Cours de Recherche Opérationnelle
IUT d'Orsay

Nicolas M. THIÉRY

E-mail address: Nicolas.Thiery@u-psud.fr
URL: http://Nicolas.Thiery.name/

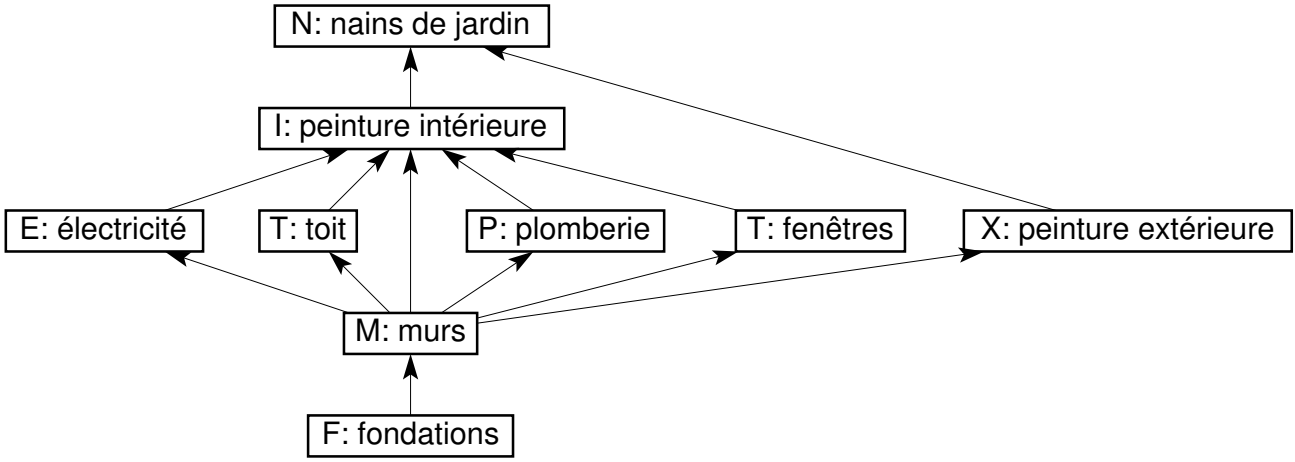
CHAPTER 1

Introduction à l'optimisation

1.1. TD: Ordonnancement par la méthode des potentiels

EXERCICE. On veut construire une maison, ce qui consiste en 9 tâches, le plus rapidement possible, avec les contraintes suivantes:

- Certaines tâches dépendent d'autres tâches
- Toutes les tâches demandent une semaine de travail.
- Chaque ouvrier ne peut travailler que sur une tâche par jour
- Il n'y a pas de gain de temps si plusieurs ouvriers travaillent sur la même tâche



EXEMPLE. Organisez un emploi du temps pour un ouvrier, de façon à construire la maison le plus rapidement.

	semaine 1	semaine 2	semaine 3	semaine 4	semaine 5	semaine 6	semaine 7
ouvrier 1							

De même, s'il y a deux ouvriers:

	semaine 1	semaine 2	semaine 3	semaine 4	semaine 5	semaine 6	semaine 7
ouvrier 1							
ouvrier 2							

De même, s'il y a trois ouvriers:

	semaine 1	semaine 2	semaine 3	semaine 4	semaine 5	semaine 6	semaine7
ouvrier 1							
ouvrier 2							
ouvrier 3							

Et s'il y a plus d'ouvriers ?

Généralisation avec des durées.

1.2. Problèmes d'optimisation

PROBLEM 1.2.1. Quelles sont les coordonnées du point culminant du massif de la Chartreuse?

Modélisation:

DÉFINITION 1.2.2. Un *problème d'optimisation* est décrit par:

- Un *domaine* S
  Ici, S est l'ensemble des points p de la surface de la terre, décrits par leur coordonnées (longitude, latitude).
  Un élément de S est appellé *solution*.
- Des contraintes C définissant un sous-ensemble de S.
  Ici, $C(p) :=$ ■ p `est dans le massif de la Chartreuse` ■.
  Une solution p vérifiant les contraintes $C(p)$ est dite *solution faisable*.
- Une *fonction objectif* : $z : S \mapsto \mathbb{R}$
  Ici la fonction $p \mapsto z(p)$ qui associe à un point p de la surface de la terre sont altitude $z(p)$.

Il faut de plus préciser s'il s'agit d'un problème de *maximisation* ou de *minimisation*.

Une solution faisable p est dite *optimale* si $z(p) \geq z(q)$ pour toute autre solution faisable q.

But: déterminer l'ensemble des solutions optimales.

EXEMPLE 1.2.3. Modéliser les problèmes suivants, puis les résoudre ou proposer des méthodes de résolution:

(1) Calcul du plus court chemin dans un graphe.
(2) Recherche d'un ordonnancement optimal pour ...
(3) Déterminer le plus petits nombre de cartons nécessaires pour un déménagement.
(4) Chercher un mat en un minimum de coup
(5) Recherche de l'élément maximal d'un tableau de n valeurs aléatoires.
(6) Maximiser z pour z dans $[0, 1[$.
(7) Maximiser z pour z dans $\mathbb{R}^+$.
(8) Déterminer la valeur minimale de la fonction $x^2 + 2x - 4$.
`plotfunc2d(x^2 + 2*x - 4)`
(9) Déterminer la valeur minimale de la fonction $2x^3 - 5x^2 - x + 7$.
`plotfunc2d(2*x^3 - 5*x - x + 7)`
(10) Déterminer la valeur maximale de la fonction $\sin(x)\sin(\pi x + 2) + 0.01x^2$.
`plotfunc2d(sin(x) * sin(PI*x+2)-0.01*x^2,x=-20..20)`
(11) Quelle est la profondeur maximale de l'océan?

Bilan: un problème d'optimisation peut avoir

- Aucune solution
- Aucune solution optimale (non borné)
- Une unique solution optimale

Quelques méthodes d'optimisation:

(1) Méthodes globales:
    - Monte-Carlo
    - Recherche exhaustive (toujours possible pour un problème discret fini; parfois la seule méthode existante)
    - Recherche exhaustive structurée, avec coupures de branches.
(2) Méthodes locales, itératives:
    Lorsque le problème a une propriété de ■convexité■, l'optimisation locale amène à une optimisation globale!
    - Algorithmes gloutons
    - Méthodes analytiques (gradient)
    - Méthode du simplexe
(3) Méthodes mixtes:
    - Monte-Carlo + gradient
    - Recuit simulé, ...

CHAPTER 2

Programmation linéaire

2.1. Rappel d'algèbre linéaire

PROBLEM 2.1.1. Considérons le système suivant:

```
5  = s1       + 2*x1 + 3*x2 +   x3
11 =    s2    + 4*x1 +   x2 + 2*x3
8  =       s3 + 3*x1 + 4*x2 + 2*x3
```

Que peut-on dire dessus?

