$, où (X, A) est un graphe orienté et λ une application de A dans un semi-anneau S. La valuation λ se généralise des arcs aux chemins en posant, pour tout chemin $f = a_1, a_2, \ldots, a_k$, $\lambda(f)$ égale au « produit » $\odot_{i=1,k}\lambda(a_i)$. Soit M la matrice dont le coefficient $M_{i,j}$ est égal à la « somme » (par $\oplus$) des $\lambda(a)$ pour tous les arcs a d'origine x_i et d'extrémité x_j, $M_{i,j}$ est égal à $\underline{0}$ s'il n'existe pas de tel arc. On a alors le théorème suivant :

THÉORÈME 2.1. *L'élément $M_{i,j}^k$ de la puissance k-ième de la matrice M est égal à la « somme » par $\oplus$ des $\lambda(f)$ pour tous les chemins f d'origine x_i, d'extrémité x_j et de longueur k.*

La preuve de ce résultat s'effectue par récurrence sur k.

Remarques :

1. On a souvent à calculer la valeur donnée par le théorème pour tous les chemins de longueur inférieure ou égale à k et non pas seulement pour ceux de longueur k, il faut alors faire la somme des puissances M^i de M pour $i = 0$ (M^0 est la matrice identité) jusqu'à k.

2. Dans de nombreux cas on constate pour un certain entier k que pour tout $k' > k$:

$$\sum_{i=0,k} M^i = \sum_{i=0,k'} M^i$$

On dit alors que M est quasi-inversible, son quasi-inverse est $M^* = \sum_{i=1,k} M^i$ en effet on a :

$$I \oplus (M \odot M^*) = M^*$$

3. Le théorème précédent donne avec les différents exemples cités plus haut des résultats intéressants d'un point de vue algorithmique. Dans le cas du semi-anneau de Boole $\mathcal{B}$, $M^k_{i,j} = 1$ indique l'existence d'un chemin de longueur k entre x_i et x_j ; dans celui de $S = \mathbb{N}$ avec les opérations usuelles $M^k_{i,j}$ est le nombre de tels chemins. Lorsque l'on prend $S = \mathbb{N} \cup \{\infty\}$ et les opérations min, + on obtient les plus courts chemins, enfin pour $S = \mathcal{P}(A^*)$ et les opérations $\cup$ et ., le coefficient $M^k_{i,j}$ est égal à l'ensemble de toutes les étiquettes des chemins (l'étiquette d'un chemin est la concaténation des étiquettes des arcs qui le composent) de longueur k qui ont pour origine x_i et pour extrémité x_j. Cette propriété est utilisée en théorie des automates et des langages rationnels.

4. Programmation dynamique

La recherche d'un chemin minimal dans un graphe entre tous les couples de sommets est un cas particulier d'une méthode générale de recherche d'optimum que l'on appelle *programmation dynamique*.

4.1. Présentation de la méthode. Les problèmes que l'on peut traiter à l'aide de cette méthode ont deux caractéristiques principales

- Une solution optimale à un problème de taille k contient dans un sens à préciser des solutions optimales à des problèmes de tailles plus petites que k.
- On peut construire une solution optimale à un problème de taille k en connaissant un petit nombre de solutions optimales à des problèmes de tailles plus petites que k.

Une méthode de résolution de problème par la programmation dynamique consiste à résoudre itérativement des problèmes de tailles de plus en plus grandes en conservant en mémoire les résultats obtenus pour les problèmes de petites tailles. En général une mémoire de l'ordre de n^2 pour un problème de taille n est nécessaire.

4.2. Un problème générique. L'exemple qui suit est un exemple typique de tel problème, on l'exprime souvent sous la forme suivante :

Un système est caractérisé par son *état* qui est un élément σ d'un ensemble fini Σ, on considère des suites $\sigma_0, \sigma_1, \ldots, \sigma_k$ d'états successifs du système, σ_i est l'état au temps $i = 1, \ldots, k$. Le système est commandé par des *décisions* à prendre représentées par des entiers $x_1, x_2, \ldots, x_k$: x_i est la décision prise au temps i. Le comportement du système est régi par k fonctions de transitions ϕ_i, une pour chaque valeur de i. La fonction de transition ϕ_i a pour donnée un état σ et une décision x, elle a pour résultat un état ; $\phi_i(\sigma, x)$ est l'état atteint au temps i si l'état au temps $i - 1$ est σ et si la décision prise au temps i est x.

Une fonction de coût c_i pour chaque i compris entre 1 et k permet de mesurer les coûts des décisions prises. La fonction de coût a les mêmes données que la fonction de transition et a pour résultat un entier $c_i(\sigma, x)$ qui est le coût de la décision x au temps i si l'état du système est σ au temps $i - 1$.

Le problème consiste à trouver une suite de transitions entre deux états donnés l'état initial et l'état final qui minimise la somme des coûts. Quelquefois l'état final n'est pas précisé, il faut alors obtenir celui qui donne le minimum.

Une autre version du problème consiste à considérer non pas des coûts c_i mais des gains g_i ; il faut alors rendre le gain maximum plutôt que de chercher un coût minimal.

Dans l'exemple de la recherche du chemin minimal la décision à prendre consiste à choisir un des arcs ou à rester sur place, l'état du système est le sommet où l'on se trouve dans le graphe. Les transitions possibles sont celles où l'on a choisi un arc ayant pour origine le sommet où l'on se trouve, le résultat de la transition consiste à passer au sommet extrémité de l'arc choisi, et le coût de la transition est égal à la longueur de l'arc.

4.3. Algorithme. L'algorithme de programmation dynamique s'exprime de la façon suivante : pour chaque couple d'états σ, τ on note $\delta_k(\sigma, \tau)$ le coût minimal d'une suite de transitions conduisant de σ à l'instant 0 à τ à l'instant k ; ce coût est infini si une telle suite n'existe pas. On a alors la formule de récurrence suivante qui permet de résoudre le problème de manière itérative :

$$\delta_k(\sigma, \tau) = \min_{\theta \in \Sigma} [\delta_{k-1}(\sigma, \theta) + c_k(\theta, x_k)]$$

Dans cette formule x_k satisfait $\phi_k(\theta, x_k) = \tau$.

On remarque qu'il suffit alors de conserver les valeurs de δ_{k-1} pour obtenir celles de δ_k, il n'est pas nécessaire de connaître les valeurs de δ_i pour $i < k - 1$, la mémoire nécessaire est bien en $O(n^2)$ où n est le nombre d'états du système.

4.4. Le sac à dos. Le problème du sac à dos est connu comme étant NP-complet en général, toutefois dans le cas particulier où les poids des objets à emporter sont des entiers on peut obtenir un algorithme polynomial en fonction de la capacité totale du sac. Précisons le problème : on se donne des entiers $p_1, p_2, \ldots, p_k$ représentant les poids de certains objets et un nombre P représentant le poids maximal des objets que l'on peut emporter dans le sac ; pour chaque objet i ($i = 1, \ldots, k$), a_i est l'indice de satisfaction obtenu si on emporte i dans le sac. Le problème consiste à trouver le sous-ensemble d'objets qui maximise la somme des indices de satisfaction et tel que la somme des poids des objets qui le composent est inférieure ou égale à P.

Avec la terminologie du paragraphe précédent on considère des gains et non pas des coûts. L'état du système est la capacité qui reste dans le sac à dos, ainsi l'état initial est P, lorsque l'on prend l'objet i on passe d'une capacité p à la capacité $p - p_i$. Les différentes décisions sont représentées par une variable qui prend les valeurs 0 et 1, $x_i = 1$ si on met l'objet i dans le sac $x_i = 0$ sinon. Ainsi $\phi_i(p, x) = p - xp_i$ et $g_i(p, x) = a_i$. L'algorithme de programmation dynamique consiste à gérer un tableau `benefice[p]` de taille P qui donne à chaque étape i le bénéfice maximum que l'on peut réaliser en prenant au plus i objets et en laissant une capacité résiduelle de p dans le sac. Les poids de chaque objet i sont donnés dans un tableau `poids[i]` et les nombres a_i dans un tableau `a[i]`.

```
Pour p = 1 jusqu'a  P faire benefice[p] = 0;
Pour i = 1 jusqu'a k faire {
         q = poids[i];
         Pour p = q jusqu'a P faire
            Si (benefice[p-q] < benefice[p] + a[i])
                 benefice[p-q] = benefice[p] + a[i];
        }
```

5. Traitement de séquences

La comparaison de deux séquences afin de déterminer leur similarité joue un rôle central en bio-informatique. On utilise souvent la programmation dynamique pour résoudre ces problèmes qui proviennent de l'analyse du génome humain et de la compréhension de sa structure primaire.

5.1. Sous-séquence communes. Un exemple simple consiste à déterminer la plus longue sous-séquence commune à deux séquences données. Précisons tout d'abord quelques définitions. Une *séquence* (ou un *mot*) est une suite finie de symboles (ou *lettres*) pris dans un ensemble fini (ou *alphabet*). Si $u = a_1 \cdots a_n$ est une séquence, où $a_1, \ldots, a_n$ sont des lettres, l'entier n est la *longueur* de u. Une séquence $v = b_1 \cdots b_m$ est une *sous-séquence* de $u = a_1 \cdots a_n$ s'il existe des entiers $i_1, \ldots, i_m$, $(1 \leq i_1 < \cdots i_m \leq n)$ tels que $a_{i_k} = b_k$ $(1 \leq k \leq m)$. Une séquence w est une *sous-séquence commune* aux séquences u et v si w est sous-séquence de u et de v. Une sous-séquence commune est dite *maximale* si sa longueur est maximale parmi toutes les sous-séquences communes.

On cherche d'abord à déterminer la longueur d'une sous-séquence commune maximale à $u = a_1 \cdots a_n$ et $v = b_1 \cdots b_m$. Pour cela, on note $L(i, j)$ la longueur d'une sous-séquence commune maximale aux mots $a_1 \cdots a_i$ et $b_1 \cdots b_j$, $(0 \leq j \leq m, 0 \leq i \leq n)$. On peut montrer que :

$$L(i,j) = \begin{cases} 1 + L(i-1, j-1) & \text{si } a_i = b_j \\ \max(L(i, j-1), L(i-1, j)) & \text{sinon.} \end{cases}$$

En effet, soit w une sous séquence de longueur maximale, commune à $a_1 \cdots a_{i-1}$ et à $b_1 \cdots b_{j-1}$ si $a_i = b_j$, wa_i est une sous-séquence commune maximale à $a_1 \cdots a_i$ et $b_1 \cdots b_j$. Si $a_i \neq b_j$ alors une sous-séquence commune à $a_1 \cdots a_i$ et $b_1 \cdots b_j$ est ou bien commune à $a_1 \cdots a_i$ et $b_1 \cdots b_{j-1}$ (si elle ne se termine pas par b_j) ; ou bien à $a_1 \cdots a_{i-1}$ et $b_1 \cdots b_j$, (si elle ne se termine par a_i). On obtient ainsi l'algorithme qui permet de déterminer la longueur d'une sous séquence commune maximale à $a_1 \cdots a_n$ et $b_1 \cdots b_m$

Il est assez facile de transformer l'algorithme pour retrouver une sous-séquence maximale commune au lieu de simplement calculer sa longueur. Pour cela, on met à jour un tableau `provient` qui indique lequel des trois cas a permis d'obtenir la longueur maximale. Une fois ce calcul effectué il suffit de remonter à chaque étape de `i,j` vers `i-1, j-1`, vers `i, j-1` ou vers `i-1,j` en se servant de la valeur de `provient[i][j]`. On peut améliorer cette méthode et rendre la complexité plus faible grâce à des techniques astucieuses de programmation, c'est ce qui est fait pour la commande `diff` du système UNIX.

5.2. Alignement de séquences. Soit u et v deux mots écrits à l'aide de l'alphabet $\mathbb{A}$ (on prend souvent $\mathbb{A} = \{G, T, A, C\}$ en bioinformatique), on appelle alignement de u, v un couple de mots u', v' sur l'alphabet $\mathbb{A} \cup \{e\}$ (où $e \notin \mathbb{A}$ représente le caractère d'espacement), couple qui satisfait les conditions suivantes

- les deux mots u' et v' ont même longueur ($|u'| = |v'|$) ;
- la suppression de e dans u' et dans v' donne u et v respectivement ;
- les lettres e ne se retrouvent pas en même position dans u' et v'.

Un exemple d'alignement de $u = CGATTAG$ et $v = GATCGA$ est

$$\begin{array}{rcl} u' & = & C\,G\,A\,T\,T\,e\,A\,G \\ v' & = & e\,G\,A\,T\,G\,G\,A\,e \end{array}$$

Il y a bien entendu de très nombreux alignements possibles pour deux séquences, en fait on cherche ceux qui sont les plus intéressants dans le sens précis que l'on décrit ici :

On suppose donnée une fonction de similarité p entre les couples de lettres. La valeur de $p(a, b)$ est un nombre reél d'autant plus grand que les lettres sont considérées comme similaires d'un point de vue biologique, ainsi on prendra des valeurs négatives quand les lettres sont différentes, soit : $b \neq a$, $p(a, a) > 0 > p(a, b)$, de plus la symétrie impose $p(a, b) = p(b, a)$. D'autre part on se donne aussi un réel $q < 0$ qui exprime la similarité d'une lettre avec l'espacement, elle est indépendante de la lettre, ainsi pour tout a on a $p(a, e) = p(e, a) = q$.

Le score d'un alignement est alors donné par

$$\sum_{i=1}^{n} p(u'_i, v'_i)$$

où n est la longueur commune à u' et v'.

Par exemple, pour l'exemple précédent lorsque $p(a, b) = -1$ pour $a \neq b$, $p(a, a) = 5$ et $q = -2$ on trouve un score de 13.

La détermination du score le plus élevé d'un alignement s'effectue grâce à un algorithme de programmation dynamique basé sur la remarque suivante :

Le meilleur alignement de ua et vb s'obtient soit à partir de celui de u et v en alignant a et b, soit à partir de celui de ua et v en alignant b avec un espacement, soit enfin à partir de celui de u et vb en alignant a avec un espacement.

On construit ainsi pas à pas un tableau ms (pour meilleur score) à l'aide de la récurrence suivante :

Soit n la longueur de u et m celle de v, on note par $ms_{i,j}$ (pour $0 \leq i \leq n,\ 0 \leq j \leq m$) le score d'un meilleur alignement entre $u_1u_2 \cdots u_i$ et $v_1v_2 \cdots v_j$. Par convention on pose $ms_{0,j} = qj,\ ms_{i,0} = qi$ correspondant à un aligenment dans lequel tous les caractères de l'une des séquences sont des espacements.

$$ms_{i,j} = \max \begin{cases} ms_{i-1,j-1} + p(u_i, v_j) \\ ms_{i-1,j} + q \\ ms_{i,j-1} + q \end{cases}$$

On note que le calcul de ms s'effectue en temps $O(nm)$ et semble nécessiter un espace de taille $O(nm)$, en fait on constate que l'on peut se limiter à conserver en mémoire une seule ligne de la matrice, la ligne suivante remplace alors la précédente lors du calcul.

