Algorithmique 2 (L3)

Serge Haddad
Professeur de l'ENS Cachan
61, Avenue du Président Wilson
94235 Cachan cedex, France
adresse électronique : haddad@lsv.ens-cachan.fr
page personnelle : www.lsv.ens-cachan.fr/∼haddad/

26 janvier 2016

Table des matières

Chapitre 1

Recherche de chaînes de caractères

Ouvrages recommandés : [Crochemore et Rytter] [Beauquier et al, chapitre 10]

1.1 Un algorithme naïf

Dans ce chapitre, nous nous intéressons au problème de la recherche d'un motif, noté P, dans un texte noté T. L'algorithme peut renvoyer soit la première occurrence du motif s'il en existe une, soit la liste des occurrences du motif.

Le texte est vu comme un tableau de caractères de dimension n et le motif comme un tableau de caractères de dimension $m \leq n$. Soit un tableau X, $X[i,j]$ (pour $i \leq j$) désigne le sous-tableau débutant à l'indice i et se terminant à l'indice j. Avec cette notation, on recherche les indices i tels que $i + m - 1 \leq n$ et $T[i,i + m - 1] = P$. On dira aussi que la chaîne P apparaît avec un décalage $i - 1$ dans T.

L'alphabet est noté Σ et les chaînes sont vues (alternativement) comme des éléments de Σ^*. La longueur d'une chaîne x est notée $|x|$. On note ε la chaîne vide. La concaténation de deux chaînes x et y est notée xy. On dit que w est un *préfixe* de x s'il existe y tel que $x = wy$ ce qu'on note $w \sqsubset x$. On dit que w est un *suffixe* de x s'il existe y tel que $x = yw$ ce qu'on note $w \sqsupset x$. Ces deux relations sont transitives et on a le lemme suivant dont la preuve est laissée au lecteur.

Lemme 1 *Soient x,y,$z \in \Sigma^*$, alors :*
- *Si $x \sqsubset z$, $y \sqsubset z$ et $|x| \leq |y|$ (resp. $|x| = |y|$) alors $x \sqsubset y$ (resp. $x = y$).*
- *Si $x \sqsupset z$, $y \sqsupset z$ et $|x| \leq |y|$ (resp. $|x| = |y|$) alors $x \sqsupset y$ (resp. $x = y$).*

L'algorithme naïf consiste à tester tous les indices en vérifiant la condition énoncée plus haut. Puisqu'on a affaire à deux boucles imbriquées, l'algorithme 1 a une complexité en $O((n - m + 1)m)$ et même en $\Theta((n - m + 1)m)$. La borne inférieure de complexité s'obtient avec $T = a^n$ et $P = a^m$.

Algorithme 1: Un algorithme naïf de recherche de motif

`RechercheNaive`(T,P) : une liste
Input : T,P, un texte de n caractères et un motif de $m \leq n$ caractères
Output : L la liste des décalages d'occurrences de P dans T
Data : i,j des indices

```
L ← NULL
for i from 0 to n − m do
    j ← 1
    while j ≥ 1 and j ≤ m do
        if T[i + j] = P[j] then
            j ← j + 1
        else
            j ← 0
        end
    end
    if j = m + 1 then AjoutListe (L,i)
end
return L
```

1.2 Un algorithme à l'aide d'automates finis

1.2.1 L'automate de recherche de motif

Le problème de l'algorithme 1 est l'absence de mémoire des tests précédents effectués lors des comparaisons du motif avec les décalages du texte. L'algorithme que nous étudions maintenant n'examine un caractère du texte qu'une unique fois. ceci est possible grâce à un pré-traitement du motif (indépendamment du texte à lire). La première approche repose sur les automates finis.

Définition 1 *Un automate fini (déterministe complet) sur l'alphabet* Σ, $\mathcal{A} = (Q,q_0,F,\delta)$ *est défini par :*

- Q *un ensemble finis d'états,* $q_0 \in Q$ *l'état initial et* $F \subseteq Q$ *le sous-ensemble des états terminaux.*
- $\delta : Q \times \Sigma \mapsto Q$*, la fonction de transition.*

La fonction de transition s'étend aux mots par les équations :

$$\forall q \in Q\ \forall a \in \Sigma\ \forall x \in \Sigma^*\ \delta(q,\varepsilon) = q \wedge \delta(q,ax) = \delta(\delta(q,a),x))$$

Un mot x est *accepté* par $\mathcal{A}$ si $\delta(q_0,x) \in F$.

L'automate associé au motif P que nous voulons construire a pour états les entiers $0,1,\ldots,m$ avec la signification suivante : l'automate est dans l'état k si après avoir lu une partie du texte $T[1,j]$ la chaîne $T[j-k+1,j]$ est le plus grand suffixe de $T[1,j]$ qui est aussi un préfixe de P ($T[j+1,j]$ représente la chaîne vide). Ainsi l'état 0 est l'état initial et l'état m est l'unique état terminal. Un indice i d'occurrence du motif dans le texte correspond à un indice $i+m-1$ pour lequel l'automate est dans l'état m. Il reste à construire l'automate (à supposer qu'il existe).

Soit x une chaîne, notons $\sigma(x) = \max(k \in \{0,1,\ldots,m\} \mid P[1,k] \sqsupset x)$. Notre objectif est de montrer que $\sigma(xa)$ ne dépend que de $\sigma(x)$ et de a.

Proposition 1 *Soit $x \in \Sigma^*$, $k = \sigma(x)$ et $a \in \Sigma$. Alors $\sigma(xa) = \sigma(P[1,k]a)$.*

Preuve
$P[1,k]a$ est un suffixe de xa ce qui implique par transivité que tout suffixe de $P[1,k]a$ est un suffixe de xa. D'où $\sigma(xa) \geq \sigma(P[1,k]a)$.

Notons $k' = \sigma(xa)$. Si $k' = 0$ alors $\sigma(xa) \leq \sigma(P[1,k]a)$

Si $k' > 0$ alors $P[k'] = a$ et $P[1,k'-1]$ est un suffixe de x. Par définition de σ, $k' - 1 \leq k$. Par conséquent, $P[1,k'-1]$ et $P[1,k]$ sont deux suffixes de x, $P[1,k]$ étant le plus grand. Le lemme 1 implique $P[1,k'-1] \sqsupset P[1,k]$. D'où $P[1,k'] = P[1,k'-1]a \sqsupset P[1,k]a$, ce qui implique $k' \leq \sigma(P[1,k]a)$. Ce cas est illustré par la figure 1.1.

L'égalité est donc établie.

c.q.f.d. ◊◊◊

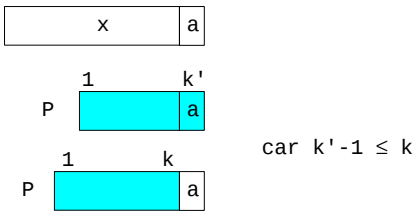


FIGURE 1.1: Illustration de la preuve de la proposition 1

La proposition précédente nous donne aussi le moyen de calculer de l'automate qui est décrit par l'algorithme 2. Cet algorithme a une complexité en $O(m^3|\Sigma|)$. Il y a en effet trois boucles imbriquées, deux en $O(m)$ et une en $O(|\Sigma|)$. De plus, le test de la boucle la plus interne se fait en $O(m)$. Une fois la construction de l'automate effectuée, la recherche de motif se fait en $O(n)$ en appliquant l'automate sur le texte et en ajoutant les indices d'occurrences à chaque fois que l'état terminal est rencontré.

L'automate présenté à la figure 1.2 illustre cette construction avec le motif *ababaca*. Nous avons fusionné les transitions qui avaient même origine et destination dans un souci de lisibilité. Par exemple la transition $5 \xrightarrow{b} 4$ correspond au fait que *abab* est le plus grand préfixe du motif qui est aussi un suffixe de *ababab*.

Algorithme 2: Un algorithme naïf de calcul de fonction de transition

`CalculTransition`(P,Σ) : une fonction de transition
Input : P,Σ, un motif de m caractères sur l'alphabet Σ

Output : δ la fonction de transition de l'automate de recherche

Data : i,j des indices, a une lettre

for i **from** 0 **to** m **do**
  for $a \in \Sigma$ **do**
    $j \leftarrow \min(m,i+1)$
    while not $P[1,j] \sqsupseteq P[1,i]a$ **do** $j \leftarrow j-1$
    $\delta(i,a) \leftarrow j$
  end
end
return δ

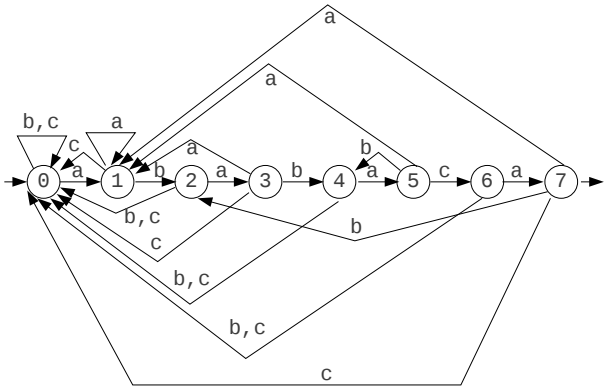


FIGURE 1.2: L'automate associé au motif *ababaca*

1.2.2 Une construction efficace de l'automate de recherche

Dans ce paragraphe, nous améliorons la construction de l'automate de recherche en nous appuyant sur une fonction intermédiaire $\pi : \{1,\ldots,m\} \mapsto \{0,\ldots,m-1\}$ définie par $\pi(k) = \max(k' < k \mid P[1,k'] \sqsupset P[1,k])$.

Par exemple, si $P = ababaca$ alors π est définie ainsi :

i	1	2	3	4	5	6	7
$\pi(i)$	0	0	1	2	3	0	1

Nous établissons une propriété reliant la fonction de transition δ à la fonction π.

Proposition 2 *Soient $a \in \Sigma$ et $k \in \{0, \ldots m\}$, alors :*

1. *Si $k < m \wedge P[k+1] = a$ alors $\delta(k,a) = k+1$*
2. *Si $(0 < k < m \wedge P[k+1] \neq a) \vee k = m$ alors $\delta(k,a) = \delta(\pi(k),a)$*
3. *Si $k = 0 \wedge P[k+1] \neq a$ alors $\delta(k,a) = k$*

Preuve
Appelons $k' = \delta(k,a) = \sigma(P[1,k]a)$. Par définition, $k' \leq k+1$.

Cas n°1. Il découle immédiatement du fait que $P[1,k]a = P[1,k+1]$.

Cas n°2. Puisque $P[1,\pi(k)] \sqsupset P[1,k]$, on déduit que $P[1,\pi(k)]a \sqsupset P[1,k]a$ et que $\sigma(P[1,\pi(k)]a) \leq \sigma(P[1,k]a)$.

Si $0 < k < m$, puisque $a \neq P[k+1]$, $P[1,k']$ est un suffixe propre de $P[1,k]a$. Si $k = m$ alors $P[1,k']$ est un suffixe propre de $P[1,k]a$ puisque $k' \leq k$.

Si $k' = 0$ alors $0 \leq \sigma(P[1,\pi(k)]a) \leq \sigma(P[1,k]a) = 0$ ce qui permet de conclure. Sinon $P[1,k'-1]$ est un suffixe propre de $P[1,k]$ donc un suffixe de $P[1,\pi(k)]$ d'après la définition de π. Par conséquent, $P[1,k']$ est un suffixe de $P[1,\pi(k)]a$ ce qui entraîne $k' \leq \sigma(P[1,\pi(k)]a)$ et par suite $k' = \sigma(P[1,\pi(k)]a)$. Ce cas est illustré par la figure 1.3.

Cas n°3. On sait que k' est soit 0 soit 1. Puisque $a \neq P[1]$, on conclut que $k' = 0$.

c.q.f.d. ◊◊◊

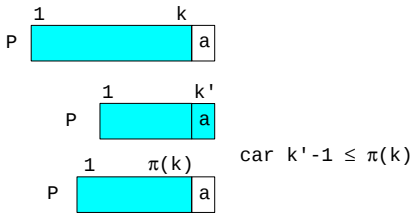


FIGURE 1.3: Illustration de la preuve de la proposition 2

L'algorithme 3 est la transcription de la proposition 2. La complexité de l'algorithme est dominée par les deux boucles imbriquées, soit une complexité en $O(m|\Sigma|)$.

Il reste à calculer la fonction π. On définit inductivement l'ensemble $\pi^*(i)$ par $\pi^*(0) = \{0\}$ et $\forall i > 0$ $\pi^*(i) = \{i\} \cup \pi^*(\pi(i))$.

Proposition 3

$$\forall i \ \pi^*(i) = \{j \mid P[1,j] \sqsupset P[1,i]\}$$

Preuve
Démontrons-le par récurrence. Le cas $i = 0$ est évident. Soit $i > 0$, toute chaîne est un suffixe d'elle-même. Par hypothèse de récurrence, tout élément de $\pi^*(\pi(i))$

Algorithme 3: Un algorithme efficace de calcul de fonction de transition

`CalculTransition`(P,Σ,π) : une fonction de transition
Input : P, un motif de m caractères sur l'alphabet Σ
Input : π, la fonction π du motif

Output : δ la fonction de transition de l'automate de recherche

Data : i un indice, a une lettre

```
for a ∈ Σ do
  | if a = P[1] then δ(0,a) ← 1 else δ(0,a) ← 0
end
for i from 1 to m do
  | for a ∈ Σ do
  |   | if i < m ∧ a = P[i+1] then δ(i,a) ← i+1 else δ(i,a) ← δ(π(i),a)
  | end
end
return δ
```

est « l'indice » d'un suffixe de $P[1,\pi(i)]$, donc par transitivité d'un suffixe de $P[1,i]$. Par conséquent, $\pi^*(i) \subseteq \{j \mid P[1,j] \sqsupset P[1,i]\}$. Soit maintenant $P[1,j]$ un préfixe de $P[1,i]$. Si $j = i$ alors $j \in \pi^*(i)$. Sinon $P[1,j]$ est un préfixe propre de $P[1,i]$. Puisque $P[1,\pi(i)]$ est le plus grand préfixe propre de ce type, le lemme 1 implique que $P[1,j]$ est un préfixe de $P[1,\pi(i)]$, donc par hypothèse de récurrence $j \in \pi^*(\pi(i)) \subseteq \pi^*(i)$. Ce qui conclut la démonstration.

c.q.f.d. ◊◊◊

La proposition suivante est la base du calcul de π car $\pi(i)$ y est défini en fonction des valeurs précédentes de π.

Proposition 4 *Soit* $0 < i \leq m$ *et* $E_i = \{j \mid j \in \pi^*(\pi(i-1)) \wedge P[j+1] = P[i]\}$. *Alors :*

$$\textit{Si } E_i = \emptyset \textit{ Alors } \pi(i) = 0 \textit{ Sinon } \pi(i) = 1 + \max(E_i)$$

Preuve
Si $\pi(i) > 0$ (i.e. $P[1,\pi(i)] \neq \varepsilon$) alors $P[1,\pi(i)-1]$ est un suffixe de $P[1,i-1]$ et $P[\pi(i)] = P[i]$. Donc la proposition 3 implique que $\pi(i) - 1 \in E_i$ et $\pi(i) \leq 1 + \max(E_i)$. Par contraposée, si $E_i = \emptyset$ alors $\pi(i) = 0$.

Si $j \in E_i$ alors en vertu de la définition de E_i et de la proposition 3, $P[1,j+1]$ est un suffixe propre de $P[1,i]$. Donc $\pi(i) \geq 1 + \max(E_i)$.

c.q.f.d. ◊◊◊

L'algorithme 4 de calcul de π s'en déduit facilement. Au début de la boucle externe j est égal à la valeur de $\pi(i-1)$. La boucle interne parcourt $\pi^*(\pi(i-1))$ par valeurs décroissantes en essayant de trouver l'élément maximal de E_i. S'il est trouvé, j est incrémenté (sinon $E_i = \emptyset$ et $j = 0$) suivant la proposition 4. Puis j est affecté à $\pi(i)$.

Analysons la complexité de cet algorithme en nombre d'instructions effectuées. A cette fin nous introduisons, la fonction de potentiel $2j$. Calculons la

complexité amortie d'un tour de la boucle **while** effectué : deux instructions sont exécutées et le potentiel décroît d'au moins 2 puisque $\pi(j) < j$. Autrement dit, un tour de cette boucle a une complexité amortie au plus égale à 0. En ajoutant le dernier test, on obtient une complexité amortie d'au plus 1. La complexité amortie du **if** est d'au plus 4 (2 instructions plus l'incrémentation de j). En ajoutant la dernière instruction, on obtient une complexité amortie d'au plus 6 soit en $O(1)$. Puisque le potentiel est nul à l'entrée de boucle **for** et positif à la sortie donc supérieur ou égal à sa valeur à l'entrée, la complexité de ce calcul est en $O(m)$.

Pour résumer la complexité du pré-traitement est dominée par le calcul de l'automate soit $O(m|\Sigma|)$.