C'est un système *linéaire* à 6 inconnues et 3 équations.

On peut décrire l'ensemble des solutions en prenant comme paramètres `x1`, `x2` et `x3`.

En effet, vu sa forme triangulaire, s1, s2 et s3 s'expriment en fonction de `x1`, `x2` et `x3`.

Transformer le système pour prendre comme paramètres `s1`, `s2`, et `x1`.

2.2. Qu'est-ce que la programmation linéaire

2.2.1. Exemple: le problème du régime de Polly [1, p.3].

- Besoins journaliers:
  Énergie: 2000 kcal
  Protéines: 55g
  Calcium: 800 mg
- Nourriture disponible

	Portion	Énergie (kcal)	Protéines (g)	Calcium (mg)	Prix/portion
Céréales	28g	110	4	2	3
Poulet	100g	205	32	12	24
Oeufs	2 gros	160	13	54	13
Lait entier	237cc	160	8	285	9
Tarte	170g	420	4	22	20
Porc et haricots	260g	260	14	80	19

- Contraintes:
  Céréales: au plus 4 portions par jour
  Poulet: au plus 3 portions par jour
  Oeufs: au plus 2 portions par jour
  Lait: au plus 8 portions par jour
  Tarte: au plus 2 portions par jour
  Porc et haricots: au plus 2 portions par jour

PROBLEM 2.2.1. Polly peut-elle trouver une solution ?

Comment formaliser le problème ? (modélisation)

Qu'est-ce qui fait la spécificité du problème ?

Savez-vous résoudre des problèmes similaires ?

2.2.2. Forme standard d'un problème de programmation linéaire.

PROBLEM 2.2.2. [**1**, p. 5]

```
Maximiser:            5*x1 + 4*x2 + 3*x3
Sous les contraintes: 2*x1 + 3*x2 +   x3 <=  5
                      4*x1 +   x2 + 2*x3 <= 11
                      3*x1 + 4*x2 + 2*x3 <=  8
                      x1, x2, x3 >= 0
Minimiser:            3*x1 -   x2
Sous les contraintes: - x1 + 6*x2 -   x3 +   x4 >= -
3
                            7*x2        + 2*x4 =   5
                       x1 +   x2 +   x3        =   1
                                     x3 +   x4 <=  2
                      x2, x3 >= 0
```

DÉFINITION. Problème de programmation linéaire sous *forme standard:*
Maximiser:

$$z := \sum_{j=1}^{n} c_j x_j$$

Sous les contraintes:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \text{ pour } i = 1, \ldots, m$$

$$x_j \geq 0, \text{ pour } j = 1, \ldots, n$$

Un choix des valeurs des variables $(x_1, \ldots, x_n)$ est appelé *solution* du problème.
Une solution est *faisable* si elle vérifie les contraintes.
z est appelé *fonction objective*. À chaque solution elle associe une valeur.
Une solution est *optimale* si elle est faisable et maximize la fonction objective.

EXERCICE 1. Peut-on mettre sous forme standard les exemples précédents ?

2.2.3. Existence de solutions optimales ?

PROBLEM 2.2.3. [**1**, p. 7] On considère les trois problèmes de programmation linéaire standard suivants, écrits avec la syntaxe du système de calcul formel `MuPAD`:

```
Chvatal7a := [ [   x1      <= 3,
                         x2 <= 7 ],
                 3 +x1  +x2,
                NonNegative]:
Chvatal7b := [ [   x1  +x2 <= 2,
                  -2*x1-2*x2 <= -10 ],
                   3*x1  -x2,
                NonNegative]:
Chvatal7c := [ [-2*x1  +x2 <= -1,
                    -x1-2*x2 <= -2 ],
                     x1  -x2,
                NonNegative]:
extra     := [ [   x1  +x2 <= 1 ],
                     x1  +x2,
                NonNegative]:
```

PROBLEM 2.2.4. Déterminer pour ces trois problèmes s'il y a des solutions optimales.

- Premier cas: une solution optimale unique
- Deuxième cas: pas de solution faisable
- Troisième cas: pas de solution optimale, car on peut faire tendre la fonction objective vers l'infini avec des solutions faisables.
- Quatrième cas: une infinité de solutions optimales.

2.3. Algorithme du simplexe

2.3.1. Premier problème.

PROBLEM 2.3.1. [**1**, p. 13]

```
Chvatal13 := [{2*x1 + 3*x2 +   x3 <= 5,
               4*x1 +   x2 + 2*x3 <= 11,
               3*x1 + 4*x2 + 2*x3 <= 8  },
               5*x1 + 4*x2 + 3*x3,
              NonNegative]:
```

Solution faisable ?

Amélioration de la solution ?

Introduction de variables d'écart:

```
5  = s1       + 2*x1 + 3*x2 +   x3
11 =    s2    + 4*x1 +   x2 + 2*x3
8  =       s3 + 3*x1 + 4*x2 + 2*x3
---------------------------------
z  =            5*x1 + 4*x2 + 3*x3
```

En augmentant `x1` jusqu'à 5/2, on fait tomber `s1` à zéro.

On transforme le système, pour se ramener à une situation similaire à la précédente:

```
5/2 = x1       + 3/2*x2 + 1/2*x3 + 1/2*s1
1   =    s2    -   5*x2           - 2*s1
1/2 =       s3 - 1/2*x2 + 1/2*x3 - 3/2*s1
---------------------------------------
z   = 25/2     - 7/2 x2 + 1/2*x3 - 5/2*s1
```

On augmente x3 jusqu'à 1, ce qui fait tomber s3 à 0:

```
1 = x3       -   x2 - 3*s1 + 2*s3
2 =    x1    + 2*x2 + 2*s1 -   s3
1 =       s2 - 5*x2 - 2*s1
-------------------------------
z = 13       - 3*x2 -   s1 -   s3
```

Et maintenant, que fait-on ?