Un dernier problème consiste à déterminer l'alignement optimal lui-même une fois calculé son score, c'est à dire connaisant le tableau ms, ceci est relativement simple si l'on connaît la totalité du tableau : on part de $ms_{m,n}$ et on remonte dans le tableau de $ms_{i,j}$ vers celui des $ms_{i-1,j-1}, ms_{i-1,j}$ ou $ms_{i,j-1}$ qui a déterminé sa valeur.

Dans le cas où l'on ne connaît que la dernière ligne du tableau, il faut utiliser une technique procédant du principe de *diviser pour regner* et qui pourra être vue en exercice.

Notons deux problèmes voisins de celui qui est considéré ici qui peuvent être traités par les mêmes techniques.

Le premier consiste étant données deux séquences u et v à trouver un facteur de chacune de celles-ci formant un couple qui présente un alignement de score maximal (rappelons qu'un facteur d'une séquence est une sous-séquence constituée de caractères consécutifs). Pour résoudre ce problème (on utilise ici le fait que $q < 0$ et $p(a, b) < 0$) si $a \neq b$ on applique l'algorithme ci-dessus en considérant $ms_{i,j}$ comme le meilleur score d'un alignement entre un suffixe de $u_1u_2 \ldots u_i$ et un suffixe de $v_1v_2 \ldots v_j$; ce suffixe pouvant être le mot vide, on voit que $m_{i,j}$ ne peut pas être négatif, ainsi 0 doit figurer comme quatrième possibilité dans la formule de récurrence donnant $m_{i,j}$.

Le second problème consiste à déterminer le meilleur alignement pour une fonction de score qui ne tient pas compte des espacements situés en début ou en fin de mot. L'algorithme pour ce problème ne différe du précédent que dans l'initialisation des $m_{i,0}$ et $m_{j,0}$ qui doivent prendre la valeur 0. On retient alors la valeur maximale de la dernière ligne ou de la dernière colonne.

5.3. Distance d'édition. Un problème voisin du précédent consiste à déterminer ce qu'on appelle la *distance d'édition* entre deux mots f et g. Cette notion est introduite en considérant les trois opérations suivantes permettant de transformer une séquence en une autre :

(1) La suppression d'une lettre (notée S)

(2) L'ajout d'une lettre (notée A)

(3) Le remplacement d'une lettre par une autre (notée R)

Chacune de ces opérations a un coût et il s'agit de déterminer la suite d'opérations ayant un coût minimum permettant de passer d'un mot à un autre. Ainsi pour passer de

ACGATC à CAGTCAA on peut faire un ajout (en tête), qui donne CACGATC, puis une suppression pour obtenir CAGATC, une nouvelle suppression et deux ajouts en fin de mot. On peut aussi faire deux remplacements pour obtenir CAGATC, puis une suppression et deux ajouts. La meilleure suite d'opérations dépend bien entendu des valeurs des coûts, mais on voit bien que des techniques très voisines de ce qui a été fait ci-dessus donnent le résultat.

CHAPITRE 3

Flots et couplages

Dans ce chapitre, nous nous préoccupons de propriétés globales d'un autre type qui mettront en jeu *simultanément* l'ensemble des arêtes du graphe ; il s'agit de modéliser la circulation de personnes, voitures, informations ou fluides à travers un réseau de façon à rendre maximum l'acheminement global tout en respectant des contraintes locales de capacités des liaisons entre deux sites, ces notions se modélisent à l'aide du concept de *flot*.

La recherche d'un flot maximal est déjà intéressante par elle-même, mais il y a aussi plusieurs applications de la notion de flot. Une des applications, parmi les plus répandues, est la recherche d'un *couplage* maximum dans un graphe biparti. Le couplage modélise le problème de l'affectation (machine/ouvrier, processus/unité de calcul, stagiaire/entreprise).

1. Flots

On considère un graphe orienté dont chaque arc est affecté d'une capacité, nombre entier positif ou nul. Étant donnés deux sommets s et t du graphe, appelés respectivement *source* et *puits*, on cherchera à faire passer dans le graphe un flot maximal de s vers t en respectant les capacités de chaque arc et en appliquant les lois de Kirchoff aux sommets qui interdisent en quelque sorte l'accumulation en un sommet.

Plus précisément on considère un graphe avec capacité $G = (X, A, \gamma)$, où X est l'ensemble des sommets, A l'ensemble des arcs et γ la fonction de capacité qui à chaque arc a associe un entier[1] positif ou nul $\gamma(a)$. On suppose donnés aussi deux sommets s et t entre lesquels on cherche à faire passer un flot maximal. On suppose qu'il y a au plus un arc d'origine et d'extrémité données, ainsi on notera (x, y) l'arc a ayant x comme origine et y comme extrémité. Par souci de simplicité on supposera qu'il n'existe pas d'arc de type (x, x) bouclant sur un sommet. On appelle *flot* de source s et de puits t dans le graphe

[1]on peut en fait considérer des réels, mais il est plus simple de supposer les capacités et les flots entiers et cette restriction ne gène pas dans la pratique

$G = (X, A)$, muni de la capacité γ, une fonction positive entière ϕ sur A qui vérifie les deux propriétés suivantes :

$$(1) \qquad 0 \leq \phi(x,y) \leq \gamma(x,y) \quad , \text{ pour tout arc } (x,y) \in A,$$

$$(2) \qquad \sum_{(x,y)\in A} \phi(x,y) = \sum_{(y,z)\in A} \phi(y,z) \quad , \text{ pour tout sommet } y \neq s,t.$$

On définit une *coupure* d'un graphe (*cut* dans la littérature anglo-saxonne) comme une partition des sommets de G en deux sous-ensembles disjoints dont l'un, S, contient s et l'autre, T, contient t. La valeur du flot associée à une coupure (S,T) est la quantité : $\Phi(S,T) = \sum_{(\sigma,\tau)\in A} \phi(\sigma,\tau) - \sum_{(\tau,\sigma)\in A} \phi(\tau,\sigma)$, où $\sigma \in S$ et $\tau \in T$. On définit également la *capacité*, $\gamma(S,T)$, d'une coupure comme la quantité :

$$\gamma(S,T) = \sum_{(\sigma,\tau)\in A} \gamma(\sigma,\tau),$$

où $\sigma \in S$ et $\tau \in T$.

On vérifie quelques propriétés simples.

PROPOSITION 3.1. *Étant donné un flot ϕ, toutes les coupures (S,T) ont la même valeur de flot $\Phi \leq \gamma(S,T)$. Si, de plus, s n'a pas d'arc entrant et t n'a pas d'arc sortant, alors $\Phi = \sum_{(s,x)\in A} \phi(s,x) = \sum_{(y,t)\in A} \phi(y,t)$.*

La preuve se fait en considérant une coupure (S,T) et en la transformant en $(S \cup \{x\}, T \setminus \{x\})$ pour $x \in T$. On vérifie alors que la nouvelle coupure possède la même valeur de flot Φ en appliquant la loi de Kirchoff sur le sommet x.

Une conséquence immédiate est alors que le flot maximal ne saurait excéder la capacité d'une coupure.

PROPOSITION 3.2. $\max_\phi \Phi \leq \min_{(S,T)} \gamma(S,T)$.

Il reste à obtenir une borne inférieure pour un flot maximal. Pour ce faire une notion naturelle est celle de *chemin saturé*. Etant donné un graphe avec capacités $G = (X, A, \gamma)$ et un flot ϕ, un arc a est *saturé* si $\phi(a) = \gamma(a)$. Un chemin est saturé s'il contient au moins un arc saturé. Il est clair que s'il existe un chemin non saturé qui joint s à t alors on peut améliorer le flot en ajoutant à chaque arc de ce chemin une quantité de flot égale au minimum de $\gamma(a) - \phi(a)$ pour l'ensemble des arcs du chemin. Un *flot est saturé* si tout chemin qui mène de s à t est saturé. Une première intuition engage à penser qu'un flot saturé est maximal, or il n'en est rien comme le montre l'exemple de la figure ci-dessous, où les capacités sont indiquées en caractère gras, et le flot ϕ à côté des capacités. Le flot est saturé mais on peut l'améliorer en ajoutant une unité de flot sur les arcs $(s,x_3), (x_3,x_4), (x_1,x_2)$ et (x_2,t) et en faisant passer un flot nul sur l'arc (x_1,x_4).

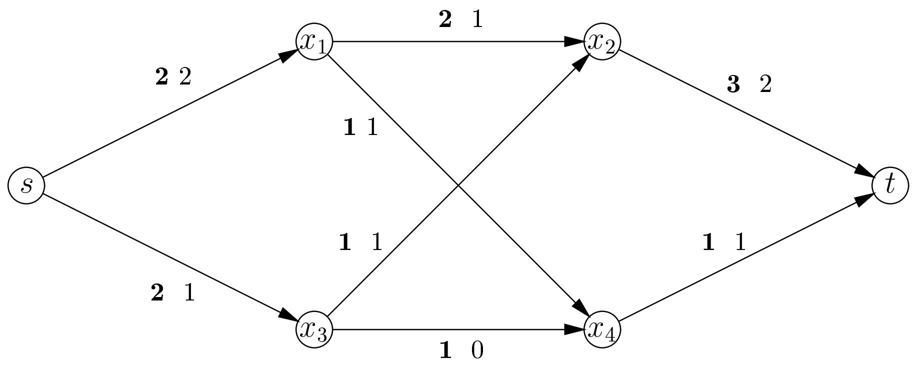


FIG. 1. Un flot saturé non maximal. Les capacités sont en caractères gras.

2. Méthode de Ford-Fulkerson

Pour améliorer un flot saturé il faut, comme on l'a vu sur l'exemple plus haut , diminuer sa valeur sur certains arcs pour pouvoir l'augmenter sur d'autres. Pour cela on construit un graphe G_ϕ associé au flot ϕ sur le graphe G, qui a les mêmes sommets que G et dont les arcs sont obtenus à partir de ceux de G en conservant ceux qui ne sont pas saturés et en *retournant* les arcs contenant une quantité non nulle de flot. On appelle parfois ce graphe le *graphe des améliorations.*

Plus précisément, $G_\phi = (X, A_\phi, \gamma_\phi)$ est tel que :

$$(x,y) \in A_\phi \quad \text{si} \quad \left\{ \begin{array}{lll} (x,y) \in A & \text{et} & \phi(x,y) < \gamma(x,y) \\ (y,x) \in A & \text{et} & \phi(x,y) > 0 \end{array} \right.$$

De plus on a les nouvelles capacités γ_ϕ données par

$$\gamma_\phi(x,y) = \quad \left\{ \begin{array}{lll} \gamma(x,y) - \phi(x,y) & \text{si} & \phi(x,y) < \gamma(x,y) \\ \phi(y,x) & \text{si} & \phi(y,x) > 0 \end{array} \right.$$

Un chemin c dans G_ϕ joignant s à t permet d'obtenir un flot ϕ' qui améliore ϕ. Soit u la plus petite valeur de $\gamma_\phi(x,y)$ pour tous les arcs appartenant au chemin c, on calcule alors ϕ' par

$$\phi'(x,y) = \quad \left\{ \begin{array}{lll} \phi(x,y) & \text{si} & (x,y) \notin c \\ \phi(x,y) + u & \text{si} & (x,y) \in A \quad \text{et} \quad (x,y) \in c \\ \phi(y,x) - u & \text{si} & (x,y) \in A \ \ \text{et} \ \ (y,x) \in c \end{array} \right.$$

Ceci fournit une technique très générale de construction par amélioration successive des flots, paradigme dû à Ford et Fulkerson :

```
initialiser le flot à 0
tant qu'il existe un chemin améliorable p dans G_φ
faire : améliorer le flot sur p.
```

La validité de la méthode est garantie par le fait qu'on obtient ainsi un nouveau flot compatible avec les contraintes de capacité sur G. En effet, il suffit de vérifier que pour chaque arc la capacité n'est pas dépassée et que le flot sur une arête de G n'est pas négatif. La valeur du nouveau flot est évidemment la somme des valeurs et est en croissance, car on ne saurait diminuer le flot partant de s. Il reste à montrer que le flot obtenu à la fin de l'algorithme est maximum pour cela on utilise :

Théorème 3.1 (Maxflow-Mincut). *Soit ϕ un flot de s vers t dans un graphe G. Les trois conditions suivantes sont équivalentes :*
1. le flot ϕ est maximal ;
2. le graphe d'amélioration G_ϕ ne contient pas de chemin joignant s à t ;
3. la valeur Φ de ϕ est égale à la capacité $\Gamma(S,T)$ pour une certaine coupure (S,T).

Preuve : (1) entraine (2) car si G' avait un chemin améliorable on pourrait augmenter le flot d'après la proposition précédente.
(2) entraine (3) ; considérons l'ensemble S des sommets accessibles de s dans G' par un chemin formé d'arcs de capacité *non nulle*, et soit $T = X \backslash S$, son complémentaire, qui est bien sûr non vide puisque contenant t. S étant maximal, les arêtes de S à T ont toutes une capacité nulle dans G'. Et donc $\Phi = \Gamma(S,T)$.
Enfin, (3) entraine (1) puisque le maximum possible est atteint.