Algorithme 4: Un algorithme de calcul de la fonction π

`CalculPi`(P,Σ) : une fonction
Input : P, un motif de m caractères sur l'alphabet Σ
Output : π, la fonction π du motif
Data : i,j deux indices

$\pi(1) \leftarrow 0; j \leftarrow 0$
for i **from** 2 **to** m **do**
  // $j = \pi(i-1)$
  while $j > 0 \wedge P[j+1] \neq P[i]$ **do** $j \leftarrow \pi(j)$
  if $P[j+1] = P[i]$ **then** $j \leftarrow j+1$
  $\pi(i) \leftarrow j$
end
return π

1.3 L'algorithme de Morris-Pratt

Lorsque l'alphabet Σ est de taille importante, par exemple avec des alphabets enrichis par des styles et stockés sur deux octets (conduisant à une taille $\approx 2^{16}$), le temps de pré-traitement peut devenir prohibitif. Or ce temps semble incompressible puisqu'il faut construire un automate déterministe complet à m états sur l'alphabet Σ.

L'idée sous-jacente de l'algorithme de Morris-Pratt est de substituer au modèle des automates finis, un modèle plus concis que nous appellerons ici automate *étendu*. Les transitions d'un automate étendu diffèrent de celles d'un automate fini sur deux points :

— Les transitions sont étiquetées par des formules booléennes dont les propositions atomiques sont les lettres de l'alphabet. Etant donnée une formule φ, on notera $[\varphi]$ le sous-ensemble des lettres qui satisfont φ où la satisfaction est obtenue par induction à partir de $a \models a$ et $a \not\models b$ pour $a \neq b \in \Sigma$. On note $\mathcal{F}(\Sigma)$ l'ensemble de ces formules.

— On associe à une transition un déplacement dans le mot lu : soit $\rightarrow$ pour passer à la lettre suivante, soit $\uparrow$ pour rester sur la lettre courante. On note $\mathcal{D} = \{\rightarrow, \uparrow\}$ l'ensemble des déplacements.

Définition 2 *Un automate étendu sur l'alphabet* Σ, $\mathcal{A} = (Q,q_0,F,\delta)$ *est défini par :*

- Q *un ensemble finis d'états,* $q_0 \in Q$ *l'état initial et* $F \subseteq Q$ *le sous-ensemble des états terminaux.*
- $\delta \subseteq Q \times \mathcal{F}(\Sigma) \times \mathcal{D} \times Q$*, la relation de transition. Soit une transition* (q,φ,d,q')*, on la note* $q \xrightarrow{\varphi,d} q'$
- *Soit un circuit de transitions* $q_0 \xrightarrow{\varphi_1,\uparrow} q_1,\ldots,q_{n-1} \xrightarrow{\varphi_n,\uparrow} q_n = q_0$*. Alors* $\bigcap_{i=1}^{n}[\varphi_i] = \emptyset$*.*

Le troisième point garantit qu'il ne peut y avoir de calcul infini. Un automate étendu est dit déterministe complet si étant données $\varphi_1,\ldots,\varphi_k$ les formules étiquetant les transitions sortantes d'un état quelconque, $[\varphi_1],\ldots,[\varphi_k]$ constituent une partition de Σ.

Nous définissons la fonction de transition dans le cas d'automate étendu *déterministe complet* que nous notons aussi δ par abus de langage. Soit q un état de l'automate et w un mot de Σ^*, alors :

- Si $w = \varepsilon$ alors $\delta(q,w) = q$.
- Si $w = aw'$ et si $q \xrightarrow{\varphi,d} q'$ avec $a \models \varphi$ (il existe toujours une unique transition de ce type) alors $\delta(q,w) = \delta(q',w'')$ où $w'' = w'$ si $d = \rightarrow$ et $w' = w$ si $d = \uparrow$.

En raison de la contrainte introduite sur les automates étendus, cette définition est valide. Un mot w est *accepté* par $\mathcal{A}$ si $\delta(q_0,w) \in F$.

La proposition 2 nous fournit tous les éléments pour construire un automate étendu dont les mots acceptés sont exactement ceux de l'automate des motifs. L'ensemble des états est $\{0,1,\ldots,m\}$ avec $q_0 = 0$ et $F = \{m\}$. Les transitions sont définies ainsi :

1. Pour tout $k < m$, on a une transition $k \xrightarrow{P[k+1],\rightarrow} k+1$.
2. Pour tout $0 < k < m$, on a une transition $k \xrightarrow{\neg P[k+1],\uparrow} \pi(k)$.
3. On a une transition $m \xrightarrow{\mathbf{true},\uparrow} \pi(m)$.
4. On a une transition $0 \xrightarrow{\neg P[1],\rightarrow} 0$.

L'automate étendu présenté à la figure 1.4 illustre cette construction avec le motif *ababaca*.

Pour construire cet automate étendu, on calcule la fonction π en $O(m)$ puis les transitions décrites ci-dessus également en $O(m)$. Le temps total de prétraitement est donc en $O(m)$. Néanmoins le temps de recherche de motif n'est *a priori* pas en $O(n)$ puisque les transitions sans déplacement doivent être prises en compte.

Appelons k l'état final de l'automate, n^+ le nombre de transitions parcourues qui ont incrémenté l'état, n^- le nombre de transitions parcourues qui ont décrémenté l'état (i.e. celles qui ne consomment pas la lettre lue) et $n^=$ le nombre de transitions parcourues qui ont laissé l'état inchangé (la boucle autour de l'état 0). On a $n = n^+ + n^=$.

Remarquons qu'une transition ne peut incrémenter l'état qu'au plus d'une unité. Par conséquent, $n^+ - n^- \geq k$. Examinons la dernière transition. Il s'agit de la transition de déplacement associée à la consommation du dernier caractère.

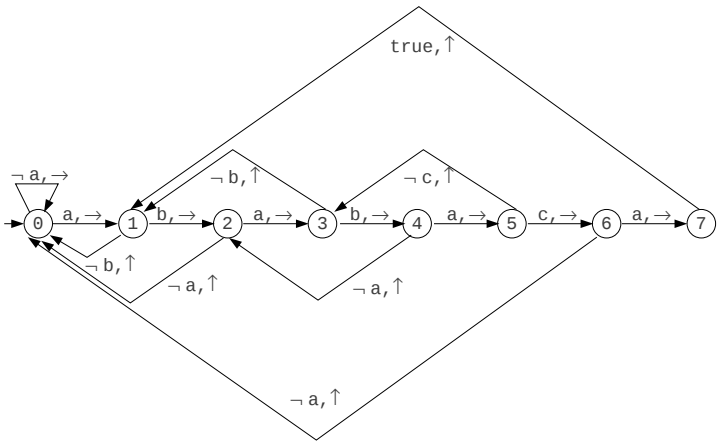


FIGURE 1.4: L'automate MP associé au motif *ababaca*

— S'il s'agit de la boucle en 0, alors $n^{=} > 0$ et $n^{+} - n^{-} \geq 0$.
D'où $n^{=} + n^{+} + n^{-} \leq n^{=} + 2n^{+} < 2n^{=} + 2n^{+} = 2n$.
— S'il ne s'agit pas de la boucle en 0, alors $k > 0$ et $n^{+} - n^{-} > 0$.
D'où $n^{=} + n^{+} + n^{-} < n^{=} + 2n^{+} \leq 2n^{=} + 2n^{+} = 2n$.

Il y a donc au plus $2n - 1$ transitions parcourues et par conséquent un temps de traitement en $O(n)$.

La forme très particulière de cet automate étendu permet de coder l'algorithme de recherche du motif en calculant uniquement la fonction π et en « codant » la recherche du motif par l'automate dans le programme. C'est ce que fait l'algorithme 5 de Morris-Pratt. La recherche de la transition sortante d'un état se fait par une comparaison. Par conséquent, il y au plus $2n - 1$ comparaisons.

1.4 L'algorithme de Knuth-Morris-Pratt

Examinons l'automate de la figure 1.4. La transition de l'état 4 à l'état 2 sera nécessairement suivie de la transition de l'état 2 à l'état 0, puis de la boucle en 0. En généralisant cet exemple, sans changer le comportement de l'algorithme, pour toute lettre a il est intéressant de substituer :

— à un chemin maximal de transitions $k_0 \xrightarrow{\neg a,\uparrow} k_1 \xrightarrow{\neg a,\uparrow} \dots 0 \xrightarrow{\neg a,\rightarrow} 0$, une unique transition $k_0 \xrightarrow{\neg a,\rightarrow} 0$.
— un chemin maximal de transitions $k_0 \xrightarrow{\neg a,\uparrow} k_1 \xrightarrow{\neg a,\uparrow} \dots k_n$, une unique transition $k_0 \xrightarrow{\neg a,\uparrow} k_n$.

Nous avons présenté à la figure 1.5 l'automate obtenu. La construction de cet automate peut se faire efficacement et l'implémentation de la reconnaissance d'un texte par cet automate n'est rien d'autre que l'algorithme de Knuth-Morris-Pratt.

Algorithme 5: L'algorithme de Morris-Pratt

`MP`(T,P,π) : une liste
Input : T,P, un texte de n caractères et un motif de $m \leq n$ caractères
Input : π, la fonction π du motif
Output : L la liste des décalages d'occurrences de P dans T
Data : i,j deux indices

```
// j est l'état courant et T[i] est le caractère lu
```

$L \leftarrow$ NULL ; $j \leftarrow 0$
for i **from** 1 **to** n **do**
  while $j > 0 \wedge P[j+1] \neq T[i]$ **do** $j \leftarrow \pi(j)$
  if $P[j+1] = T[i]$ **then** $j \leftarrow j+1$
  if $j = m$ **then**
    `AjoutListe` (L,$i-m$)
    $j \leftarrow \pi(j)$
  end
end
return L

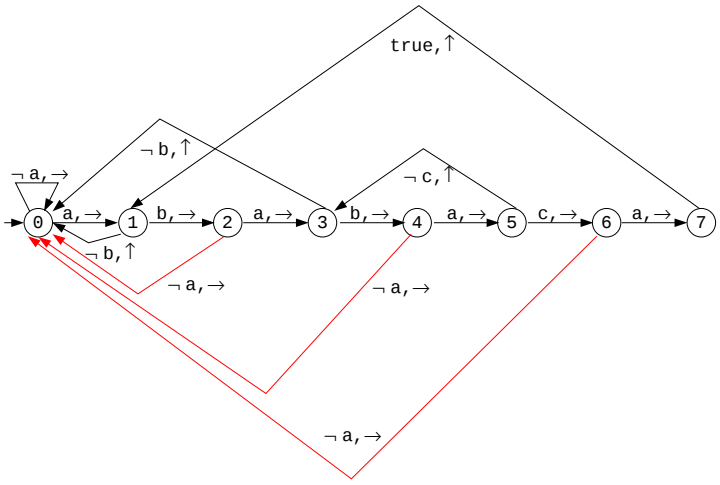


FIGURE 1.5: L'automate KMP associé au motif *ababaca*

Cet automate est basé sur une fonction π' qui est un raffinement de la fonction π. La fonction $\pi' : \{0,1,\ldots,m\} \mapsto \{-1,0,\ldots,m-1\}$ inclut -1 comme valeur possible qui signifie que l'état atteint est 0 et que le caractère courant doit être consommé.

— $\pi'(m) = \pi(m)$ car l'arc sortant de m est le seul étiqueté par `true`.
— $\pi'(0) = -1$ car le caractère courant doit être consommé.
— Soit $0 < k < m$.

Définissons $E'_k = \{k' < k \mid P[1,k'] \sqsupset P[1,k] \wedge P[k'+1] \neq P[k+1]\}$.
Si $E'_k = \emptyset$ alors $\pi'(k) = -1$ car il n'y a pas de préfixe non vide de P qui puisse coïncider avec un suffixe de $P[1,k]c$ pour c quelconque mais différent de $P[k+1]$.
Si $E'_k \neq \emptyset$ alors $\pi'(k) = \max(E'_k)$ car $P[1,\pi'(k)+1]$ est le plus grand préfixe non vide de P qui puisse coïncider avec un suffixe de $P[1,k]c$ pour c quelconque mais différent de $P[k+1]$.

Par exemple, si $P = ababaca$ alors π' est définie ainsi :

i	0	1	2	3	4	5	6	7
$\pi'(i)$	-1	0	-1	0	-1	3	-1	1

L'algorithme 6 est dû à Knuth-Morris-Pratt. Il est très similaire à l'algorithme précédent avec la fonction π' substituée à π et le test à sortie du **while** supprimé grâce à l'utilisation de la valeur -1.

Algorithme 6: L'algorithme de Knuth-Morris-Pratt

```
KMP(T,P,π') : une liste
Input : T,P, un texte de n caractères et un motif de m ≤ n caractères
Input : π', la fonction π' du motif
Output : L la liste des décalages d'occurrences de P dans T
Data : i,j deux indices
// j est l'état courant et T[i] est le caractère lu
L ← NULL ; j ← 0
for i from 1 to n do
    while j ≥ 0 ∧ P[j+1] ≠ T[i] do j ← π'(j)
    j ← j + 1
    if j = m then
        AjoutListe (L,i − m)
        j ← π'(j)
    end
end
return L
```

Intéressons-nous maintenant au calcul de la fonction π'. On observe que, par définition $E'_k \subseteq \pi^*(k) \setminus \{k\}$. Par conséquent, un calcul à partir de la fonction π est immédiat.

Lemme 2 *Soit* $0 < k < m$.
Si $P[k+1] \neq P[\pi(k)+1]$ *alors* $\pi'(k) = \pi(k)$ *sinon* $\pi'(k) = \pi'(\pi(k))$.

Preuve
Observons que si $\max(E'_k)$ existe alors c'est le plus grand élément k' de $\pi^*(k) \setminus \{k\}$ qui vérifie $P[k'+1] \neq P[k+1]$. Donc soit $P[k+1] \neq P[\pi(k)+1]$ et $\pi'(k) = \pi(k)$ soit $P[k+1] = P[\pi(k)+1]$ et le plus grand élément est à chercher dans $\pi^*(\pi(k)) \setminus \{\pi(k)\}$ avec la condition $P[k'+1] \neq P[\pi(k)+1]$.

c.q.f.d. ◊◊◊

Il nous suffit de compléter le calcul de π par le calcul simultané de π'. On peut faire encore mieux car le calcul simultané de π' permet d'accélérer le calcul de π !

Lemme 3 *Soit* $0 < k < m$.

$$\pi(k+1) = 1 + \pi'^{(h)}(\pi(k))$$

où h *est le plus petit entier qui vérifie l'une des conditions suivantes :*

1. $1 + \pi'^{(h)}(\pi(k)) = 0$
2. $1 + \pi'^{(h)}(\pi(k)) \neq 0 \wedge P[1 + \pi'^{(h)}(\pi(k))] = P[1+k]$

Preuve
Posons $k' = \pi(k)$, $a = P[1+k]$ et $b = P[1+k']$. Si $a = b$ alors $\pi(k+1) = 1+\pi(k)$, sinon il faut trouver le plus grand élément de $\pi^*(k)$ qui se prolonge par un $a \neq b$. Le plus grand élément potentiel est $\pi'(k)$ et ainsi de suite puisqu'il faut à chaque fois prolonger le préfixe par a différent de la dernière lettre testée.

c.q.f.d. $\Diamond\Diamond\Diamond$

L'algorithme 7 calcule la fonction π' en s'appuyant sur ces deux lemmes. Nous avons mis en évidence au moyen de commentaires le fait que π est implicitement calculé.

Algorithme 7: Un algorithme de calcul de la fonction π'

`CalculPiprime`(P,Σ) : une fonction
Input : P,Σ, un motif de m caractères sur l'alphabet Σ
Output : π', la fonction π' du motif
Data : i,j deux indices

$\pi'(0) \leftarrow -1$
$j \leftarrow 0$
// $\pi(1) = 0 = j$
for i **from** 1 **to** m **do**
  // $\pi(i) = j$
  if $P[i+1] \neq P[j+1]$ **then** $\pi'(i) = j$ **else** $\pi'(i) \leftarrow \pi'(j)$
  while $j \geq 0 \wedge P[j+1] \neq P[i+1]$ **do** $j \leftarrow \pi'(j)$
  $j \leftarrow j + 1$
  // $\pi(i+1) = j$
end
return π'

1.5 L'algorithme de Simon

L'algorithme de Simon repose sur une idée très simple : supprimer de l'automate les transitions qui conduisent à l'état 0. La figure 1.6 représente cet automate élagué pour le motif *ababaca*.

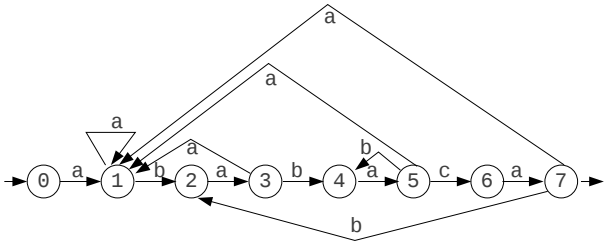


FIGURE 1.6: L'automate de Simon associé au motif *ababaca*

Nous allons dans un premier temps caractériser la réduction obtenue par cette suppression. Une transition $q \xrightarrow{a} r$ est dite *arrière* si $0 < r \leq q$. Son *amplitude* est définie par $q-r$. La proposition suivante est la clef de l'algorithme de Simon.

Proposition 5 *Soient deux transitions arrières* $q \xrightarrow{a} r$ *et* $q' \xrightarrow{a'} r'$. *Alors leur amplitude sont différentes, i.e.* $q' - r' \neq q - r$.