2.3.2. Tableaux.

PROBLEM 2.3.2. [**1**, p. 19]

```
Chvatal19 := [[  x1 + 3*x2   + x3 <= 3,
                -x1         +3*x3 <= 2,
               2*x1 + 3*x2   - x3 <= 2,
               2*x1   - x2 + 2*x3 <= 4],
               5*x1 + 5*x2 + 3*x3,
              NonNegative]:
```

DÉFINITION. *Tableau initial:*

$$b_i = s_i + \sum_{j=1}^{n} a_{ij} x_j, \text{ pour } i = 1, \ldots, m$$

$$z = \sum_{j=1}^{n} c_j x_j$$

Ou sous forme matricielle:

$$\begin{aligned} B &= S + AX \\ z &= CX \\ X &\geq 0 \end{aligned}$$

EXEMPLE. Tableau initial du problème précédent:

```
3 = s1         + x1 + 3 x2   + x3
2 =   s2       - x1        + 3 x3
2 =     s3   + 2 x1 + 3 x2   - x3
4 =       s4 + 2 x1   - x2 + 2 x3
---------------------------------
z = 0        + 5 x1 + 5 x2 + 3 x3
```

EXEMPLE 2.3.3. On peut l'abréger sous forme matricielle:

```
read("tableaux.mu"):
linopt::Transparent(Chvatal19);
+-
-+
|"linopt","restr",slk[1],slk[2],slk[3],slk[4],x3,x1,x2|
|                                                      |
|  "obj",    0,      0,     0,     0,     0,   3, 5, 5|
|                                                      |
| slk[1],    3,      1,     0,     0,     0,   1, 1, 3|
|                                                      |
| slk[2],    2,      0,     1,     0,     0,   3,-
1, 0|
|                                                      |
| slk[3],    2,      0,     0,     1,     0,  -
1, 2, 3|
|                                                      |
| slk[4],    4,      0,     0,     0,     1,   2, 2,-
1|
+-
-+
```

DÉFINITION. De manière générale, un *tableau* est un ensemble d'équations de la forme:

```
4 = x1       + 3/2 x2  - 1/2 x3 + 1/2 s4
2 =   s1     + 3/2 x2  + 3/2 x3 - 1/2 s4
3 =     s2   + 3/2 x2  + 5/2 x3 + 1/2 s4
2 =       s3   - 4 x2    + 3 x3     - s4
----------------------------------------
z = 5        - 5/2 x2 + 11/2 x3 - 5/2 s4
```

x_1, s_1, s_2, s_3 sont les variables *basiques;* $\{x_1, s_1, s_2, s_3\}$ est la *base.*

x_2, x_3, s_4 sont les variables *non basiques.*

DÉFINITION 2.3.4. Le point de coordonnées $(0, \ldots, 0)$ dans les variables non-basiques est appellé *solution basique* du tableau.

Un tableau est *faisable* si la solution basique $(0, \ldots, 0)$ est une solution faisable.

De manière équivalente, un tableau est faisable si les constantes dans les équations du haut sont toutes positives ou nulles.

Revenons à l'exemple [**1**, p. 19]:

```
read("tableaux.mu"):
```

```
t:=linopt::Transparent(Chvatal19);
t:=linopt::Transparent::userstep(t, slk[3], x3);
```

EXERCICE 2. [**1**, 2.1 p. 26]

Utilisez l'algorithme du simplexe pour résoudre les programmes linéaires suivants:

```
Chvatal26_21a :=
[[  x1  +x2+2*x3 <= 4,
  2*x1     +3*x3 <= 5,
  2*x1  +x2+3*x3 <= 7],
  3*x1+2*x2+4*x3,
 NonNegative]:
Chvatal26_21c :=
[[2*x1+3*x2 <= 3,
    x1+5*x2 <= 1,
  2*x1  +x2 <= 4,
  4*x1  +x2 <= 5],
  2*x1  +x2,
 NonNegative]:
```

EXERCICE 3. Essayez d'appliquer l'algorithme du simplexe aux programmes linéaires de l'exercice [**1**, p. 7] (cf. ci-dessus). Que se passe-t'il ?

2.4. Pièges et comment les éviter

2.4.1. Bilan des épisodes précédents. On a un algorithme qui marche sur quelques exemples.

Il faut vérifier trois points pour savoir s'il marche en général:

(1) Initialisation
(2) Itération
(3) Terminaison

2.4.2. Itération.

PROPOSITION. *Étant donné un tableau faisable, on peut toujours effectuer l'une des opérations suivantes:*

(1) Conclure que le système a une solution optimale unique, la calculer et la certifier;
(2) Conclure que le système a une infinité de solutions optimales, les calculer et les certifier;
(3) Conclure que le système est non borné, et le certifier en décrivant une demi-droite de solutions sur laquelle z prend des valeurs aussi grandes que voulu.
(4) Trouver une variable entrante, une variable sortante, et effectuer un pivot. Par construction, le tableau obtenu est équivalent au tableau précédent, et est encore faisable. De plus, z a *augmenté au sens large* (i.e. la constante z^* dans la nouvelle expression de z est supérieure ou égale à l'ancienne).

PROOF. Il suffit d'analyser le tableau faisable. Notons $S_1, \ldots, S_m$ les variables basiques, $X_1, \ldots, X_n$ les variables non-basiques, et $C_1, \ldots, C_n, z^*$ les coefficients tels que $z = z^* + \sum C_i X_i$.

Par exemple, dans le tableau final du problème2.3.1, on a $X_1 = x_2$, $X_2 = s_1$, $X_3 = s_2$, $S_1 = x_1$, $S_2 = x_3$, $S_3 = s_3$, $C_1 = -3$, $C_2 = -1$, $C_3 = -1$ et $z^* = 13$.

(1) Si $C_i < 0$, pour tout i, alors la solution basique du tableau, de coordonnées $X_1^* = \cdots = X_n^* = 0$ est l'unique solution optimale. Vérifiez le en prouvant qu'une toute solution faisable quelconque de coordonnées $X_1, \ldots, X_n$ donnant la même valeur $z = z^*$ à la fonction objective est égale à la solution basique du tableau.
(2) Si $C_i \leq 0$ pour tout i, la solution basique du tableau est optimale, et l'ensemble des solutions optimales est décrit par les inéquations linéaires du système et l'annulation des variables non-basiques X_i pour lesquelles on a $C_i < 0$. Les détails sont similaires au 1.
(3) Sinon, on peut prendre X_i, variable non-basique avec un coefficient $C_i > 0$. Si les équations du tableau n'imposent pas de limite sur X_i, le système est non borné: la demi-droite décrite par $(0, \ldots, 0, X_i, 0, \ldots, 0)$ pour $X_i \geq 0$ est composée de solutions faisables qui donnent des valeurs aussi grandes que voulu à z.
(4) Autrement, une des variables basiques S_j tombe à zéro, et on peut faire un pivot entre la variable entrante X_i et la variable sortante S_j. Par construction, la nouvelle solution basique correspond à une solution faisable $(0, \ldots, 0, X_i, 0, \ldots, 0)$ pour un $X_i \geq 0$. En particulier le nouveau tableau est faisable, et comme $C_i \geq 0$, la constante z^* a augmenté au sens large.

□

EXEMPLE. [**1**, p. 29] Système où z n'augmente pas strictement lors du pivot:

```
Chvatal29 :=  [[              2*x3 <= 1,
                - x1 + 3*x2 + 4*x3 <= 2,
                2*x1 - 4*x2 + 6*x3 <= 3],
                2*x1   - x2 + 8*x3,
                NonNegative]:
t0:= linopt::Transparent(Chvatal29);
t1:= linopt::Transparent::userstep(t0, slk[1], x3);
t2:= linopt::Transparent::userstep(t1, slk[3], x1);
t3:= linopt::Transparent::userstep(t2, slk[2], x2);
t4:= linopt::Transparent::userstep(t3, x3, slk[1]);
```

REMARQUE. Lorsque z n'augmente pas, on est forcément dans une situation de dégénérescence: le pivot change le tableau, mais pas la solution basique décrite par le tableau.