Pour résumer l'algorithme de Ford et Fulkerson s'écrit comme suit et a pour complexité $O(m\Phi^\star)$, où m est le nombre d'arcs, $\Phi^\star$ est le flot maximal et en supposant que toutes les capacités sont entières.

```
FORD-FULKERSON(G,s,t)
pour chaque (x,y) de V faire φ(x,y) :=0 ;
tant qu'il existe un chemin p de s à t dans G' faire
        h :=min capacité d'amélioration des arcs de p
        pour chaque arc a de p faire φ(a) :=φ(a)+h ;
```

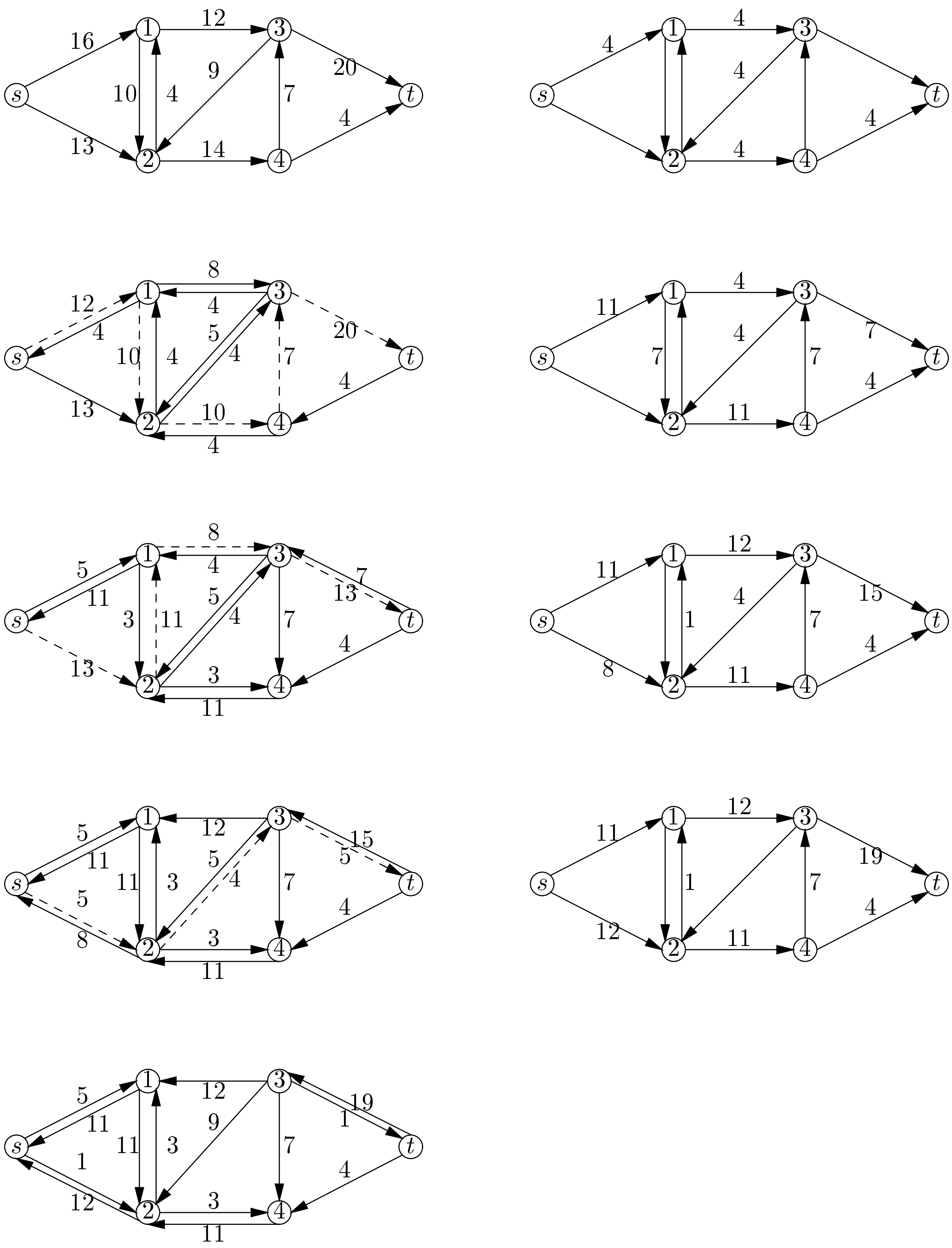


FIG. 2. Application de l'algorithme de Ford-Fulkerson

L'exemple plus haut montre une des étapes de l'algorithme sur un graphe simple, sur la figure un graphe avec les capacités est donné, puis on a fait figuré sur la partie gauche le graphe des améliorations et sur la partie droite les flots calculés au fur et à mesure des améliorations.

Remarque La complexité de cette méthode dépend beaucoup de la façon dont les améliorations sont effectuées. Sur le graphe pathologique suivant dont le flot maximal est de 200.000, le choix systématique d'un chemin passant par la diagonale (x, y) de capacité 1 conduit à faire 200.000 d'itérations alors que le choix direct des chemins (s, x, t) puis (s, y, t) permettrait de conclure en 2 étapes.

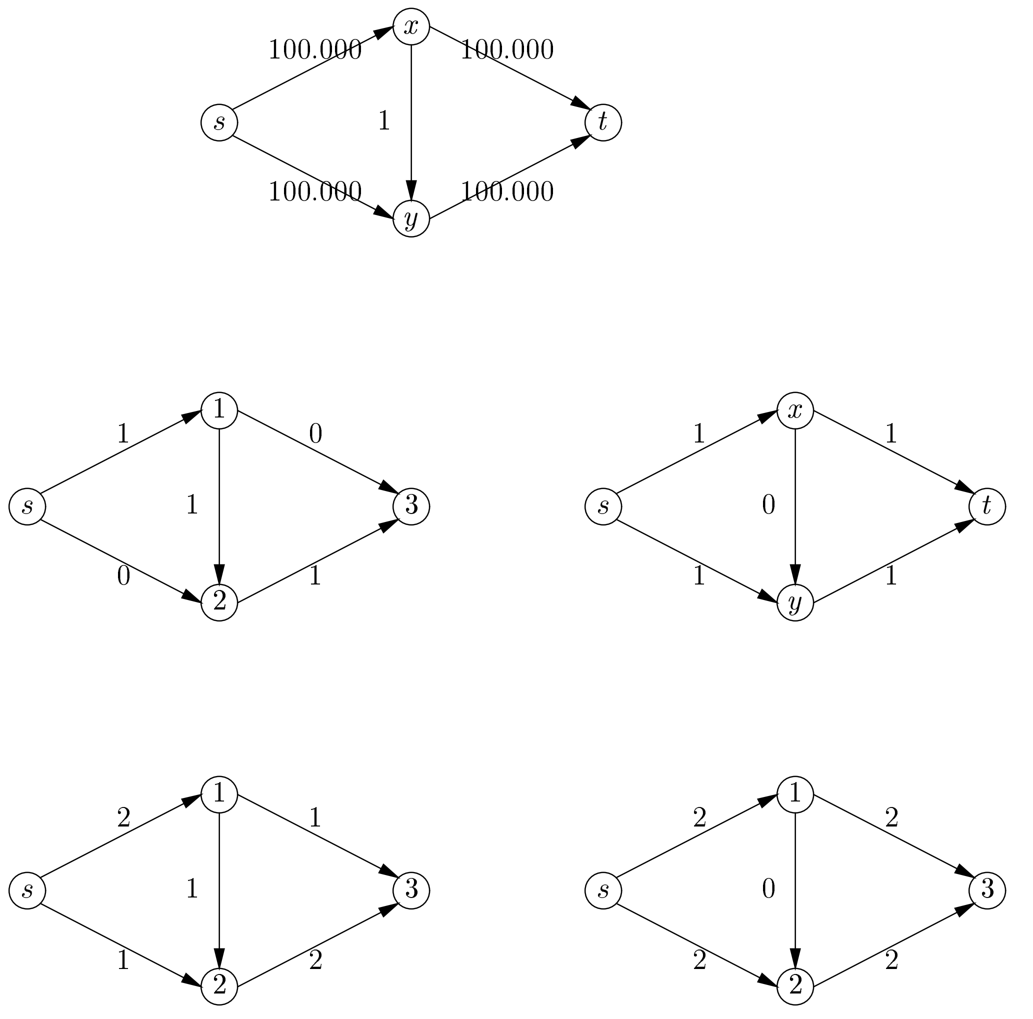


FIG. 3. Un exemple de calcul très long

3. Un algorithme en temps polynomial

Pour rendre le comportement de l'algorithme moins dépendant des données il faut préciser la stratégie de choix des chemins sur lesquels se font les augmentations du flot. Nous en décrivons deux, et il est même possible de calculer le flot maximal en temps $O(n^3)$.

3.1. L'algorithme d'Edmonds-Karp. Dans cette implantation, les chemins améliorables sont énumérés selon leur longueur au moyen d'une recherche en *largeur* des sommets accessibles depuis s dans le graphe G'.

PROPOSITION 3.3. *Dans l'algorithme de Edmonds-Karp, la longueur du chemin amélioré ne décroît jamais.*

Preuve : Supposons qu'à une étape la longueur du chemin améliorable vienne à diminuer. Ceci veut donc dire qu'il existe un sommet x dont les distances à s dans les deux graphes successifs G et G' sont telles que $d(s,x) > d'(s,x)$ et choisissons, à cette étape, un sommet x qui soit le plus proche de s. Remarquons que sur le chemin améliorable p de s à t qui passe par x, le prédécesseur y de x satisfait $d(s,y) \leq d'(s,y)$.
Considérons alors le flot $\phi(y,x)$ avant l'augmentation. Si $\phi(y,x) < \gamma(y,x)$, capacité de l'arête, alors $d(s,x) \leq d(s,y)+1 \leq d'(s,y)+1 = d'(s,x)$. Ce qui contredit le choix de x. Donc $\phi(y,x) = \gamma(y,x)$ et (y,x) n'est pas une arête du graphe d'amélioration de G ; mais comme (x,y) en est une dans celui de G', il faut donc que (x,y) soit une arête du chemin amélioré dans le passage de G à G'. Alors et comme ce chemin est de longueur minimale : $d(s,x) = d(s,y) - 1 \leq d'(s,y) - 1 = d'(s,x) - 2 < d'(s,x)$. Contradiction.

Appelons *phase* une succession d'augmentations du flot pendant lesquelles la longueur des chemins améliorés reste fixe. Le nombre de phases est au plus le nombre n de sommets du graphe. De plus dans chaque phase une arête (x,y) ne peut être critique (déterminant l'augmentation du chemin choisi) qu'une seule fois, puisqu'il aura fallu modifier entre temps (y,x) pour permettre un nouvel usage de (x,y) : du même coup x s'éloigne de s et la longueur des chemins croît. Il y a donc $O(nm)$ améliorations apportées au flot, d'où le théorème :

THÉORÈME 3.2. *Le coût de l'algorithme de Edmonds-Karp est $O(nm^2)$ opérations.*

En effet le coût de la recherche en largeur d'un chemin améliorable est proportionnel au nombre d'arêtes du graphe.

4. Couplage dans un graphe biparti

Soit $G = (X,E)$ un graphe symétrique, un *couplage* (on dit *matching* en anglais) est un sous-ensemble C de E tel que deux arêtes de C n'aient pas de sommet commun. Un exemple de couplage est donné sur la figure ci-dessous (les arêtes du couplage sont en traits gras). Dans la suite on dira qu'un sommet est *saturé* s'il appartient à une arête du couplage.

Une remarque importante sur le couplage est la suivante :

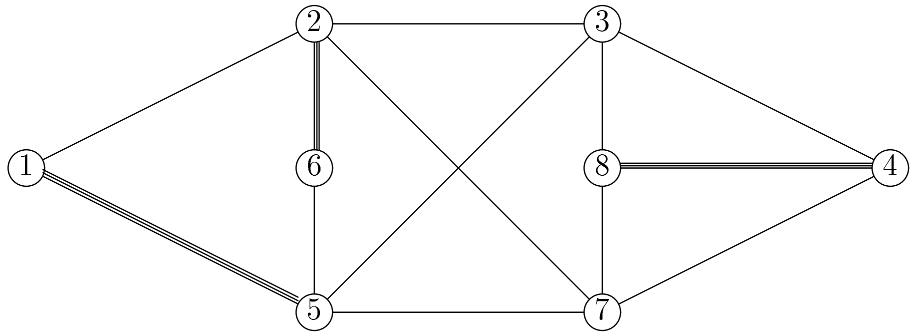


FIG. 4. Un couplage (non maximum)

Remarque Soient C et D deux couplages de $G = (X, E)$, alors la différence symétrique $C \oplus D$:

$$C \oplus D = \{e \in E | e \in Ce, \notin D\} \cup \{e \in E | e \in D, e \notin C\}$$

est formée de chemins et de cycles. Chacun de ces chemins ou cycles est composé alternativement d'une arête de C et d'une arête de D. Si $|C| > |D|$ alors il existe une composante connexe du graphe $(X, C \oplus D)$ constituée d'un chemin de longueur impaire $e_1, e_2, \ldots, e_{2p+1}$ tel que $e_1, e_3 \ldots e_{2p+1} \in C$, et$e_2, e_4, \ldots, e_{2p} \in D$. Une des extrémités de e_1 et une de e_{2p+1} ne sont incidents à aucun arc de D. On dit que le chemin est augmentable pour D, car on peut augmenter le nombre d'éléments de D en le remplaçant par $D' = D \cup \{e_1, e_3, \ldots, e_{2p+1}\} \setminus \{e_2, e_4, \ldots, e_{2p}\}$.

Dans ce paragraphe on s'intéresse à la recherche d'un couplage maximum dans le cas où le graphe G est biparti. L'ensemble des sommets est alors composé de deux sous-ensembles disjoints X et Y et chaque arête joint un sommet de X et un sommet de Y. Les graphes bipartis modélisent le problème de *l'affectation*. Un exemple typique d'un tel problème est celui de l'affectation de stages à des étudiants, l'ensemble X représente les étudiants, Y les stages proposés et les arêtes du graphe relient chaque étudiant aux stages qui font partie de sa liste de choix. Le but est d'obtenir un couplage maximal dans lequel chaque étudiant est couplé avec un stage qui fait partie de sa liste.

Dans le cas particulier des graphes bipartis la recherche d'un couplage maximal dans un graphe $G = (X, E)$ se ramène à celle d'un flot maximum dans un graphe orienté G' qui se déduit de G de la façon suivante : L'ensemble des sommets de G' est $X' = X \cup Y \cup \{s, t\}$ où s et t sont deux nouveaux sommets. L'ensemble des arcs de G' est constitué des arêtes de G orientées de X vers Y, d'arcs ayant pour origine s et pour extrémité chaque sommet de X, d'arcs reliant chaque sommet de Y à t. On suppose de plus que toutes les capacités sont égales à 1. Il est assez facile de vérifier qu'à tout couplage C dans G on associe un flot ϕ_C dans G' en posant $\phi_c(u, v) = 1$ si et seulement l'une des conditions suivantes est satisfaite :

- $\{u, v\} \in C$
- $u = s$ et $v \in X$ est couplé (il existe $y \in Y$ tel que $\{v, y\} \in C$

– $v = t$ et $u \in Y$ est couplé (il existe $x \in X$ tel que $\{x, u\} \in C$
si aucune de ces conditions n'est vérifiée $\phi_C(u, v) = 0$. On vérifie que ϕ_C est un flot et que sa valeur est égale au nombre d'arêtes dans C, réciproquement tout flot ϕ dans G' détermine un couplage de G.

La notion de chemin améliorable se traduit simplement en terme de couplage. Un chemin $P = e_1, e_2, \ldots e_k$ est *alternant* si $e_{2i+1} \notin C$ et si $e_{2i} \in C$ pour $i \leq k/2$. En reprenant la notion de sommet saturé (extrémité d'une arête du couplage) :

THÉORÈME 3.3. *Un couplage est maximal si et seulement s'il n'existe pas de chemin alternant reliant deux sommets non saturés.*

Preuve : Si $P = e_1, e_2, \ldots e_k$ est un chemin alternant reliant deux sommets non saturés alors

$$C \oplus P = C \cup \{e_1\} - e_2 \cup \{e_3\} - e_4 \ldots \cup \{e_k\}$$

est un couplage contenant une arête de plus que C. Réciproquement, soit C un couplage non maximal et soit D un couplage tel que $|D| > |C|$ alors d'après la remarque préliminaire $C \oplus D$ contient une composante connexe qui est chemin alternant les arêtes de C et D où il y a une arête de plus dans C que dans D. Ce chemin constitue un chemin alternant reliant deux sommets non saturés dans C.

Le théorème et sa preuve suggèrent l'algorithme suivant

```
Initialiser C par l'ensemble vide
Tant qu'(il existe un chemin P ameliorable)
       { Ameliorer (P);}
```

5. Mariages stables.

Une question qui généralise celle du couplage est celle de la recherche de couplages (ou mariages) *stables*. Cette notion constitue un chapitre important de la théorie des jeux.