Preuve
Puisque $r > 0$, les mots $P[1,r]$ et $P[q-r+2,q]a$ sont identiques. Par conséquent, $P[r] = a$. Sans perte de généralité, nous supposons que $q \leq q'$.

Si $q = q'$ alors $a \neq a'$ puisqu'on a affaire à deux transitions issues du même état. Si les amplitudes sont les mêmes alors $r = r'$ et de même $P[r] = a'$ ce qui est contradictoire.

Si $q < q'$, alors $P[q+1] \neq a$ par définition des transitions arrières. D'autre part, les mots $P[1,r']$ et $P[q'-r'+2,q']a'$ sont identiques. Supposons que $q'-r' = q-r$. Observons que $r' > r$. Par conséquent, $a = P[r] = P[r+(q'-r'+1)] = P[q+1] \neq a$, de nouveau contradictoire. Ce cas est illustré par la figure 1.7.

c.q.f.d. ◊◊◊

Puisqu'une amplitude appartient à $\{0,1,\ldots,m-1\}$, il y a au plus m transitions arrières et au plus $2m$ transitions dans l'automate de Simon.

Expliquons maintenant comment calculer les transitions arrières par état croissant. Soit $q < m$ un état et a une lettre t.q. $a \neq P[q+1]$. On recherche le plus grand préfixe de P qui soit un suffixe de $P[1,q]a$. Nécessairement ce préfixe est un préfixe de $P[1,\pi'(q)+1]$. Si $\pi'(q) = -1$ il ne peut y avoir de transitions arrières. Sinon il y a deux cas à considérer :

- soit $a = P[\pi'[q]+1]$ et la transition arrière recherchée est $q \xrightarrow{a} \pi'(q)+1$.
- soit $a \neq P[\pi'[q]+1]$ et par conséquent on recherche le plus grand préfixe de P qui soit un suffixe de $P[1,\pi'(q)]a$. Par conséquent, il y a une transition arrière $q \xrightarrow{a} r$ ssi il y a une transition arrière $\pi'(q) \xrightarrow{a} r$.

Le cas de $q = m$ est identique puisqu'on recherche un préfixe de $P[1,\pi(m)+1]$ et par définition $\pi(m) = \pi'(m)$.

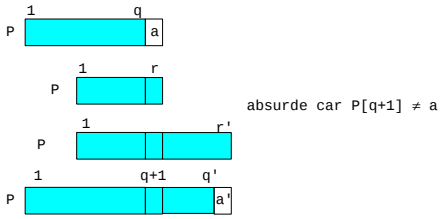


FIGURE 1.7: Illustration de la preuve de la proposition 1.5

L'algorithme consiste donc pour un état q :

— si $\pi'(q) > 0$, à ajouter les transitions arrières issues de $\pi'(q)$ excepté éventuellement celle étiquetée par $P[q+1]$ pour $q < m$,

— puis si $\pi'(q) \geq 0$, à ajouter la transition arrière $q \xrightarrow{P[\pi'(q)+1]} \pi'(q)+1$.

Les transitions arrières issues d'un état q sont gérées sous forme de liste et l'insertion dans la liste garantit que les états destinations sont de la forme $\pi'(q)+1, \pi'^{n_1}(q)+1, \ldots, \pi'^{n_i}(q)+1$ avec $1 < n_1 < \ldots < n_i$.

Pour simplifier, l'algorithme 8 de construction des transitions arrières de l'automate de Simon est donné en supposant la fonction π' déjà construite. En réalité, on peut construire la fonction et les transitions arrières simultanément.

Analysons la complexité de l'algorithme. Il est immédiat que son temps de calcul sur les listes est proportionnel à la somme des longueurs des listes manipulées. Or les listes manipulées n'ont jamais plus d'éléments que les listes finales. De plus les autres opérations sont associées aux deux boucles. On obtient encore une fois un algorithme en $O(m)$.

L'algorithme 9 (algorithme de Simon) implémente la reconnaissance des motifs dans un texte. La liste des transitions arrières est une suite de cellules composées d'un caractère (ch), d'un état destination (st) et d'un pointeur sur la cellule suivante ($suiv$).

Analysons sa complexité. Lorsqu'il examine un caractère qui ne prolonge pas le préfixe courant du motif, il doit parcourir la liste des transitions arrières de l'état courant q. Autrement dit, il essaie les états $q+1, \pi'(q)+1, \pi'^{n_1}(q)+1, \ldots, \pi'^{n_i}(q)+1$ avec $1 < n_1 < \ldots < n_i$ en comparant le caractère courant avec la dernière lettre du préfixe associé et en s'arrêtant au premier succès. Que fait en pareille situation l'algorithme KMP ? Il essaie les états $q+1, \pi'(q)+1, \pi'^2(q)+1, \ldots$ en comparant le caractère courant avec la dernière lettre du préfixe associé. Par conséquent, la complexité de l'algorithme de Simon est toujours au moins aussi bonne que celle de KMP.

Il est aussi possible d'adapter l'algorithme de Simon de telle sorte que lors de la lecture d'un caractère du texte le calcul du prochain état se fasse en $O(1)$ et non pas en $O(|\Sigma|)$ dans le pire cas. Ceci se fait au prix d'un espace mémoire en $O(m^2)$ mais en maintenant le temps du pré-traitement en $O(m)$. Cette adaptation repose sur une technique d'initialisation *paresseuse* de tableaux.

Algorithme 8: Un algorithme de calcul des transitions arrières

```
CalculTransAR(P,Σ,π') : un tableau de listes
Input : P,Σ, un motif de m caractères sur l'alphabet Σ
Input : π', la fonction π' du motif
Output : F, un tableau indexé par 1,...,m des listes
Data : i un indice, temp un pointeur sur une structure (ch,st,suiv)
Data : b un booléen
for i from 1 to m do
    F[i] ← NULL
    if π'(i) > 0 then
        Copy(F[π'(i)],F[i])
        if i < m then
            temp ← F[i] ; b ← true
            while temp ≠ NULL ∧ b do
                if temp→ch = P[i+1] then Extract(F[i],temp) ; b ← false
                else temp ← temp→suiv
            end
        end
    end
    if π'(i) ≥ 0 then
        New(temp) ; temp→ch ← P[π'(i)+1] ; temp→st ← π'(i)+1
        Insert(F[i],temp)
    end
end
return F
```

Algorithme 9: L'algorithme de Simon

```
Simon(T,P,F) : une liste
Input : T,P, un texte de n caractères et un motif de m ≤ n caractères
Input : F, le tableau des liste de transitions arrières
Output : L la liste des décalages d'occurrences de P dans T
Data : i,j deux indices, temp un pointeur
L ← NULL ; j ← 0
for i from 1 to n do
    if j = m ∨ P[j + 1] ≠ P[i] then
        if j = m then AjoutListe (L,i − m)
        if j > 0 then
            encore ← true ; temp ← F[j] ; j ← 0
            while encore ∧ temp ≠ NULL do
                if temp⇁ch = P[i] then
                    j ← temp⇁st ; encore ← false
                else
                    temp ← temp⇁suiv
                end
            end
        end
    else j ← j + 1
end
return L
```

Supposons le tableau des listes des transitions arrières F construite par l'algorithme 8. L'algorithme 10 calcule la représentation suivante de la matrice des transitions.

— Le tableau $Indir$ a ses C premières entrées significatives où $C \leq 2m$ est le nombre de transitions $i \xrightarrow{a} j$ avec $j > 0$. Une entrée (significative) $Indir[k]$ pour k correspondant à $i \xrightarrow{a} j$ a deux champs : $Indir[k].index = (i,a)$ et $Indir[k].val = j$;

— Le tableau $Delta$ indicé par $[0,m] \times \Sigma$ a uniquement les entrées (i,a) telles que $i \xrightarrow{a} j$ avec $j > 0$ initialisées. Dans ce cas, $Delta[i,a] = k$ avec $1 \leq k \leq C$ et $Indir[k].index = (i,a)$.

— Lors de la lecture d'un caractère a du texte alors que l'état courant est courant est i, on détermine l'état suivant de cette façon :

1. Si $Delta[i,a] \notin [1,C]$ ou $Indir[Delta[i,a]].index \neq (i,a)$ alors l'état suivant est 0. On laisse au lecteur le soin de vérifier que ce test détecte correctement les cellules non initialisées ;
2. Sinon l'état suivant est $Indir[Delta[i,a]].val$.

Algorithme 10: Un calcul en $O(m)$ de la matrice de transition

`Simon`(P,F) : une représentation de la matrice de transition
Input : P, un motif de m caractères
Input : F, le tableau des liste de transitions arrières

Output : C un compteur, $Delta$ un tableau entier indicé par $[0,m] \times \Sigma$
Output : $Indir$ un tableau indicé par $[1,2m]$ à deux champs,
Output : $index$ à valeurs dans $[0,m] \times \Sigma$ et val à valeurs dans $[1,m]$

Data : i un indice, $temp$ un pointeur

```
C ← 0
for i from 0 to m do
    if i < m then
        C ← C + 1 ; Delta[i,P[i + 1]] ← C
        Indir[C].index ← (i,P[i + 1]) ; Indir[C].val ← i + 1
    end
    temp ← F[i]
    while temp ≠ NULL do
        C ← C + 1 ; Delta[i,temp→ch] ← C
        Indir[C].index ← (i,temp→ch) ; Indir[C].val ← temp→st
        temp ← temp→suiv
    end
end
return C,Delta,Indir
```

1.6 L'algorithme de Boyer-Moore

Les algorithmes précédents s'attachent à améliorer la complexité au pire des cas (moins de $2n$ comparaisons de caractères). Si on change de point de vue et qu'on s'intéresse à la complexité dans le meilleur des cas, tous les algorithmes

précédents effectuent au moins n comparaisons. En effet chaque caractère du texte est comparé au moins une fois. L'algorithme de Boyer-Moore (en fait une famille d'algorithmes) vise à améliore cette complexité dans le meilleur des cas sans s'occuper de la complexité au pire des cas qui demeure alors en $O(nm)$.

Reprenons l'algorithme naïf. Il teste l'égalité du motif avec une partie du texte $T[i,i+m-1]$ en incrémentant i de 1, après chaque essai (fructueux ou infructueux). L'algorithme de Boyer-Moore cherche à obtenir un incrément supérieur en tenant compte du dernier essai. Il calcule pour cela deux minorants d'un incrément utile que nous appellerons d_1 et d_2.

Dans ces algorithmes le motif est testé contre une partie du texte de gauche à droite. Soit le texte *ababbcaabdb* et le motif *ababaca*. La première partie du texte testée est *ababbca*. Arrivé sur le troisième caractère en partant de la droite l'algorithme découvre une différence. S'il décale son texte de moins de m lettres, alors le premier caractère testé (le a) se retrouvera dans le texte testé et donc doit se trouver dans le motif *ailleurs qu'en dernière position*. Le décalage minimum correspond donc la différence entre m et la dernière occurrence d'un a dans le motif privé de la dernière lettre, ici 2. Dans le cas où un caractère n'apparaît pas dans le motif privé de la dernière lettre alors le décalage minimum est nécessairement égal à m. La fonction $d1 : \Sigma \to [1,m]$ spécifie donc le décalage minimum en fonction de la lettre lue la plus à droite du texte. Dans notre exemple : $d_1(a) = 2$, $d_1(b) = 3$ et $d_1(c) = 1$.

Algorithme 11: Première version de l'algorithme de Boyer-Moore

`CalculDecalage`(P) : une fonction
Input : P, un motif de $m \leq n$ caractères appartenant à Σ
Output : d_1, le décalage associé à chaque caractère
Data : a un caractère, i un indice

for $a \in \Sigma$ **do** $d_1[a] \leftarrow m$
for i **from** 1 **to** $m-1$ **do** $d_1[P[i]] \leftarrow m - i$
return d_1

`BoyerMoore`(T,P,d) : une liste
Input : T,P, un texte de n caractères et un motif de $m \leq n$ caractères
Input : d_1, le décalage associé à chaque caractère
Output : L la liste des décalages d'occurrences de P dans T
Data : i,j des indices

$L \leftarrow$ NULL ; $i \leftarrow m$
while $i \leq n$ **do**
  $j \leftarrow m$
  while $j > 0$ **and** $j \leq m$ **do**
    if $T[i-m+j] = P[j]$ **then** $j \leftarrow j-1$ **else** $j \leftarrow m+1$
  end
  if $j = 0$ **then** `AjoutListe`(L,i)
  $i \leftarrow i + d_1(T[i])$
end
return L

L'algorithme 11 met en oeuvre cette technique. Examinons sa complexité dans un cas favorable : un texte $(a^{m-1}b)^{n/m}$ et un motif a^m ($d_1(a) = 1$ et $d_1(b) = m$). Ici l'algorithme effectue n/m comparaisons ! Examinons sa complexité dans le pire des cas : un texte a^n et un motif a^m. Dans ce cas, l'algorithme effectue $m(n - m + 1)$ comparaisons comme l'algorithme naïf.

Nous développons maintenant un deuxième minorant du décalage qui tient compte du suffixe du motif qu'on a pu apparier avec le texte avant la découverte d'une différence. Soit $P[m - i + 1,m]$ avec $i \geq 0$ cette partie du motif. Il y a deux cas possibles :

— $i < m$ et il existe $i < j < m$ tel que $P[j - i + 1,j] = P[m - i + 1,m]$ apparaît plus tôt dans le texte avec $P[j-i] \neq P[m-i]$ (voir la figure 1.8). Soit j le plus grand indice vérifiant ces conditions, dans ce cas le décalage est $d_2(i) = m - j$.

— dans le cas contraire, soit $0 \leq j < m$ tel que $P[1,j]$ soit un suffixe propre de $P[m - i + 1,m]$ (voir la figure 1.9). Dans ce cas, $d_2(i) = m - j$ avec j le plus grand indice vérifiant ces conditions.

Lorsqu'aucun caractère n'a été reconnu, i.e. $i = 0$, on pose $d_2(0) = 1$. La fonction $d2 : [0,m] \rightarrow [1,m]$ spécifie donc le décalage minimum en fonction de la longueur du suffixe du motif reconnu en partant du caractère lu le plus à droite du texte.

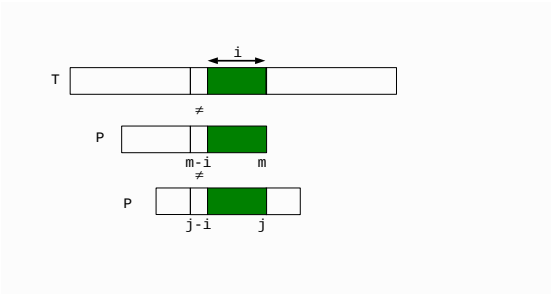


FIGURE 1.8: Situation associée à d_2^a

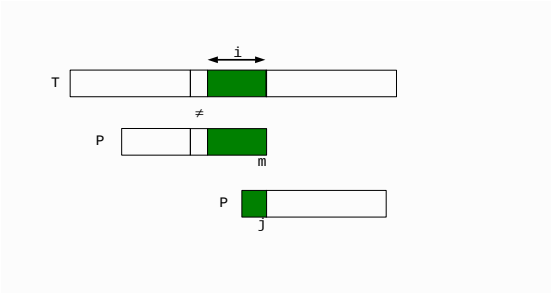


FIGURE 1.9: Situation associée à d_2^b

La prise en compte du deuxième minorant conduit à l'algorithme 12.

Il nous reste à calculer la fonction d_2. Nous allons traiter les deux cas vus précédemment en notant $d_2(i) = \min(d_2^a(i),d_2^b(i))$ avec $d_2^a(i)$ (resp. $d_2^b(i)$) la

Algorithme 12: Deuxième version de l'algorithme de Boyer-Moore

`BoyerMoore`(T,P,d_1,d_2) : une liste
Input : T,P, un texte de n caractères et un motif de $m \leq n$ caractères
Input : d_1, le décalage associé à chaque caractère
Input : d_2, le décalage associé à chaque position

Output : L la liste des décalages d'occurrences de P dans T

Data : i,j des indices, *encore* un booléen

$L \leftarrow$ NULL ; $i \leftarrow m$
while $i \leq n$ **do**
  $j \leftarrow m$; *encore* $\leftarrow$ true
  while $j > 0$ **and** *encore* **do**
    if $T[i-m+j] = P[j]$ **then** $j \leftarrow j-1$ **else** *encore* $\leftarrow$ false
  end
  if $j = 0$ **then** `AjoutListe`(L,i)
  $i \leftarrow i + \max(d_1(T[i]),d_2(m-j))$
end
return L

valeur du premier (resp. deuxième) cas. Le deuxième cas est le plus simple, il peut se réécrire :

$$d_2^b(i) = m - \max(j \mid j \leq \min(i,m-1) \wedge P[1,j] = P[m-j+1,m])$$

Les indices possibles j de l'équation précédente appartiennent à :

$$\{\pi(m),\pi^2(m),\ldots,\pi^h(m) = 0\}$$

et par conséquent le j recherché est $\max(\pi^k(m) \mid \pi^k(m) \leq \min(i,m-1))$. Ceci conduit à l'algorithme 13.