2.4.3. Terminaison.

PROBLEM 2.4.1. Peut-on garantir que l'algorithme va finir par s'arrêter ?

THÉORÈME. *Si l'algorithme du simplexe ne cycle pas, il termine en au plus* $C(n+m, m)$ *itérations.*

PROOF. (Résumé)

Chaque itération correspond à un tableau faisable.

Un tableau faisable est entièrement caractérisé par le choix des variables basiques.

Il n'y a "que" $C(n+m, m)$ choix possibles de variables basiques. □

REMARQUE. L'algorithme ne peut cycler qu'en présence de dégénérescence.

Avec une stratégie incorrecte, l'algorithme du simplexe peut cycler éternellement:

EXEMPLE. [**1**, p. 31] Système cyclant en 6 itérations avec la stratégie:

- Choix de la variable entrante avec le coefficient dans l'expression de z le plus fort
- Choix de la variable sortante avec le plus petit index

```
Chvatal31 := [[0.5*x1 - 5.5*x2 -
2.5*x3 +  9*x4 <= 0,
               0.5*x1 - 1.5*x2 - 0.5*x3 +    x4 <= 0,
                   x1                           <= 1],
                10*x1  - 57*x2   - 9*x3 - 24*x4,
              NonNegative]:
t0 := linopt::Transparent(Chvatal31);
t1 := linopt::Transparent::userstep(t0, slk[1], x1);
t2 := linopt::Transparent::userstep(t1, slk[2], x2);
t3 := linopt::Transparent::userstep(t2, x1, x3);
t4 := linopt::Transparent::userstep(t3, x2, x4);
t5 := linopt::Transparent::userstep(t4, x3, slk[1]);
t6 := linopt::Transparent::userstep(t5, x4, slk[2]);
```

Comment garantir que l'algorithme ne cyclera pas ?

- Méthode des perturbations
- La méthode du plus petit index

  THÉORÈME. *L'algorithme du simplexe termine si, lorsqu'il y a ambiguïté sur le choix de la variable entrante ou sortante, on choisit toujours la variable de plus petit index.*

  Cette méthode est simple et élégante.

  Par contre, elle empêche toute stratégie pour faire converger l'algorithme plus vite.
- Méthodes intermédiaires

2.4.4. Initialisation. Pour le moment, l'algorithme du simplexe nécessite de partir d'un tableau faisable.

PROBLÈME. Dans le cas général, comment se ramener à un tableau faisable?

Le système pourrait même ne pas avoir de solution!

EXEMPLE. [**1**, p. 39] Système P_1:
Maximiser: $x_1 - x_2 + x_3$
Sous les contraintes:
$2x_1 - x_2 + 2x_3 \leq 4$
$2x_1 - 3x_2 + x_3 \leq -5$
$-x_1 + x_2 - 2x_3 \leq -1$
$x_1, x_2, x_3 \geq 0$

EXEMPLE. Introduction d'un *système* auxiliaire P_0 pour déterminer si P est faisable:
Maximiser: $-x_0$
Sous les contraintes:
$2x_1 - x_2 + 2x_3 - x_0 \leq 4$
$2x_1 - 3x_2 + x_3 - x_0 \leq -5$
$-x_1 + x_2 - 2x_3 - x_0 \leq -1$
$x_0, x_1, x_2, x_3 \geq 0$
Remarques:

- *P_0 est faisable (prendre x_0 suffisamment grand);*
- *Les solutions faisables de P correspondent aux solutions faisables de P_0 avec $x_0 = 0$;*
- *P est faisable si et seulement si P_0 a une solution faisable avec $x_0 = 0$.*

Étudions ce nouveau système:

```
Chvatal40 := [[ -x1   + x2 - 2*x3 - x0 <= -1,
                2*x1 - 3*x2   + x3 - x0 <= -5,
                2*x1   - x2 + 2*x3 - x0 <= 4],
               -x0,
               NonNegative]:
t0:=linopt::Transparent(Chvatal40);
t1:=linopt::Transparent::userstep(t0, slk[2], x0);
t2:=linopt::Transparent::userstep(t1, slk[1], x2);
t3:=linopt::Transparent::userstep(t2, x0, x3);
```

Maintenant, nous savons que le système P est faisable.
En fait, en éliminant x_0 on obtient même un tableau faisable pour P!

Algorithme du simplexe en deux phases pour résoudre un problème P sous forme standard:
Phase I:

(1) Si $(0, \ldots, 0)$ est solution faisable de P, on passe directement à la phase II.
(2) Définir un problème auxiliaire P_0.
(3) Le premier tableau pour P_0 est infaisable.

(4) Le rendre faisable par un pivot approprié de x_0.
(5) Appliquer le simplexe habituel:
  (a) Si à une étape donnée, x_0 peut sortir de la base, le faire en priorité: En effet, il y a une solution faisable avec $x_0 = 0$, et on peut passer en phase II.
  (b) Si à une étape donnée on atteint une solution optimale:
    (i) Si x_0 n'est pas basique:
    Il y a une solution faisable avec $x_0 = 0$. On peut donc passer en phase II.
    (ii) Si x_0 est basique et $z_0 < 0$:
    P est infaisable, et on s'arrête.
    (iii) Sinon x_0 est basique et $z_0 = 0$:
    Situation impossible si on fait toujours sortir x_0 en priorité de la base.
(6) Tirer de P_0 un tableau faisable pour P;

Phase II:

(1) Appliquer le simplexe habituel à partir du tableau donné par P_0.

EXERCICE. [**1**, ex 3.9a p. 44]
Maximiser $3x_1 + x_2$
Sous les contraintes:
$x_1 - x_2 \leq -1$
$-x_1 - x_2 \leq -3$
$2x_1 + x_2 \leq 4$
$x_1, x_2 \geq 0$

```
t0:=linopt::Transparent(Chvatal44_39a0)
t1:=linopt::Transparent::userstep(t0, slk[2], x0)
t2:=linopt::Transparent::userstep(t1, slk[1], x1)
t3:=linopt::Transparent::userstep(t2, x0, x2)
t0:=linopt::Transparent(Chvatal44_39a)
t1:=linopt::Transparent::userstep(t0, slk[1], x1)
t2:=linopt::Transparent::userstep(t1, slk[2], x2)
t3:=linopt::Transparent::userstep(t2, slk[3], slk[2])
```

2.4.5. Le théorème fondamental de la programmation linéaire.

THÉORÈME. *Tout programme linéaire P sous forme standard a l'une des propriétés suivantes:*

(1) Si P n'a pas de solutions optimales, alors P est infaisable ou non borné;
(2) Si P a une solutions faisable, alors P a une solution basique faisable;
(3) Si P a une solution optimale, alors P a une solution basique optimale.