Les données du problème sont un peu plus complexes que celles du simple couplage, on se donne deux ensembles $X = \{x_1, x_2, \ldots, x_p\}$ et $Y = \{y_1, y_2, \ldots, y_p\}$, pour chaque $i = 1, \ldots, p$ une liste ordonnée F_i d'éléments de Y et pour chaque $j = 1, \ldots, q$ une liste ordonnée G_j d'éléments de X. Ces listes ne sont pas vides mais ne contiennent pas nécessairement tous les éléments de Y et de X respectivement. On note $y \prec_i y'$ si y est avant y' dans la liste F_i ou si y' ne figure pas dans la liste F_i alors que y s'y trouve. Un couplage est alors un sous-ensemble de couples (x, y) tel que y figure dans la liste F_x et x dans la liste G_y. La recherche d'un couplage maximal consiste à travailler dans le graphe biparti (X, Y, E) où les arêtes sont tous les couples $\{x, y\}$ tels que y figure dans la liste F_x et x dans la liste G_y, et à y appliquer les algorithmes vus au paragraphe **4**.

Pour introduire la notion de stabilité il est commode de considérer un couplage comme une application partielle α de X dans Y, on note alors β son inverse, ainsi les éléments du couplage sont les couples $(x, \alpha(x))$ ou de manière équivalente les couples $(\beta(y), y)$. Un couplage C est stable si et seulement si pour tout couple x, y qui n'est pas dans C on a

$$\beta(y) \prec_y x \qquad \text{ou} \qquad \alpha(x) \prec_x y$$

On interprète ceci en disant qu'un couplage est stable s'il n'existe pas deux individus qui se préfèrent mutuellement à leur conjoints respectifs.

Exemple : Considérons le cas $p = q = 3$ et les listes

$$\begin{array}{lcccc}
F_1 & : & y_2 & y_1 & y_3 \\
F_2 & : & y_1 & y_3 & y_2 \\
F_3 & : & y_1 & y_2 & y_3
\end{array}
\qquad\qquad
\begin{array}{lcccc}
G_1 & : & x_1 & x_3 & x_2 \\
G_2 & : & x_3 & x_1 & x_2 \\
G_3 & : & x_1 & x_3 & x_2
\end{array}$$

Le couplage $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ n'est pas stable en raison de x_1 et y_2 qui se préfèrent à y_1 et x_2 respectivement. Par contre le couplage $(x_1, y_1), (x_2, y_3), (x_3, y_2)$ est stable.

On montre que l'algorithme suivant donne un couplage stable.

On initialise $\alpha(x)$ au premier élément de la liste F_x et $\beta(y)$ indéfini. Ensuite on effectue des modifications itérativement de β et de α comme suit :

(1) Pour chaque y choisir l'élément x le plus intéressant dans la liste G_y parmi les x tels que $\alpha(x) = y$. et poser $\beta(y) = x$
(2) Pour chaque x,
   - S'il existe y tel que $\beta(y) = x$, alors $\alpha(x) = y$.
   - S'il n'existe pas de y tel que $\beta(y) = x$ et si $\alpha(x)$ n'est pas le dernier élément de la liste F_x alors remplacer $\alpha(x)$ par l'élément qui le suit dans dans la liste F_x,
   - Si aucune des deux conditions précédentes n'est remplie alors $\alpha(x)$ est indéfini.

6. Couplages et intersection de matroïdes

Soit $G = (X, Y, E)$, un graphe biparti et soit $\mathcal{C}$ la famille des sous-ensembles C de E qui sont des couplages. Alors la famille $E, \mathcal{C}$ ne constitue pas un matroïde. En effet, considérons le graphe biparti G composé de 4 sommets $1, 2, 3, 4$, $X = \{1, 3\}$ et $Y = \{2, 4\}$, et de 3 arêtes $(1, 2), (2, 3), (3, 4)$, alors sur les ensembles $C_1 = \{(1, 2), (3, 4)\}$ et $C_2 = \{(2, 3)\}$ on constate que C_1 possède un élément de plus que C_2 et pour tout $e \in C_1$, $C_2 \cup \{e\}$ n'est pas un couplage ce qui contredit les axiomes des matroïdes. Par contre on a :

PROPOSITION 3.4. *La famille $\mathcal{C}$ est l'intersection de deux matroïdes.*

Preuve : Considérons les deux familles de parties $\mathcal{F}$ et $\mathcal{G}$ de E telles que $F \in \mathcal{F}$ si et seulement si deux arêtes de F n'ont pas une extrémité commune dans X. De même on définit $\mathcal{G}$ comme l'ensemble des parties G de E telles que $G \in \mathcal{F}$ si et seulement si deux arêtes de G n'ont pas une extrémité commune dans Y. Il est alors clair que C est un couplage si et seulement si $C \in \mathcal{F}$ et $C \in \mathcal{G}$.

L'algorithme de recherche du couplage maximum se généralise pour déterminer étant donnés deux matroïdes $(E, \mathcal{F})$ et $(E, \mathcal{G})$ l'ensemble C de cardinal maximal qui appartienne à la fois à $\mathcal{F}$ et $\mathcal{G}$. On utilise la notion de *circuit* dans un matroïde $(E, \mathcal{F}$. Rappelons qu'un ensemble C est un circuit de $(E, \mathcal{F})$ si $C \notin \mathcal{F}$ et si tout sous-ensemble strict de C est dans $\mathcal{F}$. Il est alors clair que tout sous ensemble U de E qui n'est pas dans $\mathcal{F}$ contient un circuit.

Pour obtenir le sous-ensemble de cardinal maximal de on définit la notion suivante : Soit F un ensemble qui appartient à $\mathcal{F}$ et $\mathcal{G}$, on associe à une suite d'éléments $e_1, e_2, \ldots, e_p$ les circuits C_i et D_i. C_i est un circuit inclus dans $F \cup \{e_i\}$ pour le matroïde $(E, \mathcal{F})$; il est défini chaque fois que cet ensemble n'est pas dans $\mathcal{F}$. Symétriquement D_i est un circuit inclus dans $F \cup \{e_i\}$ pour le matroïde $E, \mathcal{G}$; il est défini chaque fois que cet ensemble n'est pas dans $\mathcal{G}$.

Une suite $P = \{e_1, e_2, \ldots e_p\}$ est augmentante si les conditions suivantes sont satisfaites :
- $F \cup \{e_1\} \in \mathcal{F}$
- Pour tout i pair compris entre 2 et k $e_i \in F \cap D_{i-1}$. De plus e_i n'appartient pas à D_j pour j impair strictement plus petit que $i-1$.
- Pour tout i impair compris entre 3 et k, $e_i \notin F$ et $e_{i-1} \in C_i$. De plus e_{j-1} n'appartient pas à C_i pour j impair strictement plus petit que i.
- $F \oplus P$ est un élément de $\mathcal{G}$.

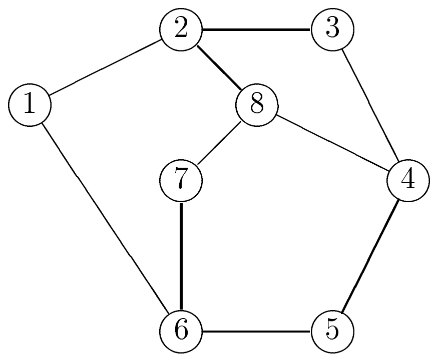
On constate alors que $F \oplus P$ appartient aussi à $\mathcal{F}$ et possède un élément de plus que F. L'itération de cette augmentation donne un ensemble appartenant à l'intersection de taille maximum.

7. Couplages généraux

On considère maintenant des graphes généraux, pour lesquels on veut déterminer un ensemble d'arêtes deux à deux disjointes et de cardinalité maximale. On généralise ainsi les problèmes d'affectation, en prenant en compte par exemple les problèmes de tournée (postier chinois) ou de chaînes. Des applications typiques sont les transitions de phase en magnétisme, le regroupement de particules dans des phénomènes de désintégrations simultanées, etc.

7.1. Principes. Étant donné un graphe *non orienté* $G = (X, A)$, on appelle couplage un sous-ensemble C de A tel que deux arêtes quelconques de C n'ont pas de point commun. Le couplage C étant donné, on dit qu'il *sature* les sommets extrémités de ses arêtes.

Appelons *chaîne* de G toute suite d'arêtes formant un chemin sans boucle. Une chaîne S est dite *alternée* par rapport à un couplage C si ses arêtes sont successivement dans C et dans $A \backslash C$. Supposons de plus que S soit telle que ses extrémités sont ou bien insaturées ou bien extrémité d'une arête commune à C et S. Alors la différence symétrique $C' = C \Delta S$ forme encore un couplage, dans lequel les arêtes de S ont simplement échangé leur rôle. Cette opération est appelée *transfert* le long de S. On remarquera que si les *deux* extrémités de S sont insaturées le nouveau couplage C' est plus riche d'une arête. Une telle chaîne est dite *augmentante*.



Exemple : Donner une chaîne augmentante sur le graphe suivant où le couplage est formé des arêtes $\{2,3\}, \{4,5\}, \{6,7\}$.

7.2. Couplages maximaux. Un théorème de Berge caractérise les couplages maximaux. Démontrons d'abord un lemme technique.

LEMME 3.1. *Soient C et C' deux couplages distincts d'un graphe $G = (X, A)$. Soit $B = C \oplus C'$ et $\Gamma = (X, B)$ le graphe construit sur cet ensemble B d'arêtes. Les composantes connexes de Γ sont soit des points isolés, soit des cycles ou des chaînes alternés d'arêtes de C et C'.*

Preuve : Le degré maximal de Γ est 2.

Ce lemme permet de prouver la condition suffisante.

THÉORÈME 3.4. *Un couplage C est maximal si et seulement s'il n'existe pas de chaîne alternée augmentante.*

Preuve : Compte tenu de la remarque ci-dessus l'existence d'une chaîne alternée augmentante permet de construire un couplage plus grand par transfert le long de cette chaîne, ce qui réfute l'hypothèse de maximalité.

Réciproquement, soit C n'ayant pas de chaîne alternée augmentante et soit C' un couplage maximal (donc n'ayant pas de chaîne alternée augmentante). Considérons le graphe d'arêtes $B = C \Delta C'$ évoqué dans le lemme. Il ne peut posséder de chaîne alternée impaire et donc C est maximal.

On remarquera enfin qu'un couplage maximal ne peut laisser qu'un seul sommet insaturé et que les couplages maximaux s'échangent par transfert sur des cycles ou des chaînes alternées pairs.

7.3. Algorithme de calcul d'un couplage maximal. Cet algorithme est relativement complexe car il repose sur l'assemblage de couplages locaux et sur de nombreux transferts. On retrouvera donc deux types de recherche sur des cycles et des chaînes ; on essaiera d'organiser la croissance des couplages selon des arbres alternés reliant des cycles. Dans cette perspective, le concept de contraction d'un graphe est utile.

Soit $G = (X, A)$ un graphe ; pour $X' \subset X$ on définit un nouveau graphe $G' = G/X'$ dont l'ensemble des sommets est $X \setminus X' \cup \{\star\}$ et les arêtes sont les mêmes en dehors de X' et pointent toutes sur le nouveau sommet $\star$ sinon. Le sommet $\star$ représente tous les sommets de X'.

L'algorithme s'appuie sur les propriétés suivantes.

LEMME 3.2. *Soit $X' \subset X$ un ensemble de sommets formant un cycle impair K. Si C est un couplage de G/X' il existe un couplage maximal C_K de K tel que $C \cup C_K$ est un couplage de G.*

LEMME 3.3. *Soit C est un couplage laissant deux sommets insaturés. Soit K un cycle impair sur les sommets X' et C_K un de ses couplages maximaux. Alors C est un couplage maximal de G si et seulement si $C \setminus C'$ est un couplage maximal de G/X'.*

Le lemme 2 est immédiat. Le lemme 3 se prouve en considérant la nature de la chaîne alternée qui mène du sommet non saturé de K à l'autre sommet non saturé de G.

7.4. Recherche des arbres alternés. Le problème est de déterminer dans le cas d'un couplage qui laisse plusieurs sommets insaturés s'il existe un chemin augmentant ou non. Le concept d'arbre alterné étend immédiatement celui de chaîne : chacune des branches de l'arbre est une chaîne alternée de longueur paire et la racine est un sommet insaturé (de fait le seul pour cet arbre).

Le processus construit progressivement des arêtes de l'arbre tout en contractant le graphe.

L'algorithme détaillé est un peu complexe et se trouve dans les ouvrages de référence.

CHAPITRE 4

Programmation linéaire

Nous nous attaquons à l'un des chapitres les plus classiques de l'optimisation combinatoire.

Le problème traité peut se présenter sous la forme imagée suivante : une entreprise fabrique des produits $P_1, P_2, \ldots, P_n$ à l'aide de matières premières $M_1, M_2, \ldots, M_m$. Chaque unité du produit P_j rapporte un bénéfice c_j et utilise $a_{i,j}$ quantité de la matière première M_i ; la matière M_i est disponible au total en quantité b_i pour l'ensemble des produits. La question question posée consiste à déterminer pour tout j la quantité x_j de produit P_j à produire de façon à réaliser un bénéfice maximal tout en satisfaisant aux contraintes de quantité.

On doit ainsi maximiser $\sum_{j=1}^{n} c_j x_j$ en satisfaisant les m contraintes $\sum_{j=1}^{n} a_{i,j} x_i \leq b_j$. On voit bien que cette question se rencontre très souvent dans la pratique, d'ailleurs les equipes de chercheurs qui ont réalisé des logiciels efficaces dans ce domaine ont obtenu d'excellents résultats commerciaux.

George B. Dantzig a été un des pionniers du domaine et a le premier proposé une méthode de résolution pour ce qui reste un des grands paradigmes de la modélisation et de la programmation (ce terme désigne dans le contexte l'organisation des calculs et non pas la réalisation d'un programme informatique).

Si l'on se place d'un point de vue de mathématicien, il s'agit de trouver un point dans un polyèdre de R^n (donné par un système de m inéquations) point qui optimise une fonction linéaire des coordonnées (appelée fonction économique). Ce problème peut donc paraître particulièrement élémentaire à ceux des mathématiciens qui ne se posent pas la question de connaître l'efficacité des méthodes de calcul permettant d'obtenir la solution d'un problème.

En effet, quelques considérations simples montrent que si le polyèdre est borné, l'optimum est atteint sur un de ses sommets. Il suffirait donc de résoudre tous les systèmes non singuliers de n équations tirées de ces m inéquations, et après avoir vérifié que la solution satisfait les autres inéquations, de tabuler la valeur de la fonction économique en ce point. Le problème est alors terminé en recherchant la valeur maximale de la table.

Pour ce qui concerne notre point de vue algorithmique, on ne saurait se contenter de cette façon de traiter le problème car il faut se poser des questions de complexité, l'algorithme envisagé demande un nombre exponentiel de calculs : on doit résoudre un nombre de systèmes égal au coefficient binomial $\binom{m}{n}$ qui croît exponentiellement avec m et n.

Après plusieurs années de recherches dans ce domaine on a pu montrer que ce problème est exemplaire car il figure parmi les cas limites qui incarnent les barrières et sauts de complexité, on a longtemps pensé qu'il n'existait pas d'algorithme polynomial pour le résoudre et maintenant on sait que tout en étant polynomial il reste un des plus compliqués de sa catégorie. Par ailleurs, l'algorithme non polynomial classique, la *méthode du simplexe*, reste expérimentalement excellent (Dan Spielman a d'ailleurs récemment montré qu'avec une certaine distribution de probabilité des données la méthode est en moyenne polynomiale). Par contre les méthodes polynomiales demandent un gros effort de compréhension des géométries sous-jacentes et sont pour les valeurs usuelles de m et n souvent moins efficaces en pratique que l'algorithme du simplexe. Il faudrait que m et n croissent considérablement pour qu'elles deviennent meilleures que la méthode du simplexe, mais dans ce cas toutes les méthodes connues sont impratiquables.