Algorithme 13: Calcul de d_2^b

`CalculDecDeuxB`(P,π) : une fonction
Input : P, un motif de $m \leq n$ caractères appartenant à Σ
Input : π, la fonction π du motif

Output : d_2^b, un décalage associé à chaque position

Data : i,j des indices

$d_2^b(m) \leftarrow m - \pi(m)$; $j \leftarrow m$
while $j > 0$ **do**
  for i **from** $\max(\pi(j),1)$ **to** $j-1$ **do** $d_2^b(i) \leftarrow m - \pi(j)$
  $j \leftarrow \pi(j)$
end
return d_2^b

Soit $j < m$, définissons :

$$\mathit{Suff}(j) = \max(i \mid 0 \leq i \leq j \wedge P[j-i+1,j] = P[m-i+1,m])$$

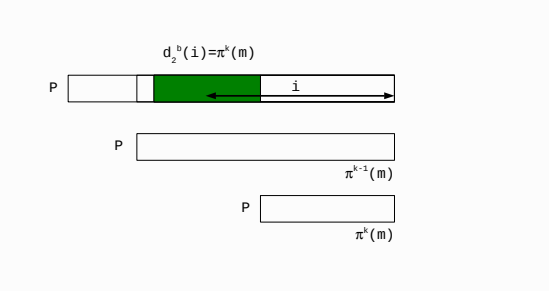


FIGURE 1.10: Calcul de d_2^b

La fonction *Suff* renvoie la taille d'un mot maximal qui se termine en j et qui est un suffixe du motif P. Par définition lorsque $d_2^a(i)$ est défini,

$$d_2^a(i) = m - \max(j \mid \mathit{Suff}(j) = i)$$

L'algorithme 14 synthétise le calcul de d_2.

Algorithme 14: Calcul de d_2

`CalculDecDeux`(π,d_2^b,*Suff*) : une fonction
Input : π, la fonction π du motif inversé
Input : d_2^b, la fonction d_2^b du motif
Input : *Suff*, la fonction *Suff* du motif

Output : d_2, le deuxième décalage associé à chaque position

Data : i un indice

$d_2(0) \leftarrow 1$
for i **from** 1 **to** m **do** $d_2(i) \leftarrow d_2^b(i)$
for i **from** 1 **to** $m-1$ **do**
  $d_2(\mathit{Suff}(i)) \leftarrow m - i$
end
return d_2

Il nous reste à calculer la fonction *Suff*. A priori, ce n'est pas difficile. Par contre, l'obtention d'un algorithme en $O(m)$ requiert de l'ingéniosité. Afin d'augmenter la lisibilité de notre présentation, nous allons plutôt calculer la fonction *Pref* définie par :

$$\mathit{Pref}(i) = \max(j \mid P[1,j] = P[i,i+j-1])$$

En passant par le mot miroir, la fonction *Suff* se déduit immédiatement de la fonction *Pref* (nous laissons au lecteur le soin d'établir cette relation).

L'algorithme 15 implémente le calcul de la fonction *Pref*. La fonction Scan prend en entrée deux indices du motif et trouve la taille du plus grand sous-mot commun aux deux suffixes démarrant dans ces indices. Cette fonction effectue un certain nombre de comparaisons de caractères *positives* et au plus une comparaison de caractères *négative*.

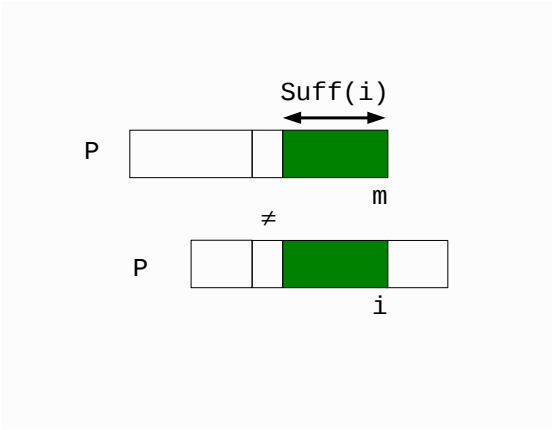


FIGURE 1.11: Calcul de d_2 à l'aide de *Suff*

L'algorithme calcule la fonction *Pref* par valeurs croissantes. A tout moment il maintient l'indice s qui maximise $i + \mathit{Pref}(i)$ parmi les indices i (déjà évalués) pour lesquels $\mathit{Pref}(i)$ est positif. Dans le cas où tous les $\mathit{Pref}(i)$ déjà évalués valent 0, s est fixé à 1 et $\mathit{Pref}(1)$ vaut 0. Ceci permet d'intégrer ce cas particulier au cas général (plus précisément au premier cas de la décomposition ci-dessous).

Nous dirons qu'un indice est *touché* lorsqu'il fait l'objet d'une comparaison positive avec un indice inférieur. Lors de l'analyse de cas, nous démontrerons par induction sur la boucle qu'un indice est touché au plus une fois. En supposant cette hypothèse vérifiée, par définition de s, les indices inférieurs à $s + \mathit{Pref}(s)$ ont été touchés au plus une fois et les indices supérieurs ou égaux n'ont pas été touchés.

Examinons les quatre cas de l'algorithme lorsqu'il évalue $\mathit{Pref}(i)$.

1. $s + \mathit{Pref}(s) \leq i$. Dans ce cas l'algorithme calcule $\mathit{Pref}(i)$ par une comparaison standard entre P et son décalage de $i - 1$. Les caractères en orange sont les caractères touchés par cette comparaison. Ils n'étaient pas touchés auparavant. Si $\mathit{Pref}(i) > 0$ alors s devient i. Donc l'invariant de boucle reste satisfait.
2. $s + \mathit{Pref}(s) > i$, autrement dit un préfixe de P démarrant en s couvre l'indice i. Appelons k l'indice de ce préfixe qui coïncide avec i. Le deuxième cas correspond à $k - 1 + \mathit{Pref}(k) < \mathit{Pref}(s)$. Ici aucune comparaison n'est nécessaire car le préfixe associé à k est celui associé à i. s est inchangé et l'invariant de boucle reste satisfait.
3. Le troisième cas correspond à $k - 1 + \mathit{Pref}(k) > \mathit{Pref}(s)$. Ici aucune comparaison n'est nécessaire car le préfixe associé à i est celui associé à k tronqué aux $s + \mathit{Pref}(s) - i - 1$ premiers caractères. s est inchangé et l'invariant de boucle reste satisfait.
4. Le quatrième cas correspond à $k - 1 + \mathit{Pref}(k) = \mathit{Pref}(s)$. Remarquons qu'alors $\mathit{Pref}(k) > 0$. Le préfixe démarrant en i est au moins aussi grand que celui démarrant en k. On effectue une comparaison entre le préfixe et son décalage par i mais en commençant à l'indice $s + \mathit{Pref}(s)$. Par conséquent, les caractères touchés par cette comparaison n'étaient pas

touchés auparavant. s devient i (observez qu'il est possible que $s+Pref(s)$ soit inchangé). L'invariant de boucle reste satisfait.

La figure 1.12 décrit les quatre cas de l'algorithme. Les caractères du texte en rouge n'ont pas été touchés. Les caractères du texte en orange n'étaient pas été touchés mais le sont par l'itération courante. La couleur bleu représente les correspondances entre les différentes portions du texte.

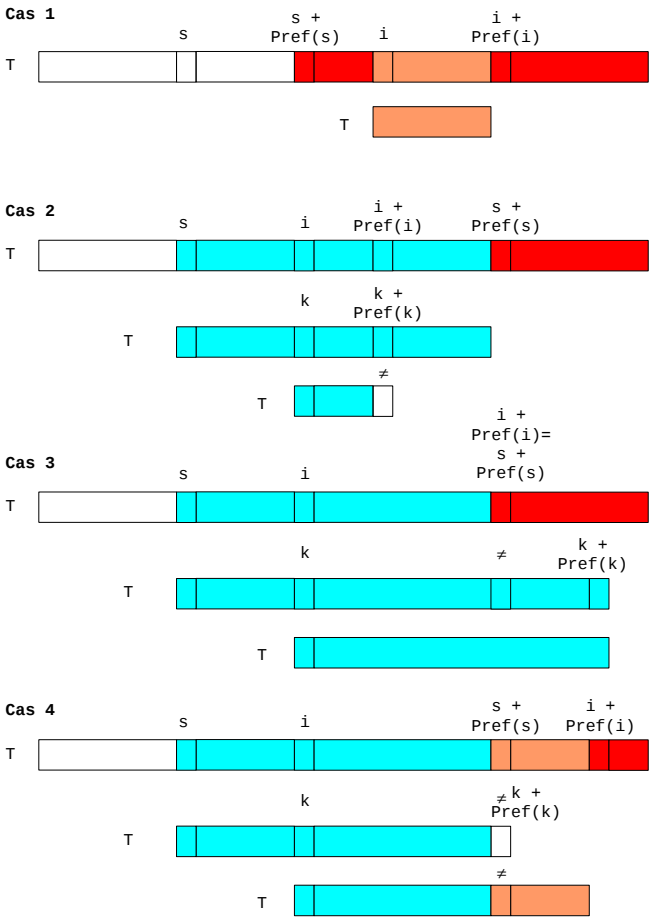


FIGURE 1.12: Les quatre cas du calcul de *Pref*

Nous avons démontré que l'algorithme est correct. Analysons sa complexité, un tour de boucle prend un temps en $O(1)$ excepté l'appel éventuel à Scan. Il y au plus une comparaison négative par appel donc au plus m comparaisons négatives. Le temps du reste de l'appel est proportionnel au nombre de comparaisons positives i.e. au nombre d'indices touchés. En cumulant sur tous les

appels et en prenant en compte le fait qu'un indice est touché au plus une fois, on obtient un temps en $O(m)$. Cet algorithme est bien linéaire.

Algorithme 15: Calcul de *Pref*

`CalculPref`(P) : une fonction
Input : P, un motif de m caractères appartenant à Σ
Output : *Pref*, la fonction du plus grand préfixe
Data : i,k,s,r,t des indices

`Scan`(x,y) : une longueur
Input : x,y, deux indices dans P
Output : z, la taille du plus grand sous-mot commun démarrant en x et y
$z \leftarrow 0$
while $x \leq m \wedge y \leq m$ **do**
  if $P[x] = P[y]$ **then** $x \leftarrow x + 1$; $y \leftarrow y + 1$; $z \leftarrow z + 1$
  else $x \leftarrow m + 1$
end
return z

Pref$(1) \leftarrow 0$; $s \leftarrow 1$
for i **from** 2 **to** m **do**
  $k \leftarrow i - s + 1$; $r \leftarrow s +$ *Pref*(s)
  if $r \leq i$ **then**
    Pref$(i) \leftarrow$ `Scan`$(i,1)$
    if Pref$(i) > 0$ **then** $s \leftarrow i$
  else if *Pref*$(k) + k - 1 \neq$ *Pref*(s) **then**
    Pref$(i) \leftarrow \min($*Pref*(k),$s +$ *Pref*$(s) - i)$
  else
    $t \leftarrow$ `Scan`$(r,r - i + 1)$
    Pref$(i) \leftarrow r - i + t$; $s \leftarrow i$
  end
end
return *Pref*

Chapitre 2

Polynômes et transformée de Fourier rapide (FFT)

Ouvrage recommandé : [Aho et al, chapitres 6,7,8]

2.1 Premiers algorithmes pour le produit de polynômes

Soit $\mathbb{A}$ un anneau. On suppose que :
— $\mathbb{A}$ est *commutatif*, i.e. $\forall x, y \in \mathbb{A}\ xy = yx$
— $\mathbb{A}$ est *unitaire*, i.e. $\exists 1\ \forall x\ 1x = x$
— $2 = 1 + 1$ est *inversible*, i.e. $\exists 2^{-1}\ 2^{-1}2 = 1$
On ne suppose pas que $\mathbb{A}$ est *intègre*, i.e $\forall x,y \in \mathbb{A}\ x \neq 0 \wedge y \neq 0 \Rightarrow xy \neq 0$. On note $\mathbb{A}[X]$ l'anneau des polynômes à une variable X sur $\mathbb{A}$. Si $P \in \mathbb{A}[X]$ alors $\mathbb{A}[X]/P$ désigne l'anneau quotient de $\mathbb{A}[X]$ dans lequel deux polynômes sont identifiés s'ils ont le même reste par la divison par P.

Soit deux polynômes $P = \sum_{i=0}^{n-1} p_i X^i$ et $Q = \sum_{i=0}^{n-1} q_i X^i$ de degré $n-1$. Le produit PQ est obtenu par l'algorithme 16.

Algorithme 16: Un algorithme naïf de produit de polynômes

`Produitnaif`(P,Q,n) : un polynôme
Input : P,Q, deux polynômes de degré $< n$
Output : R, le produit PQ
Data : i,j deux indices
for i **from** 0 **to** $2n-2$ **do** $R[i] \leftarrow 0$
for i **from** 0 **to** $n-1$ **do**
  for j **from** 0 **to** $n-1$ **do** $R[i+j] \leftarrow R[i+j] + P[i]Q[j]$
end
return R

Les deux boucles imbriquées de cet algorithme déterminent sa complexité en nombre d'opérations arithmétiques à savoir $O(n^2)$.

Une première amélioration peut être obtenue à partir d'une décomposition de la multiplication. Soit $k = \lceil n/2 \rceil$. P et Q se décomposent de manière unique en :

$$P = P^{(0)} + P^{(1)}X^k \quad Q = Q^{(0)} + Q^{(1)}X^k$$

avec $P^{(0)}$,$P^{(1)}$,$Q^{(0)}$,$Q^{(1)}$ des polynômes de degré $< k$. R s'exprime alors par :

$$R = P^{(0)}Q^{(0)} + (P^{(1)}Q^{(0)} + P^{(0)}Q^{(1)})X^k + P^{(1)}Q^{(1)}X^{2k}$$

ou de manière *a priori* équivalente :

$$R = P^{(0)}Q^{(0)} + ((P^{(0)}+P^{(1)})(Q^{(0)}+Q^{(1)}) - P^{(0)}Q^{(0)} - P^{(1)}Q^{(1)})X^k + P^{(1)}Q^{(1)}X^{2k}$$

En réalité la deuxième expression ne contient que 3 multiplications de polynômes de degré inférieur à k contre 4 multiplications pour la première expression. Cette transformation conduit à l'algorithme 17 (récursif) plus efficace dû à Karatsuba.

Algorithme 17: L'algorithme de Karatsuba

`Produitrec`(P,Q,n) : un polynôme
Input : P,Q, deux polynômes de degré $< n$
Output : R, le produit PQ
Data : i,j,k trois indices, $P^{(0)}$,$P^{(1)}$,$Q^{(0)}$,$Q^{(1)}$,$R^{(0)}$,$R^{(1)}$ des polynômes
Data : SP,SQ,SR des polynômes

if $n = 1$ **then** $R[0] \leftarrow P[0]Q[0]$; **return** R
$k \leftarrow \lceil n/2 \rceil$
for i **from** 0 **to** $k-1$ **do**
  $P^{(0)}[i] \leftarrow P[i]; SP[i] \leftarrow P[i]$
  $Q^{(0)}[i] \leftarrow Q[i]; SQ[i] \leftarrow Q[i]$
end
$P^{(1)}[k-1] \leftarrow 0; Q^{(1)}[k-1] \leftarrow 0$
for i **from** k **to** $n-1$ **do**
  $P^{(1)}[i-k] \leftarrow P[i]; SP[i-k] \leftarrow SP[i-k] + P[i]$
  $Q^{(1)}[i-k] \leftarrow Q[i]; SQ[i-k] \leftarrow SQ[i-k] + Q[i]$
end
$R^{(0)} \leftarrow$`Produitrec`($P^{(0)}$,$Q^{(0)}$,k)
$R^{(1)} \leftarrow$`Produitrec`($P^{(1)}$,$Q^{(1)}$,k)
$SR \leftarrow$`Produitrec`(SP,SQ,k)
for i **from** 0 **to** $k-1$ **do**
  $R[i] \leftarrow R^{(0)}[i]$
  $R[i+k] \leftarrow R^{(0)}[i+k] + SR[i] - R^{(0)}[i] - R^{(1)}[i]$
  $R[i+2k] \leftarrow R^{(1)}[i] + SR[i+k] - R^{(0)}[i+k] - R^{(1)}[i+k]$
  $R[i+3k] \leftarrow R^{(1)}[i+k]$
end
return R

Analysons sa complexité. Appelons $T(n)$ le nombre d'opérations arithmétiques de l'algorithme pour des polynômes de degré inférieur à n. Supposons d'abord que n est une puissance de 2. On a $T(1) = 1$ et :

$$T(n) = 4n + 3T(n/2) = 4n + (3/2)(4n) + (3/2)3T(n/4) = \ldots$$

$$= 4n \sum_{i=0}^{\log_2(n)-1} (3/2)^i + 3(3/2)^{\log_2 n - 1} = 4n((3/2)^{\log_2 n} - 1) + 2(3/2)^{\log_2 n} =$$

$$4n(n^{\log_2(3/2)} - 1) + 2n^{\log_2(3/2)} \leq 4n^{1+\log_2(3/2)} + n^{\log_2 3} = 5n^{\log_2 3}$$

On a utilisé ici l'identité $x^{\log y} = y^{\log x}$.