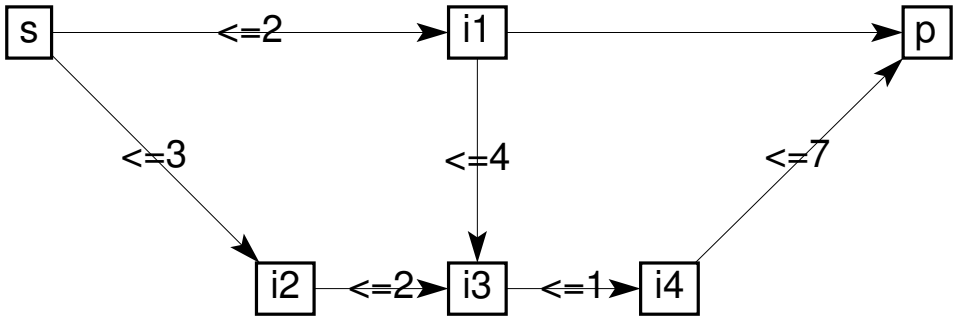
CHAPTER 3

Problèmes de flot maximum

3.1. Introduction

DÉFINITION. *Problème de flot max:*

- Réseau avec source et puits
- Pas de coûts sur les arcs
- Contraintes de capacités sur les arcs
- Production et consommation nulle sur chaque noeud intermédiaire



Objectif:
Maximiser le *volume* du flot, c'est-à-dire la quantité transportée entre s et p.

EXEMPLE. Un dimanche soir, maximiser le nombre de voitures allant d'Albertville à Lyon, en les répartissant entre les différentes routes possibles.

EXERCICE. Mettre le problème de flot dessiné ci-dessus sous forme de programme linéaire.

Clairement, cela se généralise à tout problème de flot max.

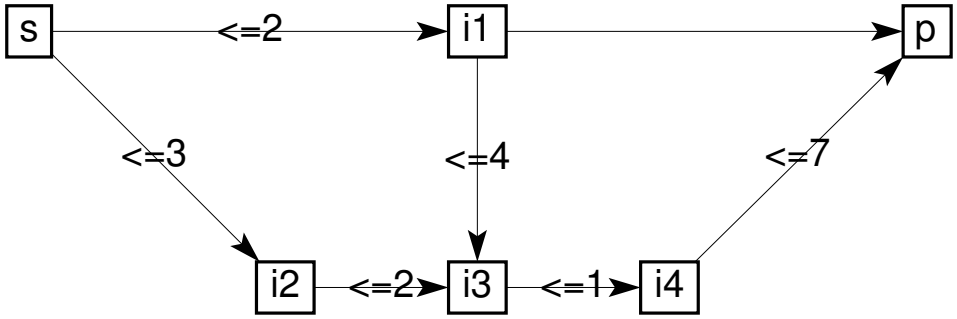
PROBLÈME. Que peut-on en déduire ?

- On a un algorithme de résolution (simplexe)
- On doit bien avoir une dualité!

3.2. Coupes dans un problème de flot

DÉFINITION. Une *coupe* C dans un réseau est un ensemble de sommets du réseau contenant la source.
La capacité de la coupe C est la somme des capacités des arrêtes sortantes de C.

EXEMPLE. Dans notre réseau, la coupe $C = \{s\}$ est de capacité 5.



EXERCICE. Quelle est la capacité de la coupe $C = \{s, i_2, i_3\}$?
Que peut-on en déduire sur la valeur d'un flot ?

PROPOSITION. *Pour toute coupe C et tout flot F dans un réseau, la capacité $|C|$ de la coupe est supérieure au volume $|F|$ du flot: $|C| \geq |F|$.*

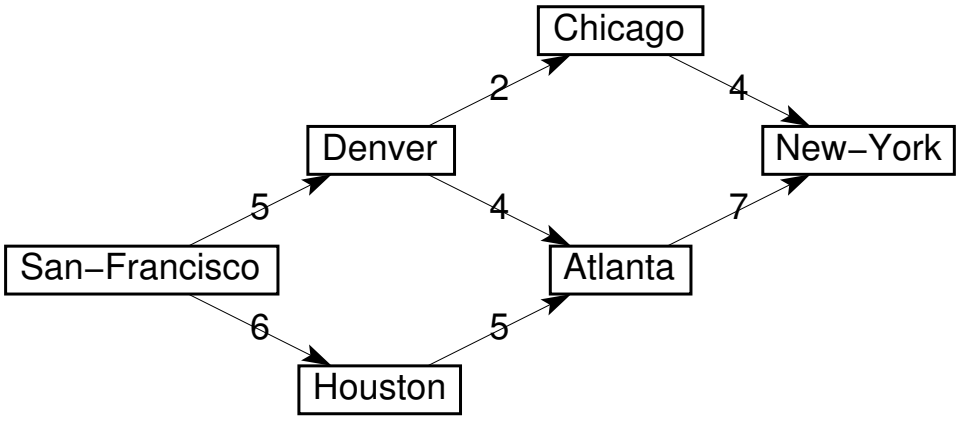
EXERCICE. Chercher un flot maximal et une coupe minimal dans notre exemple?
Que peut-on dire?

3.3. Algorithme de Ford-Fulkerson

Nous allons maintenant donner un algorithme qui permet de calculer un flot maximal.

Mieux, comme l'algorithme du simplexe, cet algorithme va donner des résultats théoriques!

EXEMPLE. On veut transporter le plus grand nombre possible de voyageurs de San-Francisco à New-York, sachant qu'il ne reste que quelques places dans les avions entre les villes suivantes:



DÉFINITION. Soit R un réseau, et F un flot donné dans ce réseau.

Un chemin allant de la source s au puits p est *F-augmentant* si pour chaque arête ij du chemin on a:

- $x_{ij} < u_{ij}$ si l'arc ij est dans le même sens que dans le réseau
- $x_{ij} > 0$ si l'arc ij est dans le sens inverse du réseau.

À partir d'un chemin F-augmentant, on peut construire un nouveau flot F' qui sera de volume strictement plus gros.

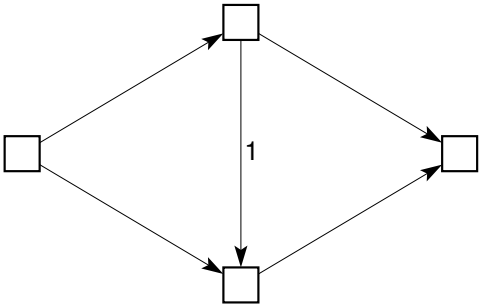
Le principe de l'algorithme de Ford-Fulkerson est de partir d'un flot F quelconque, et de l'améliorer itérativement en recherchant des chemins F-augmentant.