1. Présentation du problème

Etant donnés des nombres $a_{i,j}, b_i, c_j$, pour $1 \leq i \leq m,\ 1 \leq j \leq n$, soient $x_1, x_2, \ldots, x_n$ des inconnues astreintes à un ensemble de m contraintes pour $1 \leq i \leq m$:

$$\sum_{1 \leq j \leq n} a_{ji} x_j \leq b_i$$

On peut supposer que l'on travaille soit avec des réels soit avec des rationnels, les résultats théoriques donnés ci-dessous sont valables dans les deux cas ; toutefois, lorsqu'on s'interesse aux questions de complexité, il vaut mieux se placer dans le corps des rationnels car on peut alors mesurer la taille des données par le nombre de bits nécessaires pour représenter le numérateur et le dénominateur.

On cherche dans le domaine ainsi défini les points qui optimisent une forme linéaire, appelée fonction économique ou fonction d'objectif,

$$\gamma(x) = \sum_{j=1}^{n} c_j x_j$$

On écrit usuellement le problème de façon synthétique au moyen d'une notation matricielle :

$$(\mathcal{P}) \begin{cases} \max\ cx \\ Ax \leq b \end{cases}$$

où $c, x \in \mathtt{R}^n$, $b \in \mathtt{R}^m$ et $A \in \mathtt{R}^{m \times n}$.

Dans la suite on adopte la notation pratique qui consiste à ecrire $u \leq v$ pour deux vecteurs de $\mathtt{R}^n$ pour dire que pour tout $1 \leq j \leq n$ on a $u_i \leq v_j$, ainsi $u \geq 0$ signifiera que toutes les coordonnées de u sont positives ou nulles.

On appellera *fonction économique* la fonction qui au point x de $\mathtt{R}^n$ associe le réel cx.

Géométriquement, chaque contrainte défini un hyperplan, l'ensemble de points qui satisfont la condition $Ax \leq b$ est alors l'intersection d'un nombre fini de demis espaces définis par les hyperplans donnés par les contraintes. On appelle un tel domaine un *polyèdre* le problème se traduit en la recherche du maximum de la forme linéaire $\gamma(x) = cx$ à l'intérieur du polyèdre R donné par $Ax \leq b$, polyèdre que l'on appelera *polyèdre des contraintes.*

2. Principes généraux

2.1. Remarques géomètriques. On note R l'ensemble des points de $\mathtt{R}^n$ qui satisfont les contraintes :

$$Ax \quad \leq \quad b.$$

Ainsi R est un *polyèdre*, c'est à dire un domaine de $\mathtt{R}^n$ intersection d'un nombre fini de demis espaces, on appelle habituellement *polytope* un polyèdre borné [1].

Un polyèdre est un ensemble convexe, c'est à dire que si $x, y \in R$ alors le segment d'extrémités x, y, c'est à dire l'ensemble des points $\lambda x + (1 - \lambda y)$, pour λ réel compris entre 0 et 1, est inclus dans R. Cette propriété de convexité simplifie grandement la recherche d'un optimum comme on le verra dans le théorème qui suit.

Dans la suite on utilise les définitions suivantes : une *face* F du polyèdre R est un ensemble de points de R qui n'est coupé par aucun des segments ayant pour extrémités deux points de R, ainsi un segment qui contient un point de F (en dehors de ses extrémités) est inclus dans F,

Un *sommet* est une face réduite à un point. C'est un point de R qui n'appartient à aucun segment de R dont il n'est pas une des extrémités.

On vérifie qu'un sommet est l'intersection de n parmi les m hyperplans qui définissent R (donnés par les lignes de la matrice A). On dit que deux sommets sont *adjacents* si le segment qui les joint est une face, il est alors appelé arête du polyèdre.

Les propriétés suivantes du système $\mathcal{P}$ permettront de donner les principales idées d'un algorithme pour la résolution.

PROPOSITION 4.1. *L'optimum du système $\mathcal{P}$, se trouve ou bien à l'infini ou bien est atteint par l'un des sommets du polyèdre. Un sommet x qui n'est pas optimal est adjacent à un sommet y tel que $cy \geq cx$.*

Idée de la preuve :

Soit x_o un point où l'optimum est atteint. Considérons l'hyperplan H qui passe par x_0 et dont la direction est définie par c. En raison de l'optimalité de x_0, le polyèdre R est situé d'un même coté de H. Ainsi H contient une face de R et donc un sommet.

D'autre part si x est un sommet de R, soit H l'hyperplan passant par x et de direction définie par c. Si tous les sommets y adjacents à y sont d'un meme coté de H alors il en est de même de tous le points de R, ainsi l'optimum est atteint en x.

$\square$

[1] C'est aussi l'enveloppe convexe d'un nombre fini de points

2.2. Grandes lignes d'un algorithme. La proposition précédente nous permet de donner un canevas d'un algorithme de recherche de l'optimum (c'est en fait un pseudo-algorithme car la terminaison n'en est pas nécessairement assurée) de résolution du problème $\mathcal{P}$.

Pour un point P on note $\gamma(P)$ la valeur de la fonction économique en P.

1. Si R est vide Alors pas de solution Sinon déterminer un sommet P_0 du polyèdre des contraintes.

$i = 0$;
2. Tant qu'il existe un sommet Q adjacent à P_i tel que $\gamma(Q) \geq \gamma(P_i)$ faire
$\{P_{i+1} = Q$; ++i;
Si on trouve une direction où $\gamma(x)$ tend vers l'infini alors arrêt. $\}$

Il reste encore pas mal de points à préciser pour transformer ceci en un vrai algorithme (en particulier qui termine), toutefois la correction du principe est évidente car il s'agit d'une transposition quasi mot à mot de la proposition ci-dessus. 4.1

Enumérons les questions auxquelles il faudra répondre :

– Comment déterminer si le domaine R est vide ? On peut noter que si tous les coefficients de b sont positifs, ce qui est souvent le cas dans la pratique, alors l'origine est un point de R puisque dans ce cas $Ax = 0 \leq b$. La question se pose donc si b contient des composantes négatives.

– Trouver un sommet de R. Là encore si tous les coefficients de A, b et de c sont positifs alors le problème est aisé, car il suffit d'ajouter les contraintes $x \geq 0$ qui seront satisfaites par toute solution optimale ; dans ce nouveau problème, qui admet un même optimum que le précédent, l'origine est un sommet de polyèdre. Si A et c ont des coefficients négatifs mais que ce n'est pas le cas pour b, il faut alors doubler le nombre de variables en posant $x_i = y_i - z_i$, l'ajout des contraintes $y_i, z_i \geq 0$ ne modifie pas l'optimum et cette fois encore l'origine est un sommet du polyèdre. Si b a certains coefficients négatifs, la question devient plus complexe, on verra plus loin comment la résoudre en traitant un problème linéaire auxiliaire pour lequel on connaîtra un sommet.

– Déterminer si l'on peut trouver une direction ou la fonction cx peut tendre vers l'infini à l'interieur de R. Cette question difficile sera traitée au paragraphe suivant dans le cas où $b = 0$ et généralisée par la suite.

– Comment éviter de boucler indéfiniment sur des sommets ayant même valeur de la fonction économique ? Cette question sera aussi traitée au paragraphe suivant.

– Trouver une méthode simple à programmer permettant de parcourir tous les sommets du polyèdre adjacents à un sommet donné et trouver l'un de ses voisins qui améliore la fonction économique. Ceci est l'objet de l'algorithme du simplexe, il sera décrit dans un paragraphe qui lui est consacré.

3. Le cas $b = 0$

Dans le cas où tous les b_i sont nuls le problème se simplifie considérablement, on évite en effet toutes les questions liées au parcours des sommets du polyèdre et on se limite à déterminer s'il n'y a pas une direction où la fonction économique tend vers l'infini.

On note en effet que l'origine est un point qui satisfait les contraintes, de plus la valeur de la fonction économique est nulle à l'origine. On peut aussi noter que s'il existe un x tel que $cx > 0$ et $Ax \leq 0$ alors pour tout réel $\lambda > 1$ on a $A\lambda x \leq 0$ et $c\lambda x = \lambda cx > cx$, ainsi la valeur de cx peut être rendue aussi grande que l'on veut.

On peut résumer ces remarques en notant qu'il y a deux situations possibles lorsque $b = 0$; ou bien la solution optimale est l'origine et l'optimum est 0, ou bien il existe des points à l'intérieur du polyèdre qui donnent une valeur aussi grande que l'on veut à cx.

Etant donnée une matrice A et un vecteur c on cherche donc à savoir s'il existe un u tel que $Au \leq 0$ et $cu > 0$.

Noter que si c n'est pas une combinaison linéaire des lignes $L_1, L_2, \ldots, L_m$ de A, on peut trouver un tel u. Il suffit en effet de prendre u orthogonal à l'espace engendré par les lignes de A et non orthogonal à c, avec $uc > 0$.

On se contente donc d'examiner le cas où c est combinaison linéaire des lignes de A. On est alors aidé par le théorème suivant dont la preuve contient un algorithme de décision.

THÉORÈME 4.1. *Soit A une matrice de $\mathtt{R}^{m,n}$ et c un vecteur de $\mathtt{R}^n$ combinaison linéaire des lignes de A alors l'une des deux assertions exclusives suivantes est satisfaite :*

$$(1) \qquad \exists u, \quad \text{tel que} \quad cu > 0 \quad \text{et} \quad Au \leq 0$$

$$(2) \qquad \exists y \geq 0, \quad \text{tel que} \quad yA = c$$

.

DÉMONSTRATION. Il est assez facile de vérifier que les deux assertions ne peuvent ps être toutes les deux satisfaites, en effet si (2) est satisfaite, alors pour tout u tel que $Au < 0$ on a, puisque $y > 0$, $yAu < 0$ mais comme $yA = c$ cela donne $cu < 0$ et (1) ne saurait être satisfaite.

La preuve du théorème est ensuite obtenue en utilisant l'algorithme suivant :

SOIT I UN ENSEMBLE DE D'INDICES TEL QUE $L_i, i \in I$ FORME UNE BASE DE L'ESPACE ENGENDRÉ PAR LES LIGNES DE A, EXPRIMER c COMME UNE COMBINAISON LINÉAIRE DES LIGNES L_i, $i \in I$.

$$c = \sum_{i \in I} y_i L_i$$

TANT QU'IL EXISTE $y_i < 0$ FAIRE {

SOIT h LE PLUS PETIT INDICE TEL QUE $y_h < 0$ ET SOIT u LE VECTEUR DE R^n TEL QUE $L_j u = 0$ POUR $j \in I, j \neq h$ ET $L_h u = -1$

SI $L_i u \leq 0$ POUR $i = 1, \ldots, m$ ALORS u EST LE VECTEUR CHERCHÉ ; EXIT.

SINON CHOISIR s LE PLUS PETIT TEL QUE $L_s u > 0$;

POSER $I := I - \{h\} \cup \{s\}$ ET EXPRIMER c COMME PLUS HAUT.
}

DÉMONSTRATION. Noter que l'algorithme calcule à chaque étape des y_i et un u car la sous matrice des lignes dont les indices sont dans I est toujours non singulière. Cet u satisfait $cu > 0$ à chaque étape puisque $cu = y_h L_h u$. Si l'algorithme s'arrête c'est que l'on a obtenu un y satisfaisant (2) ou un u satisfaisant (1).

Il nous reste donc à montrer que l'algorithme termine en vérifiant à chaque étape que le sous ensemble I calculé est différent de ceux calculés avant lui. Supposons qu'au contraire on obtienne le même I à deux étapes différentes, montrons que l'on aboutit à une contradiction. Notons r le plus grand indice qui sort de I entre ces deux étapes et posons $I = I_p$ au moment où r sort, comme on retrouve le même I plus tard c'est que r est aussi rentré entre les deux étapes considérées, notons I_q la valeur de I lorsque r entre dans I.

Posons $c = \sum_{i \in I_p} y_i L_i$ et soit u' la valeur de u trouvée lorsque r entre dans I, on a alors $u'c > 0$.

Mais on vérifie que pour $i \in I_p,\ i < r$ on a $y_i > 0$ puisque r étant sorti de $I = I_p$ c'est le plus petit tel que $y_r < 0$, de plus $u'L_i < 0$ puisque r, lors qu'il entre dans I_q est le plus petit tel que $u'L_r > 0$. Ainsi pour $i \in I_p,\quad i \leq r$ on a $y_i u' L_i \leq 0$.

D'autre part si $i \in I_p,\ i > r$, puisque r est le plus grand indice sorti, i est aussi dans I_q ; et u' a été choisi de façon telle que $L_i u' = 0$. Ainsi pour $i \in I_p,\quad i > r$ on a $y_i u' L_i = 0$.

On obtient la contradiction en vérifiant que $0 < u'c = \sum_{i \in I_p} y_i u' L_i \leq 0$. □

Le corollaire qui suit est une autre façon de lire ce théorème, il peut être aussi considéré comme une version simplifiée du principe fondamental de dualité qui sera vu dans le paragraphe suivant.

COROLLAIRE 4.1. *Si l'un des deux problèmes suivants*

$$\begin{array}{lll} (\mathcal{P}) & \max & \{cx \mid Ax \leq 0\} \\ (\mathcal{D}) & \exists y, & y \geq 0 \mid yA = c \end{array}$$

admet une solution finie alors il en est de même pour l'autre et $0 = \max\{cx \mid Ax \leq 0\}$.

DÉMONSTRATION. Si $\max\{cx \mid Ax \leq 0\}$ admet une solution finie alors elle est telle que $cx = 0$, en effet $x = 0$ est tel que $Ax \leq 0$ et donne $cx = 0$. S'il existe un x tel que $Ax \leq 0$ et $cx > 0$ alors en multipliant x par un réel $\lambda > 1$ on vérifie que l'ensemble des valeurs de cx n'est pas borné et il n'y a pas de solution finie. Le théorème 4.1 montre enfin que l'existence d'un $y \geq 0$ tel que $yA = c$ est équivalente à $Ax \leq 0 \Rightarrow cx \leq 0$. □

Le corollaire qui suit nous sera utile dans la suite :

COROLLAIRE 4.2. *Le système d'inéquations* $Ax \leq b$ *admet une solution si et seulement si pour tout* y *:*

$$y \geq 0,\ yA = 0 \ \Rightarrow\ yb \geq 0$$

DÉMONSTRATION. On considère la matrice $A' = [Id\ A\ -A]$.

On montre d'abord que $Ax \leq b$ a une solution si et seulement s'il existe $u \geq 0$ tel que $A'u = b$. En effet, s'il existe un $u \geq 0$ tel que $A'u = b$ alors u s'écrit $[v\ u'\ u'']$ et $x = u' - u''$

satisfait $Ax + v = b$ soit $Ax \leq b$. Réciproquement si x satisfait $Ax \leq b$ on peut poser $x = u' - u''$ avec $u' \geq 0, u'' \geq 0$, en notant $v = Ax - b$ on observe que $[v\ u'\ u'']A' = b$.