Dans le cas général, on se ramène à la première puissance de 2 supérieure ou égale à n, ce qui fournit un accroissement multiplicatif d'au plus $2^{\log_2 3} = 3$. Donc $T(n) = O(n^{1.59})$ ce qui améliore significativement l'algorithme naïf.

2.2 Produit de polynômes via la FFT

Imaginons une manière alternative d'effectuer le produit de deux polynômes P et Q de degré inférieur à n :

- On évalue les deux polynômes sur $2n$ points $x_1, \ldots, x_{2n}$.
- On effectue les produits des évaluations $P(x_i)Q(x_i)$.
- On pratique une interpolation de PQ à partir de ces évaluations en supposant que $\mathbb{A}$ est un corps ou plus précisément que $x_i - x_j$ est inversible pour tout $i \neq j$.

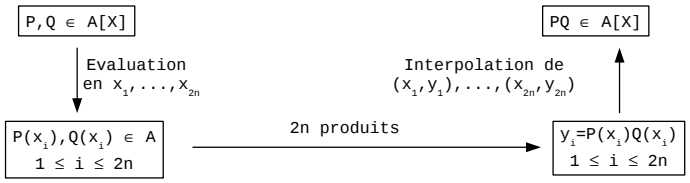


FIGURE 2.1: Produit par évaluation et interpolation

Alors que la deuxième étape s'effectue en $O(n)$ opérations, dans le cas général les deux autres étapes s'effectuent en $O(n^2)$ opérations. Cependant à l'aide d'un ensemble approprié de valeurs, cette complexité se réduit à $O(n \log(n))$ opérations comme nous le verrons dans la prochaine section. Pour l'instant, nous détaillons la première et la troisième étape dans le cas général.

Evaluation d'un polynôme

Soit un polynôme de degré inférieur à n, $P = \sum_{i=0}^{n-1} p_i X^i$. L'évaluation d'un polynôme en un point se fait en $2n-3$ multiplications et $n-1$ additions par la méthode naïve (algorithme 18). La complexité peut être réduite en s'appuyant sur la règle de Horner $P = p_0 + XP_1$ avec $P_1 = \sum_{i=1}^{n-1} p_i X^{i-1}$. L'algorithme 19 requiert ainsi $n-1$ multiplications et $n-1$ additions.

Interpolation par un polynôme

Algorithme 18: L'algorithme naïf d'évaluation de polynômes

`Evaluationnaive`(P,v) : une valeur dans $\mathbb{A}$
Input : P, un polynôme de degré $< n$, une valeur $v \in \mathbb{A}$

Output : s, l'évaluation $P(v)$

Data : i un indice, w une valeur intermédiaire

$s \leftarrow P[0]; w \leftarrow v$
for i **from** 1 **to** $n-2$ **do**
  // $w = v^i$ `et` $s = \sum_{k=0}^{i-1} P[k]v^k$
  $s \leftarrow s + P[i]w$
  $w \leftarrow wv$
end
$s \leftarrow s + P[n-1]w$
return s

Algorithme 19: L'algorithme de Horner d'évaluation de polynômes

`EvaluationHorner`(P,v) : une valeur dans $\mathbb{A}$
Input : P, un polynôme de degré $< n$, une valeur $v \in \mathbb{A}$

Output : s, l'évaluation $P(v)$

Data : i un indice

$s \leftarrow P[n-1]$
for i **from** $n-2$ **downto** 0 **do**
  // $s = \sum_{k=i+1}^{n-1} P[k]v^{k-i-1}$
  $s \leftarrow sv + P[i]$
end
return s

Supposons que nous disposions de n paires $(x_1,y_1),\ldots,(x_n,y_n)$ tels que tous les x_i soient différents. Il existe un unique polynôme P de degré inférieur à n qui vérifie pour tout i, $y_i = P(x_i)$. Ce polynôme s'écrit :

$$P = \sum_{i=1}^{n} y_i \frac{\prod_{j\neq i} X - x_j}{\prod_{j\neq i} x_i - x_j} = \sum_{i=1}^{n} y_i \frac{\prod_{j=1}^{n} X - x_j}{(\prod_{j\neq i} x_i - x_j)(X - x_i)}$$

L'algorithme 20 implémente cette interpolation. Le polynôme Q correspond à $\prod_{j=1}^{n} X - x_j$. A chaque tour i de la boucle externe le polynôme R est résultat de la division du polynôme Q par $X - x_i$.

Algorithme 20: Un algorithme d'interpolation de polynômes

`Interpolation`(X,Y) : un polynôme
Input : X, un tableau de n abcisses, Y, un tableau de n ordonnées
Output : P, un polynôme de degré $< n$
Data : i,j des indices, Q, un polynôme de degré n
Data : R, un polynôme de degré $< n$

$Q[0] \leftarrow -X[1]$; $Q[1] \leftarrow 1$; $s \leftarrow P[n-1]$
for i **from** 2 **to** n **do**
  $Q[i] \leftarrow 1$
  for j **from** $i-1$ **downto** 1 **do** $Q[j] \leftarrow -X[i]Q[j] + Q[j-1]$
  $Q[0] \leftarrow -X[i]Q[0]$
end
for i **from** 0 **to** $n-1$ **do** $P[i] \leftarrow 0$
for i **from** 1 **to** n **do**
  $c \leftarrow Y[i]$
  for j **from** 1 **to** n **do if** $j \neq i$ **then** $c \leftarrow \frac{c}{X[i]-X[j]}$
  $R[n-1] \leftarrow Q[n]$
  for j **from** $n-1$ **downto** 1 **do** $R[j-1] \leftarrow Q[j] + X[i]R[j]$
  for j **from** 0 **to** $n-1$ **do** $P[j] \leftarrow P[j] + cR[j]$
end
return P

2.2.1 Cas d'un anneau avec racines primitives de l'unité

Propriétés des racines primitives de l'unité

Afin de concevoir un algorithme de multiplication rapide de polynômes, nous étudions les racines primitives de l'unité.

Définition 3 *Soit $n \in \mathbb{N}^*$ et $\mathbb{A}$ un anneau, $\omega \in \mathbb{A}$ est une racine nième de l'unité si $\omega^n = 1$. C'est une racine primitive si de plus pour tout $1 \leq i < n$, $\omega^i - 1$ ne divise pas 0.*

Nous établissons quelques propriétés utiles des racines primitives.

Proposition 6 *Soit $n \in \mathbb{N}^*$, $\mathbb{A}$ un anneau et $\omega \in \mathbb{A}$ une racine nième primitive de l'unité.*

1. *ω^0,ω^1,...,ω^{n-1} sont des racines nièmes de l'unité toutes distinctes.*
2. *Si n est pair alors ω^2 est une racine $(n/2)$ième primitive de l'unité et par conséquent $(\omega^0)^2$,$(\omega^1)^2$,...,$(\omega^{n/2-1})^2$ sont des racines $(n/2)$ièmes de l'unité toutes distinctes. De plus $\omega^{n/2} = -1$.*
3. *Pour tout $1 \leq i < n$, on a $\sum_{j=0}^{n-1}(\omega^i)^j = 0$*

Preuve
Première affirmation
$(\omega^i)^n = (\omega^n)^i = 1$ ce qui établit que ω^i est une racine $n$ième de l'unité. Soient $0 \leq i < j < n$ tels que $\omega^j = \omega^i$ alors $(\omega^{j-i} - 1)\omega^i = 0$ donc $\omega^{j-i} - 1$ est un diviseur de 0 contrairement à l'hypothèse.

Deuxième affirmation
$(\omega^2)^{n/2} = \omega^n = 1$. Donc ω^2 est une racine $(n/2)$ième de l'unité. Soient $0 \leq i < j < n/2$ tels que $(\omega^2)^j = (\omega^2)^i$ alors $\omega^{2j} = \omega^{2i}$ contrairement à la première affirmation.
$(\omega^{n/2} - 1)(\omega^{n/2} + 1) = \omega^n - 1 = 0$. Puisque $\omega^{n/2} - 1$ n'est pas un diviseur de 0, on a $\omega^{n/2} + 1 = 0$.

Troisième affirmation
$(\sum_{j=0}^{n-1}(\omega^i)^j)(\omega^i - 1) = (\omega^i)^n - 1 = 0$. Puisque $\omega^i - 1$ n'est pas un diviseur de 0, la conclusion s'en suit.

c.q.f.d. ◊◊◊

La transformée de Fourier rapide

Supposons que nous disposions d'une racine $n$ième primitive de l'unité ω et que nous ayons (pré-)calculé en $n - 2$ multiplications ω^0,ω^1,...,ω^{n-1}. Si nous désirons calculer l'image des racines par P l'application itérée de l'un quelconque des algorithmes précédents nous obtenons un nombre d'opérations arithmétiques en $O(n^2)$.

La transformée de Fourier rapide est un algorithme qui permet d'obtenir ces images en $O(n\log(n))$ pour $n = 2^k$ une puissance de 2. L'algorithme procède d'abord à une décomposition de $P(X) = P^{(0)}(X^2) + XP^{(1)}(X^2)$ avec :
- $P^{(0)} = \sum_{i=0}^{n/2-1} p_{2i}X^i$
- $P^{(1)} = \sum_{i=0}^{n/2-1} p_{2i+1}X^i$

A l'aide de cette décomposition et en notant que $\omega^{k+n/2} = \omega^{n/2}\omega^k = -\omega^k$, on obtient :
- $\forall 0 \leq k < n/2$ $P^{(0)}(\omega^{2k}) = P^{(0)}(\omega^{2(n/2+k)}) = \sum_{i=0}^{n/2-1} p_{2i}(\omega^{2k})^i$
- $\forall 0 \leq k < n/2$ $P^{(1)}(\omega^{2k}) = P^{(1)}(\omega^{2(n/2+k)}) = \sum_{i=0}^{n/2-1} p_{2i+1}(\omega^{2k})^i$
- $\forall 0 \leq k < n/2$
  $P(\omega^k) = P^{(0)}(\omega^{2k}) + \omega^k P^{(1)}(\omega^{2k})$ et $P(\omega^{n/2+k}) = P^{(0)}(\omega^{2k}) - \omega^k P^{(1)}(\omega^{2k})$

Ces trois familles d'équations se traduisent en un algorithme 21 récursif, évaluer les polynômes $P^{(0)}$ et $P^{(1)}$ pour ω^0,ω^2,...,ω^{2n-2}, les racines $(n/2)$ièmes de l'unité, et appliquer la troisième famille d'équations afin d'obtenir l'évaluation désirée.

Analysons la complexité de cet algorithme. Soit $T(n)$ le nombre d'opérations arithmétiques. Si $n = 1$ alors $T(1) = 0$ sinon il y a $3n/2$ opérations pour un appel. Par conséquent :

Algorithme 21: L'algorithme FFT

`FFT`(P,T,n) : un tableau
Input : P, un polynôme de degré $< n$
Input : T, un tableau des n racines $n$ièmes de l'unité
Input : $\omega^0, \ldots, \omega^{n-1}$ avec ω une racine primitive
Input : n une puissance de 2
Output : F, le tableau des valeurs $P(\omega^0), \ldots, P(\omega^{n-1})$
Data : i un indice, $P^{(0)}$,$P^{(1)}$ des polynômes de degré $< n/2$
Data : v une valeur, T',$F^{(0)}$,$F^{(1)}$ des tableaux de dimension $n/2$
if $n = 1$ **then** $F[0] \leftarrow P[0]$; **return** F
for i **from** 0 **to** $n/2 - 1$ **do**
  $P^{(0)}[i] \leftarrow P[2i]$
  $P^{(1)}[i] \leftarrow P[2i + 1]$
  $T'[i] \leftarrow T[2i]$
end
$F^{(0)} \leftarrow$ `FFT`($P^{(0)}$,T',$n/2$) ; $F^{(1)} \leftarrow$ `FFT`($P^{(1)}$,T',$n/2$)
for i **from** 0 **to** $n/2 - 1$ **do**
  $v \leftarrow T[i]F^{(1)}[i]$
  $F[i] \leftarrow F^{(0)}[i] + v$
  $F[n/2 + i] \leftarrow F^{(0)}[i] - v$
end
return F

$T(n) = 3n/2 + 2T(n/2) = 2(3n/2) + 2^2T(n/4) = \ldots$
$= \log(n)(3n/2) + 2^{\log(n)}T(1) = \log(n)(3n/2)$

Produit de polynômes avec la FFT dans $\mathbb{A}[X]/(X^n - 1)$

Expliquons maintenant comment multiplier deux polynômes P et Q de $\mathbb{A}[X]/(X^n - 1)$.

1. Evaluer par FFT les valeurs $P(\omega^0), \ldots, P(\omega^{n-1})$ et $Q(\omega^0), \ldots, Q(\omega^{n-1})$.
2. Calculer $P(\omega^0)Q(\omega^0), \ldots, P(\omega^{n-1})Q(\omega^{n-1})$. Notons $P \cdot Q$ le polynôme produit de P et Q dans $\mathbb{A}[X]/(X^n-1)$ et PQ le polynôme produit de P et Q dans $\mathbb{A}[X]$. Dans $\mathbb{A}[X]$, $P \cdot Q = PQ + (X^n - 1)R$ pour un certain R. Par conséquent $P \cdot Q(\omega^i) = P(\omega^i)Q(\omega^i) + (X^n - 1)(\omega^i)R(\omega^i) = P(\omega^i)Q(\omega^i)$.
3. Interpoler le polynôme $P \cdot Q$ à partir de son évaluation sur les racines $n$ièmes de l'unité. Nous détaillons maintenant ce dernier point.

Soit un polynôme $P[X] = \sum_{i=0}^{n-1} p_i X^i$. L'application qui associe au vecteur des coefficients de P le vecteur $(P(1), P(\omega), \ldots, P(\omega^{n-1}))$ est une application linéaire dont la matrice $M_{\omega,n}$ est décrite ci-dessous.

$$(P(1), P(\omega), \ldots, P(\omega^{n-1})) = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ 1 & \omega & \ldots & \omega^{n-1} \\ \vdots & & & \vdots \\ 1 & \omega^{n-1} & \ldots & \omega^{(n-1)^2} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{pmatrix}$$

Cette application est inversible et la matrice de son inverse a une forme très particulière.

Proposition 7 *$M_{\omega,n}M_{\omega^{-1},n} = nId_n$ où Id_n la matrice identité $n \times n$. Par conséquent si n est une puissance de 2 alors $M_{\omega,n}^{-1} = (1/n)M_{\omega^{-1},n}$*

Preuve
Soit $\alpha_{i,j}$ le coefficient (i,j) de la matrice $M_{\omega,n}M_{\omega^{-1},n}$. On a :
$\alpha_{i,j} = \sum_{k=0}^{n-1}(\omega^{i-j})^k$. Si $i = j$ alors $\alpha_{i,j} = n$ sinon d'après le point 3 de la proposition 6 $\alpha_{i,j} = 0$.

c.q.f.d. ◊◊◊

Cette proposition nous permet d'extrapoler les coefficients d'un polynôme à partir de son évaluation sur les racines de l'unité et ceci par une transformée de Fourier rapide dont la racine primitive est ω^{-1}.

L'analyse de la complexité de l'algorithme 22 est très simple. En comptant les trois appels à FFT et les opérations auxiliaires, on obtient $(9/2)n\log(n) + 3n + 1$ opérations arithmétiques.

Produit de polynômes avec la FFT dans $\mathbb{A}[X]$

Rappelons que cette multiplication a lieu dans l'anneau $\mathbb{A}[X]/(X^n - 1)$. Afin de faire la multiplication dans $A[X]$ de deux polynômes de degré $< n$, il nous suffit de la faire dans $\mathbb{A}[X]/(X^{2n} - 1)$ car alors les deux produits coïncident (voir la figure 2.2).

Algorithme 22: Le produit des polynômes par la FFT

`ProduitFFT`(P,Q,ω,n) : un tableau
Input : P,Q, deux polynômes de degré $< n$
Input : ω une racine primitive $n$ième de l'unité
Input : n une puissance de 2

Output : PQ, le produit des polynômes P,Q dans l'anneau $\mathbb{A}[X]/(X^n - 1)$

Input : T,T', deux tableaux de n racines $n$ièmes de l'unité
Data : i un indice, v une valeur
Data : $PQint$,FP,FQ des tableaux de dimension n

$T[0] \leftarrow 1$; $T'[0] \leftarrow 1$
for i **from** 1 **to** $n-1$ **do** $T[i] \leftarrow \omega T[i-1]$; $T'[n-i] \leftarrow T[i]$
$FP \leftarrow$ `FFT`(P,T,n) ; $FQ \leftarrow$ `FFT`(Q,T,n)
for i **from** 0 **to** $n-1$ **do** $FPQ[i] \leftarrow FP[i]FQ[i]$
$PQint \leftarrow$ `FFT`(FPQ,T',n) ; $v \leftarrow 1/n$
for i **from** 0 **to** $n-1$ **do** $PQ[i] \leftarrow v \cdot PQint[i]$
return PQ

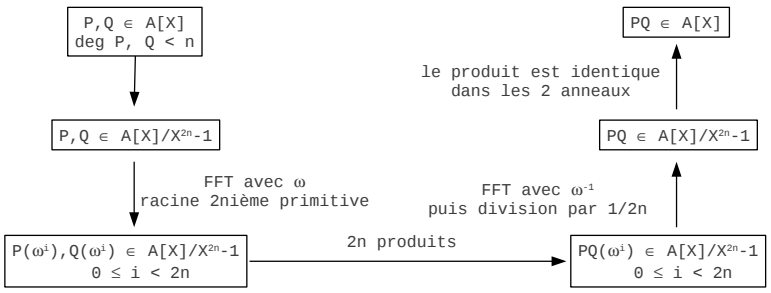


FIGURE 2.2: Principe du produit rapide de polynômes

Si les coefficients de nos polynômes sont représentés par des nombres flottants, on peut considérer que $\mathbb{A} = \mathbb{R}$ et dans ce cas on plonge $\mathbb{R}$ dans $\mathbb{C}$ avec comme racine primitive $e^{2i\pi/n}$.