À chaque étape, la recherche d'un chemin F-augmentant se fait par un parcours en profondeur, de manière similaire à la recherche d'un chemin M-augmentant dans un graphe biparti. Si cette recherche échoue, elle dévoile une coupe de capacité égale au flot, ce qui donne un certificat d'optimalité du flot.

REMARQUE 3.3.1. On peut toujours initialiser l'algorithme avec un flot nul.

Si toutes les capacités sont entières et finies, chaque itération augmente le flot d'au moins 1. Cet algorithme ne peut donc pas cycler, et il termine en un nombre fini d'étapes.

Avec une mauvaise stratégie, et des capacités infinies ou non-entières, l'algorithme peut ne pas terminer.



Avec une stratégie convenable, cet algorithme est en fait polynomial, en $O\left(n^3\right)$, même si les capacités sont infinies ou non entières.

Pour les réseaux avec peu d'arcs, il y a des algorithmes plus compliqués qui permettent d'obtenir d'encore meilleurs bornes. Cf. [**1**, p. 369] pour les détails.

3.4. Conséquences de l'algorithme de Ford-Fulkerson

3.4.1. Théorème de dualité min-max.

THÉORÈME. *(Coupe min-Flot max) Dans un réseau, le volume maximal d'un flot est égal à la capacité minimale d'une coupe.*

Ce théorème est un cas particulier du théorème de dualité de la programmation linéaire.

3.4.2. Théorème d'intégralité. Dans notre exemple, on veut transporter des personnes *entières* autant que possible!

Qu'aurait-il pu se passer si l'on avait utilisé l'algorithme du simplexe?

Qu'a t-on obtenu avec l'algorithme de Ford-Fulkerson?

Est-ce un accident?

THÉORÈME. *(dit d'intégralité) Soit P un problème de flot où les contraintes sont entières. Alors:*

(1) *Si P a une solution, alors il a une solution à coefficients entiers;*
(2) *Si P a une solution optimale, alors il a une solution optimale à coefficients entiers.*

PROOF. La solution initiale 0 est à coefficients entiers

À chaque étape, l'amélioration le long du chemin augmentant est entière ou infinie. □

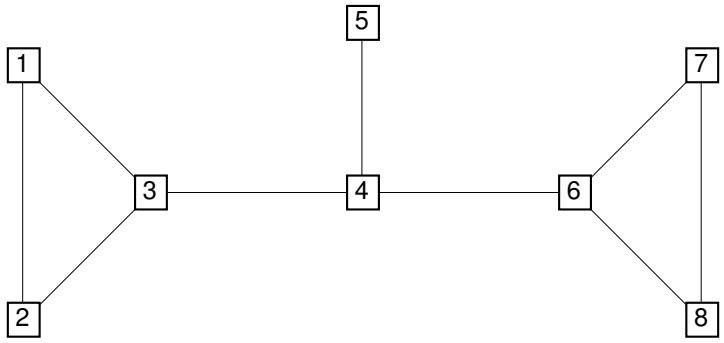
Le théorème d'intégralité est assez simple. Alors quel est son intérêt ?

Ce que dit fondamentalement le théorème d'intégralité, c'est que dans certains cas les méthodes de programmation linéaire peuvent être utilisées pour résoudre des problèmes purement combinatoire, ce qui est loin d'être trivial!

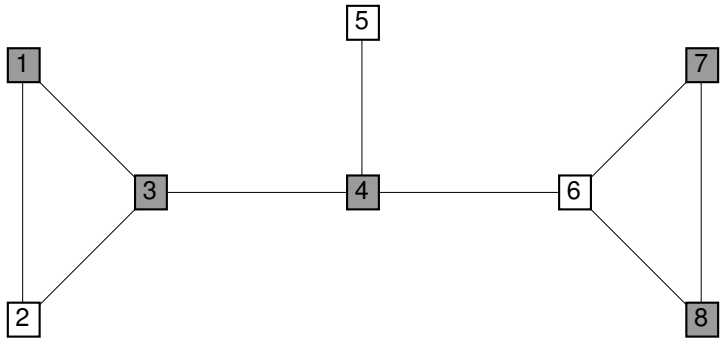
C'est le sujet de la *combinatoire polyhédrale*. On verra quelques exemples la semaine prochaine.

3.4.3. Combinatoire polyhédrale: quelques exemples.

3.4.3.1. *Couvertures et couplages dans les graphes bipartis.* On va maintenant regarder une application de la programmation linéaire pour étudier des graphes non orientés comme le suivant:



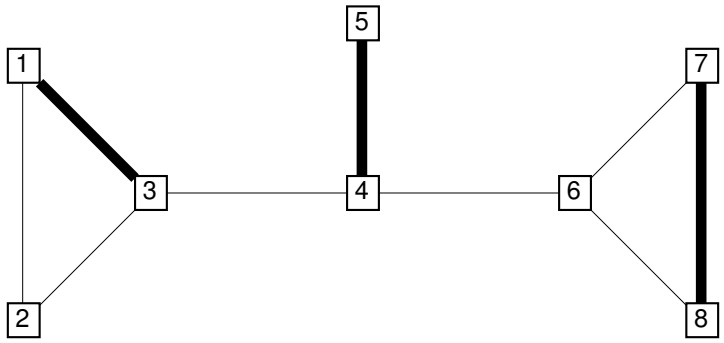
Une *couverture* C de ce graphe est un ensemble de sommets qui touchent toutes les arêtes, comme par exemple $C := \{1, 3, 4, 7, 8\}$:



EXEMPLE 3.4.1. On a 8 petits villages reliés par des routes. En cas d'accident de la route, on veut que les pompiers puissent intervenir rapidement. Le prefet impose que lorsqu'une route relie deux villages, il y ait une caserne de pompier dans au moins l'un des deux villages. Évidemment le budget est serré, donc on veut construire des casernes de pompier dans un nombre minimal de villages.

Modélisation: Chaque village est représenté par un sommet du graphe précédent, les arêtes représentant les routes. Résoudre notre problème revient à chercher une couverture de taille minimale du graphe.

Un *couplage* M de ce graphe est un ensemble d'arêtes qui ne se touchent pas, comme par exemple $M := \{\{1,3\},\{4,5\},\{7,8\}\}$:



EXEMPLE 3.4.2. On veut loger un groupe de 8 personnes dans un hotel, avec des chambres simples et doubles. Pour minimiser les dépenses, on utiliser le maximum de chambres doubles. D'un autre côté on ne veut pas forcer deux personnes qui ne se connaissent pas bien à partager une chambre.

Modélisation: chaque sommet du graphe précédent représente une personne, et chaque arête relie deux personnes qui se connaissent bien. Résoudre notre problème revient alors à rechercher un couplage de taille maximale dans le graphe.