D'après le corollaire 4.1 $A'u = b$ admet une solution $u > 0$, si et seulement si $yA' \leq 0$ implique $yb \leq 0$. Mais comme les colonnes de A' sont consistuées de celles de A, leurs opposées et celles de la matrice identité, $yA' \leq 0$ est équivalent à $y \leq 0$ et $yA = 0$ ce qui permet de conclure. □

4. Le cas b quelconque

4.1. Théorème de dualité. On a le résultat fondamental suivant qui permettra de remplacer la recherche d'un maximum par celle d'une solution à un système d'inéquations. Il permet aussi de vérifier, dans de nombreuses situations, que l'on bien obtenu l'optimum à un problèmes.

THÉORÈME 4.2. *Si l'un des deux problèmes suivants*

$$\begin{array}{ll} (\mathcal{P}) & \max\{cx \mid Ax \leq b\} \\ (\mathcal{D}) & \min\{yb \mid y \geq 0,\ yA = c\} \end{array}$$

admet une solution finie alors il en est de même pour l'autre et on a :

$$\min\{by \mid y \geq 0,\ yA = c\} = \max\{cx \mid Ax \leq 0\}.$$

DÉMONSTRATION. Si $\mathcal{P}$ n'admet pas de solution parce que $Ax \leq b$ n'en a pas, alors d'après le corollaire 4.2 il existe $z \geq$ tel que $zA = 0$ et $zb \geq 0$. Ceci montre que pour $\lambda > 1$, l'ajout de λz a toute solution de $yA = c$ donne une autre solution y' avec $y'b < yb$, ce qui interdit une solution finie à $\mathcal{D}$.

Si $\mathcal{P}$ n'admet pas de solution parce qu'il existe x tel que $Ax \leq b$ et cx plus grand que toute valeur donnée, il ne peut y avoir de $y \geq 0$ tel que $yA = c$ car alors $cx = yAx \leq yb$.

Si $\mathcal{D}$ n'admet pas de solution parce que $y \geq 0,\ yA = c$ n'en a pas alors, d'après le théorème du paragraphe précédent, il existe un z tel que $zc \geq 0$ et $Az \leq 0$ et l'ajout de λz pour $\lambda > 1$ à un x qui satisfait $Ax \leq b$ donne $x' = x + \lambda z$ tel que $Ax' \leq b$ et $cx' > cx$.

Si $\mathcal{D}$ n'admet pas de solution parce qu'il existe $y \geq 0$ tels que $yA = c$ et yb plus petit que toute valeur donnée, il ne peut exister de x tel que $Ax \leq b$ car alors $yAx = cx \leq yb$.

On a ainsi vérifié que si l'un des problèmes n'a pas de point admissible, alors l'autre a un optimum qui est infini ; supposons donc que les deux admettent un point admissible notés $\overline{x}$ et $\overline{y}$ respectivement, alors on a :

$$c\overline{x} \quad = \quad \overline{y}A\overline{x}$$

et puisque $\overline{y} \geq 0$ et $A\overline{x} \leq b$ cela donne :

$$c\overline{x} \quad \leq \quad \overline{y}b$$

Ainsi pour démontrer l'égalité entre les deux optima il suffit de vérifier qu'il existe deux points admissibles $\overline{x}'$ et $\overline{y}'$ tels que $\overline{y}'b \leq c\overline{x}'$. Soit dit de manière différente que le système suivant admet une solution :

$$yb \;\leq\; cx, \quad Ax \;\leq\; b, \quad yA \;\leq\; c, \quad -yA \;\leq\; -c, \quad -y \leq 0$$

Ce système s'écrit sous forme matricielle suivante, où ${}^tA$ dénote la transposée de A.

$$\begin{pmatrix} A & 0 \\ -c & {}^tb \\ 0 & {}^tA \\ 0 & -{}^tA \\ 0 & -Id \end{pmatrix} \begin{pmatrix} x \\ {}^ty \end{pmatrix} \;\leq\; \begin{pmatrix} b \\ 0 \\ {}^tc \\ -{}^tc \\ 0 \end{pmatrix}$$

Ce qui peut s'écrire

$$A'z \;\leq\; b'$$

avec $z = (x, {}^ty)$. On utilise alors le corollaire 4.2 ; il faut ainsi vérifier que pour tout $z \geq 0$ tel que $zA' = 0$ on a $zb' \geq 0$.

Posons $z = (u, \lambda, w', w'', v)$ tel que $zA' = 0$, on a ainsi :

$$uA - \lambda c = 0 \qquad \lambda^tB + w'\ {}^tA - w''\ {}^tA - v \;=\; 0$$

Le calcul de zb' donne :

$$zb' \;=\; ub + w'\ {}^tc - w''\ {}^tc$$

Comme λ est un réel positif ou nul on distingue le cas où il est strictement positif de celui où il est nul.

- Si $\lambda > 0$ on a alors :

$$zb' \;=\; \lambda^{-1}\lambda ub + (w' - w'')\ {}^tc$$

 Soit d'après la relation ci-dessus :

$$zb' \;=\; \lambda^{-1}(v - (w' - w'')\ {}^tA)\ {}^tu + (w' - w'')\ {}^tc$$

 Et comme ${}^tA{}^tu = \lambda c$:

$$zb' \;=\; \lambda^{-1}v\ {}^tu$$

 qui est positif puisque λ, v et u le sont.

- Si $\lambda = 0$, on a $uA = 0$ et $v = (w' - w'')\ {}^tA$, de plus

$$zb' \;=\; ub + w'\ {}^tc - w''\ {}^tc$$

 On utilise le fait que les deux systèmes $\mathcal{P}$ et $\mathcal{D}$ ont des points admissibles $\overline{x}$ et $\overline{y}$ tels que : $\overline{y}A = c$ et $A\overline{x} \leq b$, ceci donne :

$$zb' \;=\; ub + (w' - w'')\ {}^tA\ {}^t\overline{y} \;=\; ub + v{}^t\overline{y}$$

 Comme $\overline{y}, v$ et u sont positifs et que $b \geq A\overline{x}$ on a $zb \geq uA\overline{x}$; or $uA = 0$ ce qui termine la preuve.

$\square$

4.2. Algorithme du simplexe. On se propose d'abord de donner un algorithme qui résout le problème $\mathcal{P}$ dans le cas où l'on connaît un sommet du polyèdre des contraintes, c'est un point x tel que $Ax \leq b$ et x est solution d'un système d'équations extrait de la matrice A des contraintes $L_i x = b_i$ pour $i \in I$. On verra plus loin comment calculer une tel point de départ. Ici et dans la suite I désigne un ensemble d'indices de lignes et A_I la sous matrice de A qui contient les lignes dont les indices sont dans I, au cours de tout l'algorithme cette matrice est inversible, ainsi $x = A_I^{-1} b$.

On itère la boucle suivante jusqu'à obtenir $y \geq 0$ ou $v < 0$

Soit y tel que $y_i = (cA_I^{-1})_i$ pour $i \in I$ et $y_j = 0$ pour $j \notin I$.
Si $y \geq 0$ alors x est solution du système.

Sinon soit h le plus petit indice tel que $y_h < 0$ et soit U_h la colonne correspondante de A_I^{-1} ;

déterminer pour chaque ligne L_j de la matrice A la valeur $v_j = -L_j U_h$.

Si $\forall j \notin I,\ v_j < 0$ alors cx est non borné pour $Ax \leq b$

Sinon soit s tel que $\frac{b_s - L_s x}{v_s}$ soit positif et minimum.
On pose alors $I := I - \{h\} \cup \{s\}$ et on calcule x, intersection des hyperplans correspondants.

Preuve de validité de l'algorithme du simplexe :

Si l'algorithme termine en obtenant $y \geq 0$ alors on a bien l'optimum.
En effet, on note J le complémentaire de I et on a :

$$A_I x \ = \ b_I, \quad A_J x \ \leq \ b_J, \quad y_I = cA_I^{-1}, \quad y_J \ = 0$$

Ce qui donne

$$cx \ = \ y_I A_I x \ = \ y_I b_I$$

Mais puisque $y_J = 0$ on obtient $cx = yb$, et y satisfait les contraintes ($yA = c$) du problème dual $\mathcal{D}$; et d'après le théorème de dualité, x et y sont les optima de $\mathcal{P}$ et $\mathcal{D}$.

Si l'algorithme termine en obtenant $L_j U_h \geq 0$, alors l'optimum est à l'infini.
En effet, dans ce cas, la colonne U_h de la matrice A_I^{-1} satisfait :

$$y_h = cU_h < 0 \quad \text{et} \quad AU_h \geq 0$$

Donc pour tout réel $\lambda > 0$ on a

$$A(x - \lambda U_h) \leq b - \lambda A U_h \leq b$$

et $c(x - \lambda U_h) = cx - \lambda c U_h$ peut être rendu arbitrairement grand.

Le passage de I à $I - \{h\} \cup \{s\}$ revient à transformer x en :

$$x' \ = \ x + \frac{b_s - L_s x}{L_s U_h} U_h$$

En effet si $i \in I, i \neq h$ on a :

$$L_i x' = L_i x + \frac{b_s - L_s x}{L_s U_h} L_i U_h$$

mais comme U_h est une colonne de la matrice A_I^{-1} et que $h \neq i$ on a $L_i U_h = 0$ et donc $L_i x' = L_i x = b_i$ D'autre part

$$L_s x' = L_s x + \frac{b_s - L_s x}{L_s U_h} L_s U_h = b_h$$

ainsi x' est bien l'intersection des hyperplans définis par les lignes dont les indices sont dans $I - \{h\} \cup \{s\}$.

Le point x' est un sommet du polyèdre R et améliore la fonction économique.

Il suffit de vérifier que x' satisfait les contraintes; on a pour $j \in J$:

$$L_j x' = L_j x + \frac{b_s - L_s x}{L_s U_h} L_j U_h$$

Mais ou bien $L_j U_h < 0$ et $L_j x' < L_j x \leq b_j$ ou bien $L_j U_h$ satisfait $L_j U_h \geq L_s U_h$ soit :

$$\frac{b_s - L_s x}{L_s U_h} \leq \frac{b_j - L_j x}{L_j U_h}$$

Ce qui donne

$$L_j x' \leq L_j x + (b_j - L_j x) \leq b_j$$

Enfin x' améliore la fonction économique car

$$x' = cx + \frac{b_s - L_s x}{L_s U_h} c U_h$$

et $L_s U_h < 0, c U_h < 0$ implique $cx' \geq cx$.

La preuve de ce que l'algorithme du simplexe termine se fait de la même façon que pour la preuve du théorème 4.1.

5. Un exemple

On considère le problème :

$$\left\{ \begin{array}{lcl} -x_1 & \leq & 0 \\ -x_3 & \leq & 0 \\ 2x_1 + x_2 - x_3 & \leq & 4 \\ 3x_1 - 2x_2 + x_3 & \leq & 2 \\ x_1 - 2x_2 & \leq & 5 \\ -x_1 + x_3 & \leq & 8 \end{array} \right.$$

$$\max x_1 + x_2 + x_3$$

On pose $I = \{1, 2, 3\}$ et on a le point $(0, 4, 0)$ qui est réalisable

$$A_I = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 2 & 1 & -1 \end{pmatrix} \quad {A_I}^{-1} = \begin{pmatrix} -1 & 0 & 0 \\ 2 & -1 & 1 \\ 0 & -1 & 0 \end{pmatrix}$$

Etape 1.

On calcule $c{A_I}^{-1}$ qui vaut $(1,-2,1)$, on devra donc supprimer la ligne 2 la colonne correspondante dans ${A_I}^{-1}$ est alors $\begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}$. Le produit de cette colonne par l'opposée de chacune des lignes de la matrice A donne $0,-1,0,-1,-2,1$. Seul le produit avec la ligne L_6 donne un nombre positif, on fait donc entrer 6 dans I et on obtient : $I = \{1,3,6\}$. On obtient

$$A_I = \begin{pmatrix} -1 & 0 & 0 \\ 2 & 1 & -1 \\ -1 & 0 & -1 \end{pmatrix} \quad {A_I}^{-1} = \begin{pmatrix} -1 & 0 & 0 \\ 1 & 1 & 1 \\ -1 & 0 & -1 \end{pmatrix}$$

Etape 2.

Le point x est égal à $(0,12,8)$

On calcule $c{A_I}^{-1}$ qui vaut $(-1,1,2)$, on devra donc supprimer la ligne 1, la colonne correspondante dans ${A_I}^{-1}$ est alors $\begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}$. Le produit de cette colonne par l'opposée de chacune des lignes de la matrice A donne $-1,-1,0,6,3,0$. Il y a deux valeurs positives pour les lignes 4 et 5, on doit donc calculer :

$$\frac{b_j - L_j x}{v_j}$$

qui vaut 3 pour $j = 4$ et 9.66 pour $j = 5$ on retient donc la ligne 4 et on obtient $I = 3,4,6$.

$$A_I = \begin{pmatrix} 2 & 1 & -1 \\ 3 & -2 & 1 \\ -1 & 0 & -1 \end{pmatrix} \quad {A_I}^{-1} = \frac{1}{6}\begin{pmatrix} 2 & 1 & 1 \\ 4 & -1 & 5 \\ 2 & 1 & 7 \end{pmatrix}$$

Etape 3.

On trouve le point $x = (3,9,11)$ et on a $A_I^{-1}c = (8,1,13)$ qui est positif on a donc atteint l'optimum qui vaut 23 pour ce point x.

6. Obtention d'un sommet du polyèdre

Pour que l'algorithme du simplexe soit complet, il nous faut obtenir un sommet du polyèdre ceci afin d'initialiser le parcours de l'ensemble de ses sommets qui se tremine par la découverte de l'optimum.

Dans le cas où toutes les entrées de A, b et c sont positives ou nulles on remarque immédiatement que le sommet qui atteint l'optimum est aussi à coordonnées positives ou nulles ; on ne modifie donc pas celui-ci si on ajoute les contraintes $x_i \geq 0$. Avec ces nouvelles contraintes l'origine est un des sommet du polyèdre et on peut initialiser la recherche par celle-ci.

Dans le cas où les b_i sont positifs ou nuls mais pas nécessairement les c_j ni les $A_{i,j}$ on dédouble les variables en remplaçant x_i par $x_i - y_i$. On a alors un problème strictement équivalent au précédent pour lequel l'origine $\forall i, x_i = y_i = 0$ est un sommet du polyèdre.

Reste à examiner le problème lorsque certains des b_i sont négatifs. On introduit alors pour chacun des b_i négatifs une nouvelle variable t_i, en plus du dédoublement de x_i en $x_i - y_i$ considéré plus haut.