Produit de polynômes avec la FFT dans $\mathbb{A}[X]/(X^n + 1)$

Soient $P = \sum_{i<n} p_i X^i$ et $Q = \sum_{j<n} q_j X^j$ deux polynômes à coefficients dans A un anneau, de degré inférieur à n. On note $S = \sum_{k<n} s_k X^k$ le produit de P et Q dans $\mathbb{A}[X]/(X^n + 1)$.

Observons que $S = \sum_{k=0}^{n-1}(\sum_{i+j=k} p_i q_j)X^k + \sum_{k=0}^{n-1}(\sum_{i+j=n+k} p_i q_j)X^{n+k}$ mod $X^n + 1$. En remplaçant X^n par -1, on obtient :

$$\forall k \; s_k = \sum_{i+j=k} p_i q_j - \sum_{i+j=n+k} p_i q_j \text{ avec } 0 \leq i,j < n$$

On suppose que n est une puissance de 2. On dispose de θ une racine $2n$-ième primitive de l'unité et on pose $\omega = \theta^2$ (une racine n-ième primitive de l'unité). On définit l'application linéaire $F_\omega : A^n \mapsto A^n$ par :

$$F_\omega(p_0, \ldots, p_{n-1}) = (P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1}))$$

On a vu que F_ω est inversible d'inverse $n^{-1}F_{\omega^{-1}}$.

Soit $H = \sum_{i<n} h_i X^i$ un polynôme, on note $\hat{H} = \sum_{i<n} \hat{h}_i X^i$, le polynôme défini par $\forall i\ \hat{h}_i = \theta^i h_i$. Calculons $\hat{P}(\omega^i)\hat{Q}(\omega^i)$.

$$\hat{P}(\omega^i)\hat{Q}(\omega^i) = \sum_{k=0}^{n-1}(\sum_{i+j=k} p_i q_j)\theta^k(\omega^i)^k + \sum_{k=0}^{n-1}(\sum_{i+j=n+k} p_i q_j)\theta^{n+k}(\omega^i)^{n+k} =$$

$$\sum_{k=0}^{n-1}(\sum_{i+j=k} p_i q_j)\theta^k(\omega^i)^k - \sum_{k=0}^{n-1}(\sum_{i+j=n+k} p_i q_j)\theta^k(\omega^i)^k = \hat{S}(\omega^i)$$

Par conséquent :

$$F_\omega(\hat{s}_0, \ldots, \hat{s}_{n-1}) = (\hat{P}(\omega^0)\hat{Q}(\omega^0), \hat{P}(\omega^1)\hat{Q}(\omega^1), \ldots, \hat{P}(\omega^{n-1})\hat{Q}(\omega^{n-1}))$$

Ce qui équivaut à :

$$P *_{\widehat{\mathbb{A}[X]/X^n+1}} Q = \hat{P} *_{\mathbb{A}[X]/X^n-1} \hat{Q}$$

A partir de cette dernière égalité, on en déduit un algorithme (similaire à celui dans $\mathbb{A}[X]/(X^n - 1)$) qui effectue le produit de deux polynômes dans $\mathbb{A}[X]/(X^n + 1)$ en $O(n\log(n))$ opérations arithmétiques lorsqu'on dispose de θ :

- Construire les polynômes $\hat{P}$ et $\hat{Q}$ (en $O(n)$)
- Evaluer les polynômes $\hat{P}$ et $\hat{Q}$ sur les racines $\omega^0, \ldots, \omega^{n-1}$ (par la FFT en $O(n\log(n))$)
- Effectuer le produit de ces évaluations (en $O(n)$)
- Obtenir les coefficients de $\hat{S}$ (par la FFT « inversée » en $O(n\log(n))$)
- En déduire les coefficients de S (en $O(n)$)

La figure 2.3 décrit cette méthode.

2.2.2 Cas d'un anneau sans racine primitive de l'unité

Cependant dans le cas où les coefficients sont des entiers ou plus généralement des rationnels sous forme d'un couple (numérateur, dénominateur) et que l'on désire effectuer un calcul exact, le plongement dans $\mathbb{C}$ ne convient pas. L'objet de cette section est de proposer un calcul rapide même dans un anneau sans racine primitive (comme par exemple $\mathbb{Q}$).

Lemme 4 *Soit $n = 2^k$ avec $k \in \mathbb{N}^*$, alors $\theta = X$ est une racine $2n$-ième primitive de l'unité dans $\mathbb{A}[X]/(X^n + 1)$. Par conséquent X^2 est une racine n-ième primitive de l'unité dans cet anneau.*

Preuve
$\theta^n = -1$. Donc θ est une racine $2n$-ième de l'unité. Montrons que θ est une racine primitive. Il suffit d'établir que pour tout $1 \leq t < 2n$, $\theta^t - 1$ n'est pas un diviseur de 0.

Algorithme 23: L'algorithme FFT dans un autre anneau

`FFTbis`(P,n,m,s) : un tableau
Input : P, un polynôme (en Y) de degré $< m$ à coefficients dans $\mathbb{A}[X]/(X^n+1)$, i.e. un tableau de dimension $m \times n$
Input : n une puissance de 2, $m \leq n$ une puissance de 2, s un signe
// Si $s=1$ alors $\omega = X^{2n/m}$ est la racine $m$ième primitive
// de l'unité dans $\mathbb{A}[X]/(X^n+1)$ à utiliser
// Si $s=-1$ alors $\omega = -X^{n-2n/m}$ est la racine à utiliser
Output : F, le tableau des valeurs $P(\omega^0),\ldots,P(\omega^{m-1})$
Data : i,j des indices, $P^{(0)}$,$P^{(1)}$ des polynômes en Y de degré $< m/2$
Data : v un polynôme en X
Data : $F^{(0)}$,$F^{(1)}$ des tableaux de dimension $m/2 \times n$
if $m=1$ **then** $F[0] \leftarrow P[0]$; **return** F
for i **from** 0 **to** $m/2-1$ **do** $P^{(0)}[i] \leftarrow P[2i]$; $P^{(1)}[i] \leftarrow P[2i+1]$
$F^{(0)} \leftarrow$ `FFTbis`$(P^{(0)},n,m/2)$; $F^{(1)} \leftarrow$ `FFTbis`$(P^{(1)},n,m/2)$
for i **from** 0 **to** $m/2-1$ **do**
  // Calcul de $P(\omega^i)$ et de $P(\omega^{\frac{m}{2}+i})$
  for j **from** 0 **to** $n-1$ **do** $v[j] \leftarrow 0$
  // Calcul de $v = \omega^i P^{(1)}(\omega^{2i})$
  for j **from** 0 **to** $n-1$ **do**
    // Multiplier par $X^{2ni/m}$
    // revient à décaler les coefficients ...
    if $s=1$ **then** $k \leftarrow (2ni/m)+j$
    else $k \leftarrow n-(2ni/m)+j$
    // ... en tenant compte du fait que $X^n=-1$
    // qui modifie le décalage et change le signe
    // car k est compris entre 0 et $2n-1$
    if $k<n$ **then** $v[k] \leftarrow sF^{(1)}[i][j]$
    else $v[k-n] \leftarrow -sF^{(1)}[i][j]$
  end
  $F[i] \leftarrow F^{(0)}[i]+v$; $F[m/2+i] \leftarrow F^{(0)}[i]-v$
end
return F

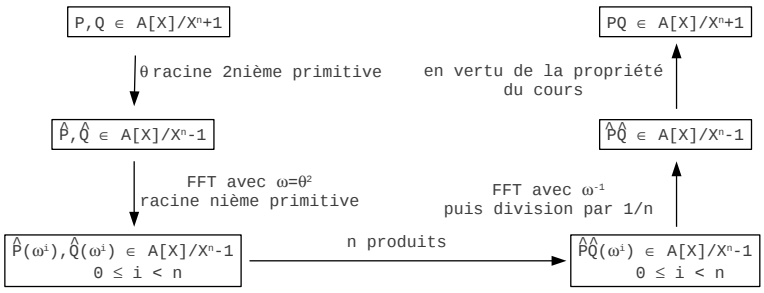


FIGURE 2.3: Principe du produit rapide de polynômes dans $\mathbb{A}[X]/X^n + 1$

Nous effectuons une remarque préalable. Soient p,q deux entiers quelconques avec q positif, $\theta^{pq} - 1 = (\theta^p - 1)(1 + \theta^p + \theta^{2p} + \ldots + \theta^{(q-1)p})$. Par conséquent, si $\theta^{pq} - 1$ n'est pas un diviseur de 0 alors $\theta^p - 1$ ne l'est pas non plus. Ce résultat est aussi pour q est négatif car en vertu de l'égalité $\theta^{pq} - 1 = \theta^{pq}(1 - \theta^{-pq})$, $\theta^{pq} - 1$ est un diviseur de zéro si et seulement si $\theta^{-pq} - 1$ est un diviseur de zéro. D'autre part, $\theta^n - 1 = -2$ qui est inversible donc pas un diviseur de 0. Posons $g = pgcd(t,2n)$, g s'écrit $2^{k'}$ avec $k' \leq k$, donc g divise n. D'après la remarque préalable, $\theta^g - 1$ n'est pas un diviseur de 0. D'après le théorème de Bezout, il existe u et v tels que $tu + 2nv = g$. Donc :

$$\theta^t - 1 \mid \theta^{tu} - 1 = \theta^{tu}\theta^{2nv} - 1 = \theta^g - 1$$

Par conséquent, $\theta^t - 1$ n'est pas un diviseur de 0.

c.q.f.d. ◊◊◊

Produit de polynômes dans un anneau sans racine primitive

L'algorithme 23 effectue l'évaluation de P un polynôme en Y dans l'anneau $\mathbb{A}[X]/(X^n + 1)$ appliqué aux racines $m$ièmes de l'unité (avec $m \leq n$) :

— $1, X^{2n/m}, X^{4n/m}, \ldots, X^{2(m-1)n/m}$ (mod $X^n + 1$)

— $-X^n (= 1), -X^{n-2n/m}, -X^{n-4n/m}, \ldots, -X^{n-2(m-1)n/m}$ (mod $X^n + 1$)

Observons que seules les $m/2$ premières racines de l'unité sont utilisées dans l'algorithme. Par conséquent, les degrés des polynômes correspondants ci-dessus sont tous compris entre 0 et $n - 1$ (avant application du modulo). Cependant la multiplication par les polynômes renvoyés par les appels récursifs peuvent conduire à des degrés compris entre entre 0 et $2n - 2$ ce qui explique le test $k < n$.

Les opérations dans $\mathbb{A}[X]/(X^n + 1)$ sont soient des additions, soient des soustractions soient des multiplications par une racine qui consistent à effectuer un décalage des valeurs modulo n avec éventuellement un changement de signe. Toutes ces opérations sont en $O(n)$. Par conséquent, la complexité de l'algorithme est en $O(nm \log(m))$.

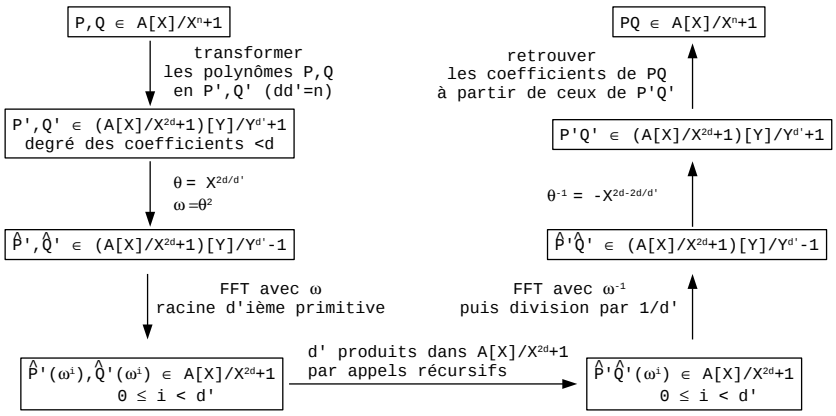


FIGURE 2.4: Principe du produit de polynômes dans un anneau sans racines primitives

Produit rapide de polynômes dans un anneau sans racine primitive

L'algorithme 24 utilise l'algorithme précédent afin de faire un produit rapide de deux polynômes P,Q de degré $< n$ dans $\mathbb{A}[X]/(X^n+1)$ (voir la figure 2.4). Pour l'effectuer dans $\mathbb{A}[X]$, il suffit de doubler n et ainsi d'éviter le modulo.

Il décompose d'abord n en deux puissances de 2 proches ou égales à $\sqrt{n}$, d et d' telles que $n = dd'$.

— Si $\log(n)$ est pair alors $d = d' = \sqrt{n}$.
— Si $\log(n)$ est impair alors $d = \sqrt{2n}$ et $d' = \sqrt{\frac{n}{2}}$.

Puis il calcule des polynômes de degré $< d'$, $P'[X][Y]$ et $Q'[X][Y]$ appartenant à $(\mathbb{A}[X]/X^{2d}+1)[Y]/X^{d'}+1$ vérifiant :

$$P[X] = P'[X][X^d] \text{ et } Q[X] = Q'[X][X^d]$$

Illustrons cette transformation sur $n = 8$ et $d = 4$ et $d' = 2$.
Si $P = p_0 + p_1X + \ldots + p_7X^7$ alors :

$$P' = (p_0 + p_1X + p_2X^2 + p_3X^3) + (p_4 + p_5X + p_6X^2 + p_7X^3)Y$$

De manière plus systématique, $p'_i = \sum_{j<d} p_{id+j}X^j$

Il applique la transformée de Fourier rapide dans l'anneau des polynômes $(\mathbb{A}[X]/X^{2d}+1)[Y]/(Y^{d'}+1)$ pour calculer le produit $P'Q'$ avec pour racine $2d'$ième primitive $X^{2d/d'}$ puis $-X^{2d-2d/d'}$.

Il reste à retrouver les coefficients de PQ à partir de ceux de $P'Q'$.

Appelons c_u le coefficient de PQ de degré u : $c_u = \sum_{s+t=u} p_sq_t - \sum_{s+t=n+u} p_sq_t$.
Posons $u = ad+b$ avec $0 \leq a < d'$ et $0 \leq b < d$. Cette somme peut aussi s'écrire (avec $0 \leq h,l < d$) :
Cas $a > 0$

$$c_u = \sum_{i+j=a} \sum_{h+l=b} p_{id+h}q_{jd+l} - \sum_{i+j=d'+a} \sum_{h+l=b} p_{id+h}q_{jd+l}$$

$$+ \sum_{i+j=a-1} \sum_{h+l=d+b} p_{id+h}q_{jd+l} - \sum_{i+j=d'+a-1} \sum_{h+l=d+b} p_{id+h}q_{jd+l}$$

Cas $a = 0$

$$c_u = \sum_{h+l=b} p_h q_l - \sum_{i+j=d'} \sum_{h+l=b} p_{id+h}q_{jd+l}$$

$$- \sum_{i+j=d'-1} \sum_{h+l=d+b} p_{id+h}q_{jd+l}$$

Examinons maintenant le coefficient c'_a du polynôme $P'Q'$ (en vertu de notre choix, le modulo $X^{2d} + 1$ ne s'applique donc pas) :

$$c'_a = \sum_{i+j=a} p'_i q'_j - \sum_{i+j=d'+a} p'_i q'_j =$$

$$\sum_{i+j=a} (\sum_{h<d} p_{id+h}X^h)(\sum_{l<d} q_{jd+l}X^l) - \sum_{i+j=d'+a} (\sum_{h<d} p_{id+h}X^h)(\sum_{l<d} q_{jd+l}X^l)$$

Son coefficient en X^b, noté $c'_{a,b}$ est égal à :

$$\sum_{i+j=a} \sum_{h+l=b} p_{id+h}q_{jd+l} - \sum_{i+j=d'+a} \sum_{h+l=b} p_{id+h}q_{jd+l}$$

Le coefficient en X^{d+b} du coefficient c'_{a-1} $(a > 0)$ est égal à :

$$\sum_{i+j=a-1} \sum_{h+l=d+b} p_{id+h}q_{jd+l} - \sum_{i+j=d'+a-1} \sum_{h+l=d+b} p_{id+h}q_{jd+l}$$

Par conséquent lorsque $a > 0$, on obtient $c_u = c'_{a,b} + c'_{a-1,d+b}$.
Examinons $c'_{d'-1}$:

$$c'_{d'-1} = \sum_{i+j=d'-1} (\sum_{h<d} p_{id+h}X^h)(\sum_{l<d} q_{jd+l}X^l) - \sum_{i+j=2d'-1} (\sum_{h<d} p_{id+h}X^h)(\sum_{l<d} q_{jd+l}X^l)$$

$$= \sum_{i+j=d'-1} (\sum_{h<d} p_{id+h}X^h)(\sum_{l<d} q_{jd+l}X^l)$$

car puisque $i \leq d' - 1$ et $j \leq d' - 1$, il n'est pas possible que $i + j = 2d' - 1$.
D'où :

$$c'_{d'-1,d+b} = \sum_{i+j=d'-1} \sum_{h+l=d+b} p_{id+h}q_{jd+l}$$

Par conséquent lorsque $a = 0$, on obtient $c_u = c'_{0,b} - c'_{d'-1,d+b}$.