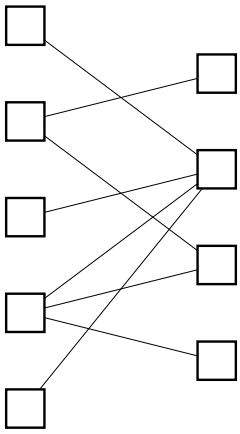
EXERCICE 4. Montrer que pour un couplage M et une couverture C d'un même graphe, on a toujours $|M| \leq |C|$.

PROOF. Comme C est une couverture, chaque arête de M devra être touchée par au moins un sommet dans C. De plus, M étant un couplage, chaque sommet de C touche au plus une arête de M. Donc, on a bien $|M| \leq |C|$. □

PROBLEM 3.4.3. Peut-on trouver M et C de façon à avoir égalité ?

Dans notre exemple, non. Par contre, on va voir que pour certaines classes de graphe, cela va être vrai: on aura un théorème de dualité min-max. Comme par hasard, c'est une conséquence de la programmation linéaire.

On appelle *graphe biparti* un graphe dont on peut partitioner les sommets en deux paquets A et B de sorte que toutes les arêtes soient entre A et B:



EXERCICE 5. On veut rechercher un couplage maximal du graphe précédent. Montrer comment on peut résoudre ce problème en utilisant un problème de flot.

Utiliser l'algorithme de Ford-Fulkerson pour le résoudre.

On appelle cela la "Méthode du chemin augmentant". Pourquoi?

Chaque solution entière F du problème de flot correspond à un couplage M du graphe biparti (les arêtes sur lesquelles passent une unité), avec $|M| = |F|$. Maximiser le flot revient à rechercher un couplage de taille max. *Le théorème d'intégralité nous garanti que Ford-Fulkerson donnera bien une solution optimale entière*!

EXERCICE 6. Prendre maintenant la coupe minimale donnée par l'algorithme de Ford-Fulkerson, et définir C l'ensemble des sommets a du graphe biparti tels que:

- soit a est du côté de la source et a n'est pas dans la coupe.
- soit a est du côté du puit et a est dans la coupe.

(1) Que constate-t'on?

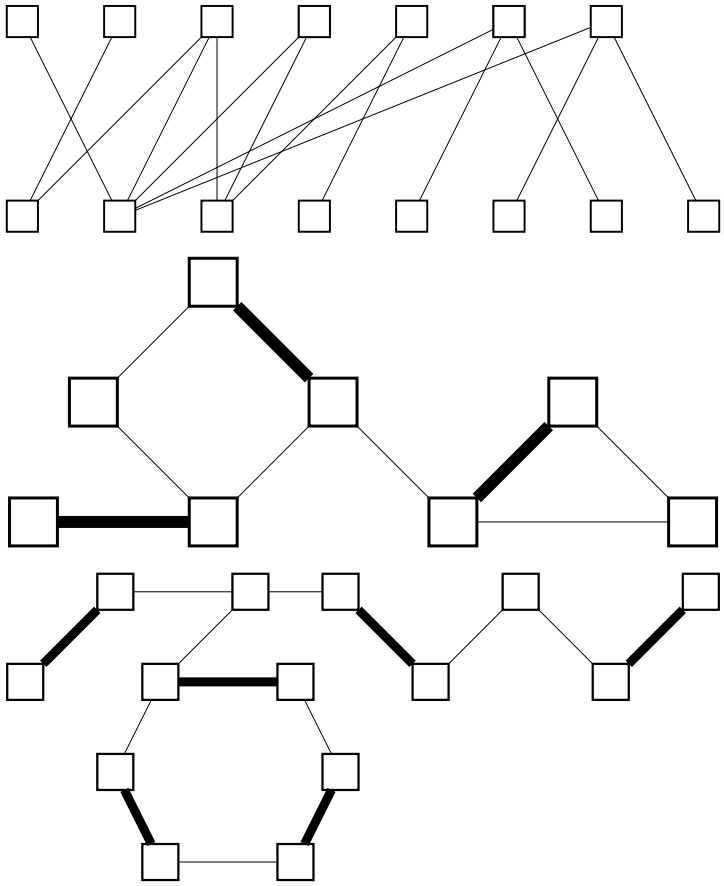
PROBLEM 3.4.4. Est-ce une coïncidence?

EXERCICE 7. Soit G un graphe biparti, M un couplage maximal donné par l'algorithme de Ford-Fulkerson, et C l'ensemble de sommets obtenu comme ci-dessus. Montrer que C est une couverture et que $|M| = |C|$.

THEOREM 3.4.5. *(König-Egerváry) Dans tout graphe biparti, la taille d'un couplage maximal est égale à la taille d'une couverture minimale.*

C'est une exemple typique où le théorème de dualité de la programmation linéaire donne un théorème min-max reliant deux problèmes combinatoires qui ne sont pas en lien direct a priori.

EXERCICE 8. Utiliser la méthode du chemin augmentant pour rechercher un couplage maximal des graphes suivants (on pourra partir du couplage suggéré):



3.4.3.2. *Dualités chaînes/antichaînes dans les ordres partiels; théorème de Dilworth.*

PROBLEM 3.4.6. [**1**, p. 338] Problème des visites guidées. Une compagnie propose 7 visites guidées dans la journée, notées a, b, c, d, e, f, g, dont les horaires et durées sont fixées. Si une visite (par ex. a) termine suffisament avant une autre (par exemple c), le guide de la première visite peut enchaîner sur la deuxième; on notera alors $a \to c$. En l'occurence, voici tous les enchaînements possibles:

$a \to c,\ a \to d,\ a \to f,\ a \to g,\ b \to c,\ b \to g,\ d \to g,\ e \to f,\ e \to g.$

- Combien faut-il de guides au minimum dans cet exemple ?
- Comment trouver le nombre minimum de guides nécessaires dans le cas général ?

DÉFINITION 3.4.7. Soit $P = (E, <)$ un ordre partiel.

Une *chaîne* C de P est un ensemble de sommets de P deux à-deux comparables:

$$x \in C \text{ et } y \in C \ \Rightarrow\ x < y \text{ ou } y < x.$$

Une *antichaîne* A de P est un ensemble de sommets deux-à-deux incomparables.

Une *couverture en chaînes* de P est un ensemble $C_1, \ldots, C_k$ de chaînes, de sorte que tout sommet de P est dans une unique chaîne C_i.

Une *couverture en antichaînes* de P est un ensemble $A_1, \ldots, A_k$ d'antichaînes, de sorte que tout sommet de P est dans une unique antichaîne A_i.

EXERCICE 9. Trouver dans l'ordre partiel P précédent:

(1) Une chaîne de taille maximale
(2) Une antichaîne de taille maximale
(3) Une couverture en chaînes de P de taille minimale
(4) Une couverture en antichaînes de P de taille minimale

Que remarquez vous ?