On écrit alors les contraintes $\mathcal{P}$ sous la forme plus pratique à manipuler suivante (attention dans cette notation A_J est une matrice extraite de $-A$ de même que b_J qui est extrait de $-b$) :

$$\begin{pmatrix} A_I \\ -A_J \end{pmatrix} x \quad \leq \quad \begin{pmatrix} b_I \\ -b_J \end{pmatrix} \qquad x \geq 0$$

Après avoir introduit les variables t_i on considère le nouveau problème linéaire $\mathcal{P}'$ suivant où J désigne l'ensemble des indices j où $b_j < 0$ et I celui où $b_i \geq 0$:

$$\begin{pmatrix} A_I & 0 \\ -A_J & -Id \end{pmatrix} [x \ , \ t] \quad \leq \quad \begin{pmatrix} b_I \\ b_J \end{pmatrix} \qquad x, t \geq 0$$

$$\max d[x,t] \quad = \quad \Sigma_{j\in J}(L_j x - t_j)$$

où L_j est la ligne de A_J correspondant à l'indice j.

On remarque les points suivants sur ce nouveau programme :

• Pour tout point du polyèdre des contraintes de $\mathcal{P}'$ on a

$$d.[x,t] \leq \Sigma_{j\in J} b_j$$

En effet la contrainte pour chaque $j \in J$ s'écrit :

$$L_j x - t_j \leq b_j$$

• Il existe un x satisfaisant $Ax \leq b$ si et seulement si le programme avec les t_j associé admet un optimum égal à $\sum_{j\in J} b_j$:

En effet si x est un point du polyèdre de $\mathcal{P}$ alors on a : $b_j \leq L_j x$, puis en posant $t_j = L_j x - b_j$ on a $t_j \geq 0$ et

$$\Sigma_{j\in J}(L_j x - t_j) = \Sigma_{j\in J} b_j$$

Réciproquement, si l'optimum de $\mathcal{P}'$ est $\Sigma_{j\in J} b_j$, comme tous les $L_j x - t_j$ sont positifs ou nuls on a $L_j x - t_j = b_j$ pour tout j, ainsi les contraintes de $\mathcal{P}$ sont satisfaites.

• Si l'optimum de $\mathcal{P}'$ atteint en $[x^*, t^*]$ donne pour valeur $\Sigma_{j\in J} b_j$, on vérifie que x^* est un sommet du polyèdre des contraintes de $\mathcal{P}$. En effet $[x^*, t^*]$ est un sommet de celui de $\mathcal{P}'$, il est donc à l'intersection de $n + |J|$ hyperplans des contraintes et donc x^* est l'intersection de n hyperplans de $\mathcal{P}$.

7. Le simplexe sous forme de tableaux

Nous avons présenté l'algorithme du simplexe dans le cadre de l'algèbre linéaire qui nous a permis de donner les preuves des résultats sans faire trop appel à des détails de calcul techniques.

On présente souvent le simplexe sous la forme de mise à jour de tableaux en traduisant l'algorithme plus haut en une suite d'instructions simples. Nous nous proposons re reprendre ici cette formulation qui permet de gagner dans la complexité de l'algorithme de changement de sommet du polyèdre ; celle-ci passe en effet de $O(n^3)$ à $O(n^2)$.

On considère un problème linéaire se présentant sous la forme

$$Ax \leq b, x \geq 0, \max cx$$

et on a vu que tout problème peut se ramener à cette forme.

On suppose ainsi que les contraintes (ou de manière équivalente les hyperplans) sont au nombre de $n+m$ et que les n premières sont $-x_i \leq 0$; les m suivantes correspondent aux lignes de A.

L'algorithme du simplexe se présente alors sous la forme de la mise à jour de 3 tableaux α, β, γ dont la valeur est initialisée à A, b, c respectivement. On utilise aussi un ensemble d'indices I initialisé à $\{1, 2, \ldots, n\}$, son complémentaire $J = \{n+1, n+2, \ldots, n+m\}$ est appelé la base pour une raison que nous verrons plus loin.

L'algorithme du simplexe s'exprime de la façon suivante dont on notera l'analogie avec la formulation précédente.

TANT QU'IL EXISTE $i \in I$ TEL QUE $\gamma_i > 0$ FAIRE

SOIT h LE PLUS PETIT INDICE DE I TEL QUE $\gamma_h > 0$;

POUR TOUT $j \in J$ FAIRE

CALCULER $\Delta_j = \frac{\beta_j}{\alpha j, h}$ POUR $\alpha j, h \neq 0$

SI TOUS LES Δ_j CALCULÉS SONT NÉGATIFS ALORS OPTIMUM À L'INFINI. SINON SOIT S TEL QUE Δ_s SOIT MINIMUM POSITIF.

POSER $I = I - \{h\} \cup \{s\}, \ J = J - \{s\} \cup \{h\}$;

FAIRE **Pivot**$(\alpha, \beta, \gamma, \mathbf{h}, \mathbf{s})$

L'opération de pivot pivot met à jour les tableaux α, β, γ qui deviennent α', β', γ' par les formules qui suivent.

Pour α :

$$\alpha'_{h,s} = \frac{1}{\alpha_{s,h}}, \quad \forall i \in I, i \neq h, \ \alpha'_{h,i} = \frac{\alpha_{s,i}}{\alpha_{s,h}}$$

$$\forall j \in J, \ j \neq s, \ \ \forall i \in I, \ i \neq h \quad \alpha'_{j,i} = \alpha_{j,i} - \alpha_{j,h}\frac{\alpha_{s,i}}{\alpha_{s,h}}, \quad \alpha'_{j,s} = \frac{\alpha_{j,h}}{\alpha_{s,h}}$$

Pour β :

$$\beta'_h = \frac{\alpha_{s,i}}{\alpha_{s,h}} \ \text{ et } \quad \forall j \in J, \ j \neq s, \ \ \beta'_j = \beta_j - \alpha_{j,h}\frac{\beta_s}{\alpha_{s,h}}$$

Pour γ :

$$\gamma'_s = -\frac{\gamma_h}{\alpha_{s,h}} \quad \text{et} \qquad \forall i \in I,\ i \neq h \ \ \gamma'_i = \gamma_i - \gamma_h \frac{\alpha_{s,i}}{\alpha_{s,h}}$$

En fait cette gestion de tableaux provient d'une autre écriture des contraintes obtenues en introduisant des nouvelles variables x_{n+i}, une pour chaque hyperplan des contraintes.

Ces contraintes sont alors écrites comme un système d'équations ; la contrainte $L_i x \leq b_i$ donnant l'équation : $x_{n+i} = b_i - L_i x$.

A chaque étape chaque variable x_j pour $j \in J$ satisfait l'équation :

$$x_j = \beta_j - \Sigma_{i \in I} \alpha_{j,i} x_i$$

De plus la fonction à optimiser s'écrit au cours de cette étape

$$cx + \Sigma_{i \in I} \gamma_i x_i$$

Au cours de chaque étape on détermine la variable x_h qui sort de I pour entrer dans la base J comme une de celles qui correspond à un c_i positif, et la variable x_s qui sort de J pour entrer dans I comme celle qui correspond à un Δ_s positif minimum.

Le pivot, qui correspond au passage d'un des sommets du polyèdre à l'un de ses sommets adjacents, est ainsi un changement de variable consistant à exprimer x_h en fonction de x_s et des autres variables x_i pour $i \in I$.

Les transformations reviennent à exprimer alors toutes les variables $x_j, j \in J$ en fonction des $x_i, i \in I$ en utilisant les équations de l'étape précédente.

Illustrons ceci sur notre exemple qui peut s'écrire après avoir remplacé x_2 (qui peut être négatif) par $x_2 - x_4$:

$$\left\{ \begin{array}{lcl} x_i & \geq 0 & i = 1,2,3,4,5,6,7,8 \\ x_5 & = & 4 - (2x_1 + x_2 - x_4 - x_3) \\ x_6 & = & 2 - (3x_1 - 2x_2 + 2x_4 x_3) \\ x_7 & = & 5 - (x_1 - 2x_2 + 2x_4) \\ x_8 & = & 8 - (-x_1 + x_3) \\ & \max & cx = (x_1 + x_2 - x_4 + x_3) \end{array} \right.$$

On chosit un x_i qui apparaît avec un coefficient positif dans cx par exemple ici x_1 (ainsi $h = 1$) et on calcule le Δ_s minimal parmi les positifs, c'est ici $\Delta_6 = \frac{2}{3}$, donc $s = 6$.

On exprime x_1 à l'aide des autres variables en utilisant l'équation donnant x_6 ; on obtient ainsi :

$$x_1 = \frac{2}{3} - \frac{1}{3}x_6 + \frac{2}{3}x_2 - \frac{2}{3}x_4 - \frac{1}{3}x_3$$

On reporte dans les équations donnant x_5, x_7, x_8 pour obtenir :

$$\begin{array}{rcl} x_5 & = & \frac{8}{3}+\frac{2}{3}x_6-\frac{7}{3}x_2+\frac{7}{3}x_4+\frac{5}{3}x_3 \\ x_7 & = & \frac{13}{3}+\frac{1}{3}x_6+\frac{4}{3}x_2-\frac{4}{3}x_4+\frac{1}{3}x_3 \\ x_8 & = & \frac{26}{3}-\frac{1}{3}x_6+\frac{2}{3}x_2-\frac{2}{3}x_4-\frac{4}{3}x_3 \\ c & = & \frac{2}{3}-\frac{1}{3}x_6+\frac{5}{3}x_2-\frac{5}{3}x_4+\frac{2}{3}x_3 \end{array}$$

On choisit $h = 2$ car x_2 a un coefficient positif dans cx ; le seul Δ_i non négatif est alors Δ_5, ainsi on tire la valeur de x_2 dans l'équation qui donne x_5 ; on obtient :

$$x_2 = \frac{8}{7}+\frac{2}{7}x_6-\frac{3}{7}x_5+x_4+\frac{5}{7}x_3$$

On reporte dans les autres :

$$\begin{array}{rcl} x_1 & = & \frac{10}{7}-\frac{1}{7}x_6-\frac{2}{7}x_5+\frac{1}{7}x_3 \\ x_7 & = & \frac{41}{7}+\frac{5}{7}x_6-\frac{4}{7}x_5+\frac{9}{7}x_3 \\ x_8 & = & \frac{66}{7}-\frac{1}{7}x_6-\frac{2}{7}x_5-\frac{6}{7}x_3 \\ c & = & \frac{18}{7}-\frac{1}{7}x_6+\frac{5}{7}x_5+\frac{13}{7}x_3 \end{array}$$

On prend enfin ici $h = 3$ et $s = 8$ et on effectue le dernier pivotement :
Ce qui donne le nouveau système;

$$\begin{array}{rcl} x_3 & = & 11-\frac{1}{6}x_6-\frac{1}{3}x_5-\frac{7}{6}x_8 \\ x_1 & = & 3-\frac{1}{6}x_6-\frac{1}{3}x_5-\frac{1}{6}x_8 \\ x_2 & = & 9+\frac{1}{6}x_6-\frac{2}{3}x_5-\frac{5}{6}x_8 \\ x_7 & = & 20+\frac{1}{6}x_6-x_5-\frac{3}{2}x_8 \\ c & = & 23-\frac{1}{6}x_6-\frac{4}{3}x_5-\frac{13}{6}x_8 \end{array}$$

Dans ce système, tous les coefficients de c sont négatifs, on a atteint l'optimum qui est bien, comme il a été calculé par la version matricielle du simplexe, au point $(3, 9, 11)$.

8. Problèmes équivalents

On montre que si l'on a un algorithme polynomial qui résout l'un des problèmes suivants, alors il en existe un pour tous les autres. De fait on donne des réductions entre problèmes, réductions qui peuvent être très utiles lorsque l'on veut résoudre l'un d'entre eux. Dans ce qui suit on suppose que tous les nombres qui interviennent dans les matrices, vecteurs ou autres sont des rationnels; maximum des logarithmes de tous les numérateurs et du dénominateurs détermine la taille du problème.

Dans l'ennoncé qui suit *résoudre* $\max\{cx \mid Ax \leq b\}$ signifie déterminer si ce maximum est fini, infini ou si le problème est non réalisable. Dans le premier cas donner un x qui fournit le maximum, dans le second donner un x tel que $Ax \leq b$ et un $y \geq 0$ tel que $yA = 0$ et $cy > 0$.

THÉORÈME 4.3. *Les problèmes suivants sont polynomialement équivalents*
- *Déterminer s'il existe un x tel que $Ax \leq b$*
- *Déterminer s'il existe un $x \geq 0$ tel que $Ax = b$*
- *Trouver un x, s'il existe, tel que $Ax \leq b$*
- *Trouver un $x \geq 0$, s'il existe, tel que $Ax = b$*
- *Etant donné un nombre γ, déterminer s'il existe un x tel que $Ax \leq b$ et $cx \geq \gamma$*
- *Résoudre* $\max\{cx \mid Ax \leq b\}$.
- *Résoudre* $\max\{cx \mid x \geq 0, \quad Ax \leq b\}$
- *Résoudre* $\max\{cx \mid x \geq 0, \quad Ax = b\}$
- *Résoudre* $\min\{cx \mid \quad Ax \geq b\}$
- *Résoudre* $\min\{cx \mid x \geq 0, \quad Ax \geq b\}$
- *Résoudre* $\min\{cx \mid x \geq 0, \quad Ax = b\}$

DÉMONSTRATION. Laissée en exercice en utilisant le théorèmes donnés plus haut.

9. L'algorithme Primal-Dual

On considère donc le problème primal sous forme normale.

$$(\mathcal{P}) \begin{cases} \max \ vx \\ Ax = b \\ x \geq 0 \end{cases} .$$

Considérons le système dual

$$(\mathcal{D}) \begin{cases} \min \ yb \\ yA \geq v \end{cases} ,$$

et supposons que nous possédons pour $\mathcal{D}$ une solution réalisable u. On peut alors transformer $\mathcal{P}$ en un système $\mathcal{P}'$ ayant les mêmes solutions optimales, car seule la fonction d'objectif a été translatée de la constante ub :

$$(\mathcal{P}') \begin{cases} \max \ (v - uA)x \\ Ax = b \\ x \geq 0 \end{cases} .$$

Ceci pourra advenir si, par exemple, une solution réalisable de $\mathcal{P}$ est telle que

$$x_i > 0 \Rightarrow v_i - yA_i = 0 \ \text{ et } \ v_i - yA_i < 0 \Rightarrow x_i = 0$$

remplissant ainsi les conditions du théorème de complémentation.

On essaye de se mettre dans cette situation et de trouver une solution réalisable de $\mathcal{P}$ telle que $x_i > 0 \Rightarrow v_i - yA_i = 0$.

Pour ce faire on considère le système, dit primal restreint, défini par :

$$(\mathcal{P}R(u))\begin{cases}\min\varphi=\sum y_i\\ Ax+y=b\\ x\geq 0\\ x_i=0 \text{ si } (v-uA)_i<0\\ y>0\end{cases}$$

et réalisable à partir de la solution de base $y = b$ pour laquelle on aura rendu tous les b_i positifs possiblement en multipliant l'équation correspondante par -1.

La solution optimale peut être de deux types.
1. Elle vérifie $\varphi = 0$, signifiant qu'il existe une solution de base x réalisable pour $\mathcal{P}$ et satisfaisant les conditions de complémentarité avec le u de départ.
2. À l'optimum de $\mathcal{P}R(u)$, la fonction φ est toujours strictement positive : il n'existe donc pas de solution réalisable de $\mathcal{P}$ qui soit complémentaire avec u. On modifie alors u en u' pour obtenir un $\mathcal{P}R(u')$ dont le domaine de solutions est un peu plus grand. Considérons le dual de $\mathcal{P}R(u)$:

$$(\mathcal{D}(u))\begin{cases}\max\vartheta b\\ \vartheta A\leq 0, \text{ pour les colonnes où } (v-uA)_i=0\\ \vartheta\leq 1\end{cases}$$

sans contrainte de signe sur ϑ.