Les trois appels à la FFT se font par l'algorithme 23 tandis que le produit des évaluations de P' et de Q' se fait par un appel récursif.

Analysons la complexité de l'algorithme. Les appels à la FFT coûtent :

$$O(2dd' \log(d')) = O(n \log(n))$$

Les opérations internes à l'algorithme se font en $O(n)$. Il faut y ajouter l'appel récursif. Nous notons $T(n)$ le temps d'exécution. D'après nos observations pour un certain c :

Algorithme 24: L'algorithme de Schönhage et Strassen

`ProduitSS`(P,Q,n) : un polynôme
Input : P,Q, deux polynômes de degré $< n$ à coefficients dans $\mathbb{A}$
Input : n une puissance de 2
Output : PQ, le produit de P et Q dans $\mathbb{A}[X]/(X^n+1)$
Data : P',Q',P'',Q'' des polynômes de degré $< d'$ à coefficients de degré $< d$ dans $(\mathbb{A}[X]/(X^{2d}+1))[Y]/(Y^{d'}+1)$
Data : PQ',PQ'' des polynômes intermédiaires
Data : i,j,ij,u,v des indices
Data : FP,FQ,FPQ des tableaux de d' polynômes en X dans $\mathbb{A}[X]/(X^{2d}+1)$
Data : d,d' des puissances de 2 telles que $dd' = n$, v une valeur

if $n \leq 8$ **then return** `ProduitNaif`(P,Q,n)
if $\log(n) \mod 2 = 0$ **then** $d \leftarrow \sqrt{n}$; $d' \leftarrow \sqrt{n}$; $p \leftarrow 2$
else $d \leftarrow \sqrt{2n}$; $d' \leftarrow \sqrt{n/2}$; $p \leftarrow 4$
// θ `la racine (2`d'`)ème dans` $\mathbb{A}[X]/(X^{2d}+1)$ `est le monôme` X^p
$ij \leftarrow 0$; $P' \leftarrow 0$; $Q' \leftarrow 0$; $P'' \leftarrow 0$; $Q'' \leftarrow 0$
for i **from** 0 **to** $d'-1$ **do**
  for j **from** 0 **to** $d-1$ **do**
    $P'[i][j] \leftarrow P[ij]$; $Q'[i][j] \leftarrow Q[ij]$
    // P'', Q'' `sont les polynômes` $\hat{P}'$,$\hat{Q}'$
    $u \leftarrow j + pi$ **div** $2d$
    $v \leftarrow j + pi$ **mod** $2d$
    if u **mod** $2 = 0$ **then** $P''[i][v] \leftarrow P'[i][j]$; $Q''[i][v] \leftarrow Q'[i][j]$
    else $P''[i][v] \leftarrow -P'[i][j]$; $Q''[i][v] \leftarrow -Q'[i][j]$
    $ij \leftarrow ij + 1$
  end
end
$FP \leftarrow$ `FFTbis`(P'',$2d$,d',1) ; $FQ \leftarrow$ `FFTbis`(Q'',$2d$,d',1)
for i **from** 0 **to** $d'-1$ **do** $FPQ[i] \leftarrow$ `ProduitSS`($FP[i]$,$FQ[i]$,$2d$)
$PQ'' \leftarrow$ `FFTbis`(FPQ,$2d$,d', -1)
// $\frac{1}{d'}PQ''$ `est le polynôme` $\hat{P}'\hat{Q}'$ `dans` $(\mathbb{A}[X]/(X^{2d}+1))[Y]/Y^{d'}-1$
for i **from** 0 **to** $d'-1$ **do**
  for j **from** 0 **to** $2d-1$ **do**
    $u \leftarrow j - pi$ **div** $2d$
    $v \leftarrow j - pi$ **mod** $2d$
    if u **mod** $2 = 0$ **then** $PQ'[i][v] \leftarrow PQ''[i][j]/d'$
    else $PQ'[i][v] \leftarrow -PQ''[i][j]/d'$
  end
end
// PQ' `est le polynôme` $P'Q'$ `dans` $(\mathbb{A}[X]/(X^{2d}+1))[Y]/Y^{d'}+1$
for i **from** 0 **to** $d'-1$ **do**
  for j **from** 0 **to** $d-1$ **do**
    $u \leftarrow di + j$
    if $i = 0$ **then** $PQ[u] \leftarrow PQ'[0][j] - PQ'[d'-1][j+d]$
    else $PQ[u] \leftarrow PQ'[i][j] + PQ'[i-1][j+d]$
  end
end
return PQ

— Si $\log(n)$ est pair $T(n) \leq cn\log(n) + \sqrt{n}T(2\sqrt{n})$
— Si $\log(n)$ est impair $T(n) \leq cn\log(n) + \sqrt{n/2}T(2\sqrt{2n})$

Nous affirmons que $T(n) \leq e \cdot n\log(n)\log(\log(n))$ pour tout $n \geq n_0$ pour un certain n_0 et un certain e.

Démontrons-le par induction. Supposons d'abord que $\log(n)$ est pair. Alors :
$T(n) \leq cn\log(n) + \sqrt{n}T(2\sqrt{n})$
$T(n) \leq cn\log(n) + e\sqrt{n}(2\sqrt{n}\log(2\sqrt{n})\log(\log(2\sqrt{n})))$
$T(n) \leq cn\log(n) + 2en\log(2\sqrt{n})\log(\log(2\sqrt{n}))$
$T(n) \leq cn\log(n) + en(\log(n)+2)\log(\log(2\sqrt{n}))$
$T(n) \leq cn\log(n) + en(\log(n)+2)\log((1/2)\log(n)+1)$
$T(n) \leq cn\log(n) + en(\log(n)+2)(\log((1/2)\log(n))+1/2)$
(avec n_0 suffisamment grand)
$T(n) \leq cn\log(n) + en(\log(n)+2)(\log(\log(n))-1/2)$
$T(n) \leq en\log(n)\log(\log(n)) + (c-e/2)n\log(n) + 2\log(\log(n))$
$T(n) \leq en\log(n)\log(\log(n))$
(avec $e > 2c$ et n_0 suffisamment grand)

Supposons maintenant que $\log(n)$ est impair. Alors :
$T(n) \leq cn\log(n) + \sqrt{n/2}T(2\sqrt{2n})$
$T(n) \leq cn\log(n) + e\sqrt{n/2}(2\sqrt{2n}\log(2\sqrt{2n})\log(\log(2\sqrt{2n})))$
$T(n) \leq cn\log(n) + 2en\log(2\sqrt{2n})\log(\log(2\sqrt{2n}))$
$T(n) \leq cn\log(n) + en(\log(n)+3)\log(\log(2\sqrt{2n}))$
$T(n) \leq cn\log(n) + en(\log(n)+3)\log((1/2)\log(n)+3/2)$
$T(n) \leq cn\log(n) + en(\log(n)+3)(\log((1/2)\log(n))+1/2)$
(avec n_0 suffisamment grand)
$T(n) \leq cn\log(n) + en(\log(n)+3)(\log(\log(n))-1/2)$
$T(n) \leq en\log(n)\log(\log(n)) + (c-e/2)n\log(n) + 3\log(\log(n))$
$T(n) \leq en\log(n)\log(\log(n))$
(avec $e > 2c$ et n_0 suffisamment grand)

2.3 Inversion de série formelle

Une série formelle est notée $P = \sum_{i\in\mathbb{N}} p_i X^i$. Une série ayant un nombre infini de coefficients, ceux-ci ne sont pas stockés explicitement mais calculés à la demande à l'aide par exemple d'une relation de récurrence entre les coefficients ou d'une expression symbolique. On fait l'hypothèse que ce temps de calcul est en $O(1)$ et par conséquent à un certain niveau d'abstraction, on peut imaginer disposer de ces coefficients.

Les séries formelles sont dotées du produit de la manière suivante :
soit $P = \sum_{i\in\mathbb{N}} p_i X^i$ et $Q = \sum_{j\in\mathbb{N}} q_j X^j$ alors $PQ = \sum_{k\in\mathbb{N}}(\sum_{i+j=k} p_i q_j)X^k$.

Une autre opération fréquente est l'inversion de série.

Lemme 5 *Une série formelle $P = \sum_{i\in\mathbb{N}} p_i X^i$ est inversible ssi p_0 est inversible dans $\mathbb{A}$. Dans ce cas, son inverse $Q = \sum_{i\in\mathbb{N}} q_i X^i$ est définie inductivement par :*

$$q_0 = p_0^{-1} \text{ et } q_n = -q_0 \sum_{i<n} q_i p_{n-i}$$

Preuve
L'inverse $Q = \sum_{i\in\mathbb{N}} q_i X^i$ doit vérifier $\forall i > 0 \ \sum_{i+j=k} p_i q_j = 0$ et $p_0 q_0 = 1$. La preuve en découle immédiatement.

c.q.f.d. ◊◊◊

Observations. Le coefficient q_n de l'inverse ne dépend que des valeurs $p_0, \ldots, p_n$. Le calcul des n premiers coefficients de l'inverse en utilisant la relation de récurrence se fait en $O(n^2)$.

Nous allons concevoir un algorithme qui permet d'obtenir les n premiers coefficients de manière plus efficace. En vertu de l'observation précédente, on peut supposer que P est un polynôme de degré inférieur à n.

Soit $n = 2^k$. Décomposons Q l'inverse de P de la façon suivante :
— $Q = Q_0 + Q_1 + Q_2$
— $Q_0 = \sum_{i<n/2} q_i X^i$, $Q_1 = \sum_{n/2\leq i<n} q_i X^i$ et $Q_2 = \sum_{i\geq n} q_i X^i$

Nous allons nous servir deux fois de l'équation : $P(Q_0 + Q_1 + Q_2) = 1$

Tout d'abord :
$1 - PQ_0 = PQ_1 + PQ_2$
En multipliant par Q_0 :
$(1 - PQ_0)Q_0 = PQ_0Q_1 + PQ_0Q_2$
puis en utilisant à nouveau l'équation :
$(1 - PQ_0)Q_0 = (1 - PQ_1 - PQ_2)Q_1 + PQ_0Q_2$
$(1 - PQ_0)Q_0 = Q_1 - P(Q_1)^2 - PQ_1Q_2 + PQ_0Q_2$
Puisque $(Q_1)^2$ et Q_2 sont des séries dont tous les termes inférieurs à n sont nuls et Q_0 est de degré strictement inférieur à $n/2$, on obtient :

$$Q_1 = -P(Q_0)^2 \textit{ tronqué aux coefficients d'indice compris entre } n/2 \textit{ et } n-1$$

C'est la base de l'algorithme 25. Analysons sa complexité : le coût d'un appel récursif est dominé par les appels à la fonction `Produit`. Le coût d'un produit est borné par $cn\log(n)(\log\log(n))$ dans le cas des algorithmes les plus performants. On obtient donc en notant $T(n)$ le temps d'exécution de l'algorithme :
$T(n)$
$\leq cn\log(n)(\log\log(n)) + c(n/2)\log(n/2)(\log\log(n/2)) + \ldots$
$\leq 2cn\log(n)(\log\log(n))$
Autrement dit, le coût d'une inversion de série est du même ordre que celui du produit.

2.4 Division de polynômes

Soient deux polynômes P de degré n et D de degré $m \leq n$ dont le coefficient de plus haut degré est inversible. On désire effectuer la division euclidienne de P par D :

$$P = QD + R$$

avec $deg(R) < m$

La division standard requiert $O((n - m + 1)m)$ opérations soit un coût quadratique dans le pire des cas. Nous allons maintenant améliorer cet ordre de

Algorithme 25: Un algorithme d'inversion de séries

`Inverse`(P,n) : un polynôme
Input : P, un polynôme inversible de degré $< n$ une puissance de 2
Output : Q, l'inverse de P
Data : i un indice, Q_0 un polynôme de degré $< n/2$
Data : Q_a un polynôme de degré $< n$, Q_b un polynôme de degré $< 2n$
if $n = 1$ **then** $Q[1] \leftarrow 1/P[1]$; **return** Q
$Q_0 \leftarrow$ `Inverse`(P,$n/2$)
$Q_a \leftarrow$ `Produit`(Q_0,Q_0,$n/2$)
$Q_b \leftarrow$ `Produit`(P,Q_a,n)
for i **from** 0 **to** $n/2 - 1$ **do** $Q[i] \leftarrow Q_0[i]$
for i **from** $n/2$ **to** $n - 1$ **do** $Q[i] \leftarrow -Q_b[i]$
return Q

grandeur par des manipulations algébriques. On divise l'équation précédente par D ce qui nous donne :

$$\frac{P}{D} = Q + \frac{R}{D}$$

On applique cette égalité à $X = 1/T$ où T est une nouvelle variable, puis on multiplie l'équation par T^{n-m} ce qui nous donne :

$$\frac{T^n P(1/T)}{T^m D(1/T)} = T^{n-m} Q(1/T) + \frac{T^n R(1/T)}{T^m D(1/T)}$$

Dans cette équation tous les numérateurs et les dénominateurs sont des polynômes et $T^m D(1/T)$ a pour terme constant, un coefficient inversible. Enfin $T^n R(1/T)$ est divisible par $n - m + 1$. Par conséquent :

$T^{n-m} Q(1/T) = T^n P(1/T)(T^m D(1/T))^{-1}$ *(tronqués aux $n-m$ premiers coefficients)*

A partir de $T^{n-m}Q(1/T)$, Q s'obtient par inversion des coefficients puis $R = P - QD$. C'est qu'effectue l'algorithme 26. La complexité de cet algorithme est dominée par le dernier produit en $O(n \log(n) \log(\log(n)))$.

Algorithme 26: Un algorithme de division de polynômes

`Division`(P,D,m,n) : deux polynômes
`// tous les polynômes sont stockés`
`// comme des polynômes de degré` $< n$
Input : P, un polynôme inversible de degré $< n$
Input : D, un polynôme inversible de degré $< m \leq n$
Output : Q,R, les polynômes quotient et reste de la division P/D
Data : i un indice
Data : D',D'',Q',QD des polynômes de degré $< n$
for i **from** 0 **to** $n-1$ **do** $P'[i] \leftarrow P[n-i]$
for i **from** 0 **to** $m-1$ **do** $D'[i] \leftarrow D[n-i]$
$D'' \leftarrow$ `Inverse`(D',$n-m$)
$Q' \leftarrow$ `Produit`(P',D'',$n-m$)
for i **from** 0 **to** $n-m-1$ **do** $Q[i] \leftarrow Q'[n-i]$
$QD \leftarrow$ `Produit`(Q,D,n)
for i **from** 0 **to** $m-2$ **do** $R[i] \leftarrow P[i] - QD[i]$
return Q,R

Chapitre 3

Compression de données

Ouvrage recommandé : [Nelson et Gailly]

3.1 Eléments de la théorie de l'information

L'entropie

Essayons de définir la valeur, $v(p)$ d'une information de la réalisation d'un événement futur en fonction de sa probabilité d'occurrence p. Un événement certain n'apporte pas d'information donc $v(1) = 0$. Plus généralement, plus un événement est rare, plus grande est sa valeur. Par conséquent, v doit être décroissante et par conséquent positive. Puisque cette valeur est relative, nous affectons arbitrairement la valeur 1, à un événement qui a une chance sur deux de se réaliser ($v(1/2) = 1$).

Quelle peut être la valeur $v(pp')$ d'un événement de probabilité d'occurrence pp' ? Cet événement est équivalent à la conjonction de deux événements indépendants ayant respectivement une probabilité d'occurrence p et p'. Par additivité, puisque l'occurrence d'un des deux événements ne modifie pas la probabilité de l'autre, on conclut que cette valeur est $v(p) + v(p')$. Par conséquent, $v(pp') = v(p) + v(p')$.

Posons $f(x) = v(2^{-x})$ pour $x \geq 0$, on a immédiatement $f(x+y) = f(x)+f(y)$ et $f(1) = 1$. Un raisonnement arithmétique entraîne $f(\frac{p}{q}) = \frac{p}{q}$ et par croissance de f, on conclut que $f(x) = x$ et donc $v(x) = -\log(x)$ où la base du logarithme est 2.

Soit maintenant une variable aléatoire (v.a.) discrète X à valeurs dans l'ensemble $\{e_1, \ldots, e_n\}$ et telle que $Pr(X = e_i) = p_i$. Quelle peut être la valeur d'un tirage de X sans en connaître le résultat ? Cette valeur est la valeur de chacun des événements pondérée par sa probabilité d'occurrence. Autrement dit $H(X)$, qu'on appelle l'entropie, est définie par :

$$H(X) = -\sum_{i=1}^{n} p_i \log(p_i)$$

Observons qu'on peut admettre dans cette formule des p_i nuls puisque : $\lim_{x \to 0} x \log(x) = 0$. Notons aussi que :

$$H(X) = -E(\log(p(X)))$$

où la fonction p associe à un événement sa probabilité d'occurrence.