Y-aurait-il un théorème min-max reliant la taille de la plus grande chaîne et la taille de la plus petite couverture en antichaînes ? Et un autre reliant la taille de la plus grande antichaîne et celle de la plus petite couverture en chaînes ?

EXERCICE 10. Soit P un ordre partiel quelconque.

(1) Soit C une chaîne de P et $A_1, \ldots, A_k$ une couverture de P en antichaînes. Montrer que $|C| \leq k$.
(2) Soit A une antichaîne de P et $C_1, \ldots, C_k$ une couverture de P en chaînes. Montrer que $|A| \leq k$.

PROPOSITION 3.4.8. *Soit P un ordre partiel. La taille de la plus grande chaîne de P est égale à la taille de la plus petite couverture en antichaînes de P.*

EXERCICE 11. Prouvez-le!

Le théorème dans l'autre sens est plus difficile et bien plus profond. Il n'y a pas de construction élémentaire de l'antichaîne et de la couverture en chaîne idoine. On va en fait se ramener au théorème de dualité de la programation linéaire (surprise).

THEOREM 3.4.9. *(Dilworth) Soit P un ordre partiel. La taille de la plus grande antichaîne de P est égale à la taille de la plus petite couverture en chaînes de P.*

PROOF. On note n le nombre de sommets de P.

Choisir une couverture en chaîne de P est équivalent à sélectionner un certain nombre d'arcs dans P, de sorte que chaque sommet ait au plus un arc sortant de sélectionné, et un arc rentrant de sélectionné.

Remarque: s'il y a k chaînes, il y a $n-k$ arcs sélectionnés.

Cela ressemble à un problème de couplage maximal dans un graphe biparti.

On construit un graphe biparti B dans lequel chaque sommet x de P est dupliqué en $(x,1)$ et $(x,2)$.

Chaque fois que $x < y$ dans P, on relie $(x,1)$ et $(y,2)$.

Qu'est-ce qu'un couplage dans B?

Un ensemble d'arcs de P vérifiant exactement les conditions voulues.

Une couverture de P en k chaînes correspond à un couplage de B de taille $n-k$.

Prenons une couverture de P de taille k minimale.

Cela donne un couplage de taille max $n-k$ de B.

Le théorème min-max pour les graphes bipartis indique qu'il y a une couverture de B de même taille: $n-k$ sommets de B qui touchent tous les arcs.

Dans P cela correspond à au plus $n-k$ sommets qui touchent tous les arcs.

Soit A l'ensemble des sommets restants qui est de taille au moins k.

Il ne peut pas y avoir d'arcs entre deux sommets de A.

Conclusion: A est une antichaîne de taille au moins k. □

EXERCICE 12. Suivez le déroulement de la preuve sur l'ordre partiel précédent.

3.4.3.3. *Problème des mines à ciel ouvert.*

PROBLÈME. Des études géologiques ont permis de déterminer précisément la nature du sous-sol, et l'emplacement des gisements orifères à l'endroit ou l'on a décidé de creuser une mine à ciel ouvert. Certains gisements sont profonds, et il n'est pas clair qu'il soit rentable d'excaver tout le sol au-dessus pour y accéder.

Modèle: le sous-sol a été délimité en un certain nombre de blocs. Pour chaque bloc i, on connaît le coût C_i d'excavation, et le profit P_i que l'on peut escompter de son traitement.

Au final, on associe à chaque bloc i la quantité $w_i = C_i - P_i$. Si l'on ne considère pas les autres blocs, il est rentable de creuser i si et seulement si $w_i < 0$.

On veut déterminer quels blocs on doit creuser pour maximiser le profit total $-\sum_i w_i$ (ou autrement dit minimiser $\sum_i w_i$).

Maintenant, il y a des contraintes supplémentaires: si un bloc i est sous un bloc j, on ne peux pas creuser i sans creuser j!

On introduit un ordre partiel, de sorte que $i < j$ si pour creuser i on doit creuser j.

Comme on le verra, la forme des blocs, et le type d'ordre partiel n'est pas relevant.

EXEMPLE. On considère le sous-sol suivant:

Comment modéliser notre problème sous forme de problème de flot max ?

La modélisation des contraintes de précédences est un peu astucieuse!

On introduit le réseau suivant:

C'est la remarque suivante qui va faire marcher la machine:

REMARQUE. Soit C une coupe.

S'il existe deux blocs $i < j$, avec $i \in C$ et $j \in C$, alors C est de capacité infinie.

La réciproque est vraie.

Les coupes de capacité finie sont en correspondance avec les coupes respectant les contraintes.

Maintenant, on peut vérifier que la capacité d'une coupe finie vaut exactement

$$\sum_{i \in C,\ i \text{ bloc non rentable}} w_i - \sum_{i \notin C,\ i \text{ bloc rentable}} w_i.$$

Quitte à rajouter le terme constant $\sum_{i,\ i \text{ bloc}} w_i$, on est en train de calculer le profit lorsque l'on enlève les blocs i avec $i \in C$.

Résumé: Soit I un ensemble de blocs, et C la coupe $\{s\} \cup I$.

- Si I ne satisfait pas les contraintes, la capacité de C est infinie.
- Si I satisfait les contraintes, la capacité de C est l'opposé du profit.

Maximiser le profit revient à trouver une coupe min.

REMARQUE. En termes pédants: on peut résoudre par un algorithme de flot le problème de trouver une section finale de poids minimal dans un ordre partiel.

3.5. Synthèse

Plusieurs modèles généraux pour faire de l'optimisation:

(1) Programmation linéaire
   (a) Algorithme du simplexe
   Efficace en pratique (quasiment linéaire), non polynomial en théorie
   (b) Algorithme de l'élipsoide
   Polynomial, mais non efficace en pratique
   (c) Méthode des points intérieurs
   Plus ou moins efficace que le simplexe selon les cas
   (d) Théorème de dualité $\Longrightarrow$ Certification, optimisation, coûts marginaux, ...

   (e) Mais: solutions dans $\mathbb{Q}$
(2) Problèmes de flots
   (a) Algorithme de Ford-Fulkerson
   Polynomial $O(n^3)$. Plus efficace que le simplexe.
   (b) Théorème de dualité (flots/coupes)
   (c) Théorème d'intégralité
   $\Longrightarrow$ Algorithmes et théorèmes min-max sur des problèmes discrets.
(3) Réseaux de transports
   (a) Algorithme du simplexe pour les réseaux
   (b) Théorème de dualité (coûts marginaux)
   (c) Théorème d'intégralité

Bibliography

[1] V. Chvatal. Linear Programming.

[2] R. Vanderbie. Linear Programming; Foundations and Extensions. http://www.princeton.edu/~rvdb/LPbook/index.html

[3] Linear Programming FAQ http://rutcor.rutgers.edu/~mnk/lp-faq.html

[4] http://en.wikipedia.org/wiki/Linear_programming