On a donc maintenant $\min\varphi = \varphi^\star = \vartheta^\star b = \max\vartheta b > 0$. Ces variables duales optimales $\vartheta^\star$ vérifient, elles, les conditions de complémentarité : $x_i > 0 \Rightarrow \vartheta^\star A_i = 0$.
On considère alors le nouveau vecteur générique $u' = u - \lambda\vartheta$: pour toutes les composantes de la base, le vecteur $v - u'A$ reste nul comme précédemment puisque $\vartheta^\star A$ l'est. Pour $\lambda > 0$ on a $u'b = ub - \lambda\vartheta^\star b < ub$ et donc on peut diminuer l'écart entre la fonction primale et la fonction duale en choisissant la plus grande valeur de λ compatible avec les contraintes qui assurent que u' reste une fonction réalisable.

Si la valeur de λ n'est pas bornée, on en déduit que le primal $\mathcal{P}$ n'a pas de solution, sinon on remplace u par u' dans $\mathcal{P}'$ et on itère le processus. En raison de la décroissance de la fonction d'objectif duale et en l'absence de dégénérescence, le processus s'arrête en un nombre fini d'étapes.

10. Complexité du simplexe

Conceptuellement simple, la méthode évoquée depuis le début du chapitre s'est révélée délicate à implanter dans le détail. Il se trouve de plus que certains problèmes vont forcer le programme à passer par quasiment tous les sommets du simplexe qui, en dimension n, peuvent être au nombre de 2^n.

Un cas extrême V. Klee et G. J. Minty ont proposé le problème suivant en dimension n.

$$(\mathcal{K}M)\begin{cases} \max \sum_{j=1}^{n} 10^{n-j} x_j \\ (2\sum_{j=1}^{i-1} 10^{i-j} x_j) + x_i \leq 100^{i-1}, \text{ pour } (1 \leq i \leq n) \\ x_j \geq 0, \text{ pour } (1 \leq j \leq n) \end{cases}.$$

Prenons par exemple les cas $n = 3$ et résolvons le système en faisant systématiquement le pivot sur la variable dont le coefficient dans la fonction d'objectif est le plus élevé. Le problème s'énonce donc :

$$(\mathcal{K}M)\begin{cases} \max\ 100x_1 + 10x_2 + x_3 & \\ x_1 & \leq 1 \\ 20x_1 + x_2 & \leq 100 \\ 200x_1 + 20x_2 + x_3 & \leq 10000 \\ x_1, x_2, x_3 & \geq 0 \end{cases}.$$

On commence par mettre ce système en forme normale ajoutant les variables positives x_4, x_5, x_6. On obtient alors mis dans une forme un peu différente

$$(\mathcal{K}M)\begin{cases} & \max\ 100x_1 + 10x_2 + x_3 \\ x_4 = & 1 - x_1 \\ x_5 = & 100 - 20x_1 - x_2 \\ x_6 = & 10000 - 200x_1 - 20x_2 - x_3 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{cases}.$$

On effectue donc le pivot sur x_1, variable qui est dans la base, ce qui donne d'après la première équation

$$(\mathcal{K}M1)\begin{cases} & \max\ 100 - 100x_4 + 10x_2 + x_3 \\ x_1 = & 1 - x_4 \\ x_5 = & 80 + 20x_4 - x_2 \\ x_6 = & 9800 + 200x_4 - 20x_2 - x_3 \end{cases}.$$

Le pivot maintenant se fait sur x_2 et la deuxième équation :

$$(\mathcal{K}M2)\begin{cases} & \max\ 900 + 100x_4 - 10x_5 + x_3 \\ x_1 = & 1 - x_4 \\ x_2 = & 80 + 20x_4 - x_5 \\ x_6 = & 8200 - 200x_4 + 20x_5 - x_3 \end{cases}.$$

et il faudra encore 5 pivots, que le lecteur est convié à effectuer, avant d'obtenir

$$(\mathcal{K}M7)\begin{cases} & \max\ 10000 - 100x_1 - 10x_2 - x_6 \\ x_4 = & 1 - x_1 \\ x_5 = & 100 - 20x_1 - x_2 \\ x_3 = & 10000 - 200x_1 - 20x_2 - x_6 \end{cases},$$

sur lequel il n'est plus possible d'améliorer la fonction d'objectif et qui donne l'optimum.

Il est un peu plus dur de montrer en toute généralité que le coût est exponentiel. Cependant c'est bien ce qui se passe.

Plus généralement, il a été montré pour toutes les stratégies naturelles de sélection du pivot qu'il existait des systèmes pour lesquels la stratégie obligeait à parcourir tous les sommets du simplexe, ruinant ainsi les espoirs de comportement polynomial.

11. Des algorithmes polynomiaux

Depuis de nombreux travaux ont été publiés sur le sujet. Popularisée par la contribution de S. Smale, l'approche stochastique à la complexité moyenne du problème mit en évidence que ce cas extrême est bien peu fréquent et que le nombre moyen de pivots reste polynomial en la dimension du problème.
De plus, Katchian puis Karmarkar ont donné des algorithmes polynomiaux. On a même montré que la programmation linéaire était complète dans la classe des problèmes polynomiaux pour la réduction en espace logarithmique, plaçant ainsi très précisément le problème dans les hiérarchies de complexité.

11.0.1. *La méthode de Katchian.* L'algorithme a été conçu pour résoudre des systèmes d'inéquations. Il peut également s'appliquer à la programmation linéaire soit en récrivant le systèmes d'équations comme système d'inéquations, soit en utilisant une méthode dite de *fonction objectif glissante*, qui est une simple variante de la méthode. En voici une description simplifiée.
On veut donc résoudre le système :

$$Ax \leq b, \qquad (1)$$

où $x \in \mathtt{R}^n$, $b \in (\mathtt{R}^+)^m$ et $A \in \mathtt{R}^{m \times n}$.
On notera a_i le vecteur correspondant à la i-ème ligne de A, et qui est donc un vecteur ligne.

Le principe. On travaille ici avec les réels des ordinateurs, et donc on peut normaliser les entrées du système afin d'en faire des entiers, par simple multiplication par une puissance convenable de 10. On ne se pose pas ici la question de la qualité de l'approximation que l'on va obtenir, et des risques d'erreur consécutifs à un mauvais conditionnement de la matrice. Soit L le nombre total de bits nécessaire à l'écriture du système.
On montre que le système (2) suivant :

$$Ax \leq b + 2^{-L}, \qquad (2)$$

où on ajoute donc la quantité 2^{-L} à chaque membre droit d'inéquation, a une solution si et seulement si (1) en a une. Intuitivement, la perturbation est tellement faible qu'elle n'affecte pas la nature du résultat. De plus, on peut obtenir la solution de (1) en temps polynomial à partir de celle de (2). Enfin, le système (2) est plus facile à traiter car s'il a une solution, il existe une boule de rayon 2^{-2L} dont tous les points sont solution.

La méthode de recherche des solutions s'exprime géométriquement par la construction d'une suite d'ellipsoïdes emboîtés, dans lesquels, à chaque étape, on sélectionne un point dont on espère qu'il satisfait le système. Si tel n'est pas le cas, la k-ième contrainte étant violée par exemple, on restreint l'ellipsoïde parallèlement à cette facette : le lecteur remarquera une certaine analogie avec la méthode du simplexe.

On construit donc une suite de points $z_0, z_1, \ldots, z_p \in \mathtt{R}^n$ et de matrices définies positives $E_0, E_1, \ldots, E_p \in \mathtt{R}^{n\times n}$, par récurrence, comme suit.
Le premier ellipsoïde est défini par :

$$z_0 = 0, E_0 = 2^{-L} I.$$

À la k-ième étape on procède aux opérations suivantes :
– tester si z_k satisfait le système (2) et alors s'arrêter ;
– sinon, soit i l'indice d'une inégalité violée par z_k, $a_i^t z_k > b[i]$; calculer

$$z_{k+1} = z_k - \frac{1}{n+1} \frac{E_k a_i^T}{\sqrt{a_i E_k a_i^T}},$$

$$E_{k+1} = \frac{n^2}{n^2-1} \left(E_k - \frac{2}{n+1} \frac{(E_k a_i^T)^T (E_k a_i^T)}{a_i E_k a_i^T} \right).$$

On définit ainsi un ellipsoïde de centre z_{k+1}, dont les points satisfont :

$$\mathcal{E}_{k+1} = \{x \mid (x - z_{k+1})\ E_{k+1}^{-1}\ (x - z_{k+1})^T \leq 1\};$$

cet ellipsoïde contient les points de $\mathcal{E}_k$ qui sont encore susceptibles d'être solution. En particulier, si l'ensemble des solutions est non vide, tous les $\mathcal{E}_k$ contiennent la boule de solutions de rayon 2^{-2L} initialement située dans $\mathcal{E}_0$.
On observe que le volume des ellipsoïdes décroît à chaque étape d'un facteur majoré par $e^{-1/(2n+2)}$. En conséquence, si rien n'arrête l'itération ce volume deviendra plus petit que celui de la boule cherchée de rayon 2^{-2L}, à savoir après Cn^2L itérations, pour une constante C.
Si l'algorithme n'a pas permis à ce stade de trouver un centre d'ellipsoïde solution du système, c'est que le système n'en a pas.

Détails de réalisation. Dans la pratique, il faut prendre en compte les erreurs d'arrondi et augmenter significativement la précision des calculs. On ne peut cependant pas aller jusqu'à calculer en nombres rationnels, car alors la taille des fractions risquant de croître exponentiellement, on remettrait en cause le coût polynomial en nombre d'itérations. De fait, il est possible de trouver un compromis et de conserver le caractère polynomial de l'algorithme. Remarquons que cette méthode est presque tout le temps plus coûteuse que celle du simplexe.

11.0.2. *La méthode des fonctions d'objectif glissantes.* On considère de nouveau un problème d'optimisation :

$$(\mathcal{P}^+) \left\{ \begin{array}{l} \max\ vx \\ Ax \leq b \end{array} \right. .$$

On commence par déterminer une solution du système d'inégalités associé de type (2), comme décrit plus haut. On détermine ainsi un point $\overline{z}$.
On ajoute alors l'inégalité $vx \geq v\overline{z}$ aux contraintes et on continue d'itérer le processus, y compris pour améliorer le $\overline{z}$ de départ.
Chaque fois qu'une nouvelle solution x_k qui satisfait $vx_k \geq v\overline{z}$ est trouvée, on pose $\overline{z} := x_k$ et on continue comme plus haut.

On résout ainsi le problème (3) en temps polynomial, mais ce polynôme dépend de la précision initiale L. De plus, comme déjà mentionné, cette dépendance fait qu' en pratique la méthode de Katchian n'est pas vraiment compétitive.

La méthode de Karmarkar. Après une amélioration de Megiddo, le pas décisif fut effectué par Karmarkar qui améliore notablement la méthode de Katchian en proposant une vision géométrique de type gradient, dont on donne une idée.
La procédure de base est un algorithme rapide pour résoudre les problèmes du type :

$$(\mathcal{K}^+)\left\{\begin{array}{l} \max\ vx,\ \text{pour } x \in U \cap S, \\ U = \{x \mid Ax = 0\} \\ S = \{x \mid x \geq 0, ||x|| = 1\}, \end{array}\right. .$$

où $v, x \in \mathtt{R}^n$, $b \in \mathtt{R}^m$ et $A \in \mathtt{R}^{m\times n}$ et $||x|| = \sum x[i]$ est la somme des coordonnées de x.
On suppose de plus que :
– U est défini par des équations homogènes ;
– la valeur minimale de x est 0 ;
– le centre de la sphère S, le point de coordonnées $1/n$ dans chaque dimension, est solution du système ; – on connaît un paramètre entier q qui permet d'arrêter les itérations dès que $vz_k \leq 2^{-q} vz_0$.
La suite des solutions z_k est telle que la fonction dite de potentiel $f(x) = \sum \log \frac{vx}{x[i]}$, diminue à chaque itération d'une constante (d'où la convergence rapide).
Géométriquement, la transformation projective appliquée à chaque étape de S sur S applique z_k sur le point $\frac{1}{n}e$ qui dans ce nouveau système de coordonnées permet de minimiser la fonction d'objectif.
On obtient ainsi une solution optimale pour le système $\mathcal{K}$ en un temps $O(n^{3.5}L^2)$.

Bibliographie

[1] Ausiello Giorgio, Crescenzi Pierluigi, Gambosi Giorgio, Kann Viggo, Marchetti-Spaccamela Alberto, Protasi Marco, Complexity and Approximation. Combinatorial optimization problems and their approximability properties. ISBN 3-540-65431-3. Springer-Verlag, 1999.

[2] Berge Claude, *Graphes et Hypergraphes*, Bordas, 1973.

[3] Chvátal Vasek, *Linear Programming* McGraw-Hill, 1983.

[4] Cormen Thomas, Leiserson Charles, Rivest Ronald, *Introduction to Algorithms*, MIT Press, 1990.

[5] Crochemore Maxime, Rytter W. *Text Algorithms.* Oxford University Press, 1994.

[6] Even Shimon, *Graph Algorithms*, Computer Science Press, 1979.

[7] Garey Michael R., Johnson David S., Computers and Intractability : A Guide to the Theory of NP-completeness. ISBN 0-7167-1045-5. Freeman, 1979.

[8] Gibbons Alan, *Algorithmic Graph Theory* Cambridge University Press, 1985.

[9] Goldberg David, *Genetic Algorithms*, Addison Wesley 1989.

[10] Gondran Michel, Minoux Michel, *Graphes et Algorithmes*, Eyrolles, 1995.

[11] Knuth Donald, *The Art of Computer Programming, Vol 3 : Sorting and Searching*, Addison Wesley, 1973.

[12] Lawler, E. L, Lenstra J.K, Rinnooy K, Shmoys D. B, *The Travelling Salesman Problem : a Guided Tour of Combinatorial Optimization*, John Wiley and sons, 1985 .

[13] Lint J. van, Wilson Robert, *A course in Combinatorics*, Cambridge University Press, 1992.

[14] Motwani R., Raghavan P. Randomized Algorithms. ISBN 0-521-47465-5. Cambridge University Press, 1995.

[15] Nemhauser Georges, Wolsley Laurence, *Integer and Combinatorial Optimization*, John Wiley and sons 1988

[16] Papadimitriou C. H. Computational complexity. ISBN 0-201-53082-1. Addison-Wesley, 1994.

[17] Papadimitriou C. H., Steiglitz K. Combinatorial Optimization. Prentice-Hall, 1982.

[18] Roth Alvin, Sotomayor Marilda, *Two sided matching*, Cambridge University Press, 1990.

[19] Tarjan Robert Endre, *Data Structures and Network Algorithms*, SIAM, Philadelphia, 1983.

[20] Vazirani Vijay V., *Approximation Algorithms*, Springer 2001.

[21] Welsh Dominic, *Matroid Theory*, Academic Press, 1976.