Changer d'échelle revient à changer la base du logarithme. Aussi on note $H_D(X) = -\sum_{i=1}^{n} p_i \log_D(p_i)$ avec $D > 1$. On notera aussi $H(p)$ l'entropie d'une v.a. de distribution p.

L'entropie conditionnelle

Nous introduisons à présent un ensemble de notions qui complètent le concept de base. Soit un couple de v.a. (X,Y) et soit $p_{i,j} = Pr(X = x_i \wedge Y = y_i)$. On note $p_{Xi} = \sum_j p_{i,j}$ la probabilité marginale de X. L'entropie conditionnelle mesure la quantité d'information apportée par Y, connaissant X.

Définition 4 *Soit (X,Y) un couple de v.a.. L'entropie conditionnelle $H(Y|X)$ est définie par :*

$$H(Y|X) = \sum_i p_{Xi} H(Y|X = x_i) = -\sum_i p_{Xi} \sum_j \frac{p_{i,j}}{p_{Xi}} \log\left(\frac{p_{i,j}}{p_{Xi}}\right) = -\sum_{i,j} p_{i,j} \log\left(\frac{p_{i,j}}{p_{Xi}}\right)$$

La proposition suivante justifie l'intuition qui conduit à l'entropie conditionnelle.

Proposition 8 *Soit (X,Y) un couple de v.a.. Alors :*

$$H(X,Y) = H(X) + H(Y|X)$$

Preuve

$H(X,Y) = -\sum_{i,j} p_{i,j} \log(p_{i,j})$

$= -\sum_{i,j} p_{i,j} \log\left(\frac{p_{i,j}}{p_{Xi}}\right) - \sum_{i,j} p_{i,j} \log(p_{Xi}) = H(Y|X) + H(X)$

c.q.f.d. ◊◊◊

L'entropie relative

L'entropie relative vise à établir une distance entre deux distributions sur le même ensemble d'événements.

Définition 5 *Soient p et p' deux distributions sur le même ensemble d'événements $\mathcal{X}$. L'entropie relative $D(p||p')$ de p' par rapport à p est définie par :*

$$D(p||p') = \sum_{x \in \mathcal{X}} p(x) \log(\frac{p(x)}{p'(x)}) = E(\log(\frac{p(X)}{p'(X)}))$$

où X est une v.a. de distribution p.

Observons que $D(p||p') \neq D(p'||p)$ et d'autre part que $D(p||p')$ peut être égal à $+\infty$ dans le cas d'un événement x tel que $p(x) \neq 0$ et $p'(x) = 0$. Cependant la proposition suivante souligne l'intérêt de l'entropie relative.

Proposition 9 *Soient p et p' deux distributions sur le même ensemble d'événements $\mathcal{X}$. Alors : $D(p||p') \geq 0$ avec l'égalité ssi $\forall x \in \mathcal{X}$ $p(x) = p'(x)$*

Preuve
On suppose sans perte de généralité que $\forall x \in \mathcal{X}\ p(x) > 0$.
$D(p||p') = -\sum_{x\in\mathcal{X}} p(x)\log(\frac{p'(x)}{p(x)}) \geq -\log(\sum_{x\in\mathcal{X}} p(x)\frac{p'(x)}{p(x)})$
par convexité de $-\log$.
$= -\log(\sum_{x\in\mathcal{X}} p'(x)) = 0$
Puisque $-\log$ est strictement convexe, l'égalité n'a lieu que si $\forall x \in \mathcal{X}\ p(x) = p'(x)$.

c.q.f.d. ◊◊◊

L'information mutuelle

L'information mutuelle caractérise l'information associée à l'interaction de deux v.a.

Définition 6 *Soient X et Y deux v.a. de distribution jointe $p(x,y)$ et de distributions marginales $p_X(x)$ et $p_Y(y)$. L'information mutuelle de X et Y, notée $I(X;Y)$ est définie par :*

$$I(X;Y) = D(p||p_X p_Y)$$

La proposition suivante justifie l'intuition associée à l'information mutuelle. Elle démontre aussi que le conditionnement diminue l'entropie d'une v.a.

Proposition 10 *Soient X et Y deux v.a., alors :*

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

Par conséquent, $H(Y) \geq H(Y|X)$ avec égalité ssi X et Y sont indépendantes.

Preuve
$I(X;Y) = \sum_{x,y} p(x,y)\log(\frac{p(x,y)}{p_X(x)p_Y(y)}) = \sum_{x,y} p(x,y)\log(\frac{p(x|y)}{p_X(x)})$
$= -\sum_{x,y} p(x,y)\log(p_X(x)) + \sum_{x,y} p(x,y)\log(p(x|y))$
$= -\sum_{x} p_X(x)\log(p_X(x)) + \sum_{x,y} p(x,y)\log(p(x|y))$
$= H(X) - H(X|Y)$

c.q.f.d. ◊◊◊

Maximisation de l'entropie

Etant donné un espace d'événements, il est intéressant de connaître la distribution qui maximise l'entropie.

Proposition 11 *Soient X une v.a. à valeurs dans $\mathcal{X}$ (fini). Alors $H(X)$ est maximal ssi X est équirépartie sur $\mathcal{X}$.*

Preuve
Notons $\mathcal{X} = \{x_1, \ldots, x_n\}$ et $p_i = Pr(X = x_i)$. Soit Y une v.a. équirépartie sur $\mathcal{X}$ de distribution u. Alors :
$0 \leq D(p||u) = \sum_{i\leq n} p_i \log(p_i n) = \sum_{i\leq n} p_i \log(p_i) + \log(n) = H(Y) - H(X)$

c.q.f.d. ◊◊◊

L'entropie d'une v.a. dans un espace dénombrable est définie de la même façon que dans le cas fini. Cette entropie peut être infinie. Cependant si on fixe la moyenne de la v.a., on obtient le résultat suivant.

Proposition 12 *Soient X une v.a. (avec $p_i = Pr(X = i)$) à valeurs dans $\mathbb{N}$ de moyenne $\mu = \frac{p}{1-p}$ avec $0 \leq p < 1$. Alors $H(X)$ est maximal ssi X a une distribution géométrique $p_i = p^i(1-p)$. Dans ce cas :*

$$H(X) = (\mu + 1)\log(\mu + 1) - \mu \log(\mu)$$

Preuve
Soit Y de distribution géométrique $g_i = p^i(1-p)$.
$H(X) = -\sum_{i\in\mathbb{N}} p_i \log(p_i) = -\sum_{i\in\mathbb{N}} p_i \log(\frac{p_i}{g_i} g_i) = -D(p||g) - \sum_{i\in\mathbb{N}} p_i \log(g_i)$
$\leq -\sum_{i\in\mathbb{N}} p_i \log(p^i(1-p))$
$= -\sum_{i\in\mathbb{N}} ip_i \log(p) - \log(1-p)$
$= -\sum_{i\in\mathbb{N}} ig_i \log(p) - \log(1-p)$
$= -\sum_{i\in\mathbb{N}} g_i \log(p^i(1-p))$
$= H(Y)$
$H(Y) = -\sum_{i\in\mathbb{N}} p^i(1-p)\log(p^i(1-p))$
$= -\sum_{i\in\mathbb{N}} ip^i(1-p)\log(p) - \sum_{i\in\mathbb{N}} p^i(1-p)\log(1-p))$
$= -\mu\log(p) - \log(1-p) = -\mu(\log(\mu) - \log(\mu+1)) + \log(\mu+1)$
(car $p = \frac{\mu}{\mu+1}$)
$= (\mu+1)\log(\mu+1) - \mu\log(\mu)$

c.q.f.d. ◊◊◊

Entropie et flux d'informations

Les flux d'informations que nous considérons dans la suite sont définis par une distribution d'occurrences des caractères de l'alphabet d'entrée. Il s'agit évidemment d'une simplification car l'occurrence d'un caractère dépend généralement des caractères qui sont déjà apparus (par exemple une consonne a plus de chances d'apparaître après une voyelle qu'après une consonne).

Proposition 13 *Soient $X_1, \ldots, X_n, \ldots$ une suite de v.a. mutuellement indépendantes et identiquement distribuées sur un espace d'événements fini $\mathcal{X}$ (de distribution p), alors :*

$$\lim_{n\to\infty} (-1/n)\log(p(X_1)\ldots p(X_n)) = H(p) \text{ presque sûrement}$$

Preuve
C'est une conséquence immédiate de la loi forte des grands nombres (théorème 5) appliquée aux v.a. $-\log(p(X_1)), \ldots, -\log(p(X_n)), \ldots$ puisque ces variables ont une moyenne finie.

c.q.f.d. ◊◊◊

3.2 Codes

Un code est associé à une v.a. X. L'objectif visé est le codage efficace d'information (l'occurrence des événements) en fonction de leur probabilité d'occurrence.

Définition 7 *Soit Σ un alphabet et X une v.a. à valeurs dans $\mathcal{X}$. Un code C est une fonction de $\mathcal{X}$ dans Σ^*.*

L'efficacité d'un code C se définit naturellement par le nombre moyen de caractères utilisés.

Définition 8 *La longueur moyenne d'un code C pour X une v.a. à valeurs dans $\mathcal{X}$ dont la distribution est donnée par $p(x)$ avec $x \in \mathcal{X}$ est définie par :*

$$L(C) = \sum_{x \in \mathcal{X}} p(x)|C(x)|$$

Un code doit être décodé ! Formaliser cette propriété n'est pas difficile. On étend d'abord la fonction C à une fonction $\mathcal{X}^*$ dans Σ^* par concaténation des codes.

Définition 9 *Un code C est* uniquement déchiffrable *si la fonction (étendue) C est injective.*

Les codes standard qui associent par exemple un mot différent de longueur *fixe* à chaque événement sont uniquement déchiffrables. Ne tenant pas compte de la v.a. X, ils ne sont généralement pas optimaux. Parmi les codes uniquement déchiffrables de longueur variable, les codes préfixes sont particulièrement intéressants.

Définition 10 *Un code C est un code préfixe si $\forall x \neq y \in \mathcal{X}$, $C(x)$ n'est pas un préfixe de $C(y)$ et vice versa.*

La démonstration qu'un code préfixe est uniquement déchiffrable est immédiate par récurrence sur la longueur de la séquence à coder et laissée au lecteur.

Avant de proposer des codes efficaces, établissons d'abord un résultat fondamental qui démontre que la recherche de codes efficaces peut se limiter aux codes préfixes.

Théorème 1 (de McMillan) *Les longueurs des mots d'un code C uniquement déchiffrable vérifient l'inégalité suivante, dite inégalité de Kraft :*

$$\sum_{x \in \mathcal{X}} |\Sigma|^{-|C(x)|} \leq 1$$

De plus, étant donné un code C qui vérifie cette inégalité, il est possible de construire un code préfixe C' avec les mêmes longueurs de mots.

Preuve
Dans la suite, on note $D = |\Sigma|$ et $l(\sigma) = |C(\sigma)|$ pour $\sigma \in \mathcal{X}^*$. Soit k un entier non nul, on observe que le nombre de séquences $\sigma \in \mathcal{X}^k$ dont le code est de taille n, est inférieur ou égal à D^n.
$\left(\sum_{x \in \mathcal{X}} D^{-l(x)}\right)^k = \sum_{x_1 \in \mathcal{X}} \cdots \sum_{x_k \in \mathcal{X}} D^{-l(x_1)} \ldots D^{-l(x_k)} = \sum_{\sigma \in \mathcal{X}^k} D^{-l(\sigma)}$
Posons maintenant $a(m) \leq D^m$ le nombre de séquences $\sigma \in \mathcal{X}^k$ telles que $l(\sigma) = m$ et $l_{max} = \max(l(x) \mid x \in \mathcal{X})$. On a par définition :
$\sum_{\sigma \in \mathcal{X}^k} D^{-l(\sigma)} = \sum_{m=1}^{kl_{max}} a(m) D^{-m} \leq \sum_{m=1}^{kl_{max}} D^m D^{-m} = kl_{max}$

Par conséquent :
$\forall k > 0 \ \sum_{x \in \mathcal{X}} D^{-l(x)} \leq (k l_{max})^{1/k}$

En passant à la limite lorsque k tend vers l'infini, on obtient :

$$\sum_{x \in \mathcal{X}} D^{-l(x)} \leq 1$$

Rangeons les événements $x_1, \ldots, x_n$ par $l(x_i)$ croissants. Sans perte de généralité, posons $\Sigma = \{0,1,\ldots,D-1\}$. On définit les mots du code préfixe par induction :

— $C(x_1) = 0^{l(x_1)}$.

— Soit $C(x_i) = \alpha_1 \ldots \alpha_{l(x_i)}$ et soit $j \leq l(x_i)$ le plus grand indice t.q. $\alpha_j < D-1$ alors $C(x_{i+1}) = \alpha_1 \ldots \alpha_j + 1(0)^{l(x_{i+1})-j}$.

Si cette construction se termine, montrons qu'il s'agit d'un code préfixe. La preuve se fait en établissant par induction qu'après l'attribution du code de x_i, on a pour tout $a < b \leq i$:

1. $C(x_a) \leq_{lex} C(x_b)$ où $\leq_{lex}$ désigne l'ordre lexicographique.
2. $\exists u \leq l(x_a) \ C(x_a)[u] < C(x_b)[u]$

Considérons l'itération $i+1$ et soit $a \leq i$. Par construction, $C(x_i) \leq_{lex} C(x_{i+1})$ ce qui établit la première propriété : $C(x_a) \leq_{lex} C(x_{i+1})$ puisque $\leq_{lex}$ est un ordre.

Si $a = i$, alors $C(x_i)[j] < C(x_{i+1})[j]$ où j est l'indice utilisé dans la construction de $C(x_{i+1})$. Si $a < i$, par hypothèse d'induction, $\exists u \leq l(x_a) \ C(x_a)[u] < C(x_i)[u]$ et choisissons le plus petit u. Puisque $C(x_a) \leq_{lex} C(x_i)$, pour tout $s < u$, $C(x_a)[s] = C(x_i)[s]$. Si $u \leq j$, alors $C(x_a)[u] < C(x_i)[u] \leq C(x_{i+1})[u]$. Si $u > j$, alors $C(x_a)[j] = C(x_i)[j] < C(x_{i+1})[j]$. Ce qui établit la deuxième propriété.

Il reste à vérifier qu'on a pu trouver un tel j à chaque étape. On identifie $\bigcup_{i=0}^{l_{max}} \Sigma^i$ à l'arbre de profondeur l_{max} où chaque sommet interne a D fils. Un mot $\alpha_1 \ldots \alpha_k$ correspond au sommet atteint en suivant depuis la racine le fils α_1 puis le fils α_2, etc. Le nombre de feuilles de cet arbre est $D^{l_{max}}$. Un sommet à hauteur h a $D^{l_{max}-h}$ feuilles dans son sous-arbre. Les sommets associés au code préfixe ne partagent pas leurs feuilles mais les feuilles couvertes par les codes qu'on a pu construire sont contiguës (voir la figure 3.1). Supposons maintenant que la construction se bloque avec $C(x_i) = D^{l(x_i)}$ et $i < n$. Toutes les feuilles sont couvertes. Donc :
$\sum_{j \leq i} D^{l_{max}-l(x_j)} = D^{l_{max}}$
D'où :
$\sum_{j \leq i} D^{-l(x_j)} = 1$
Puisque cette somme est partielle, cela contredit l'inégalité de Kraft.

c.q.f.d. ◊◊◊

Nous généralisons le résultat précédent au cas d'un code infini.

Théorème 2 *Les longueurs des mots d'un code C dénombrable uniquement déchiffrable vérifient l'inégalité suivante, dite inégalité de Kraft :*

$$\sum_{x \in \mathcal{X}} |\Sigma|^{-|C(x)|} \leq 1$$

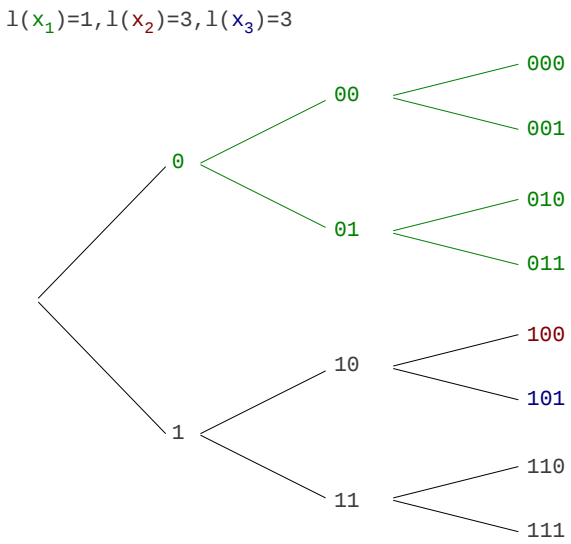


FIGURE 3.1: Des longueurs au code préfixe

De plus, étant donné un code C dénombrable qui vérifie cette inégalité, il est possible de construire un code préfixe C' avec les mêmes longueurs de mots.

Preuve
Soit $\mathcal{X} = \{x_1, \ldots\}$ et $\mathcal{X}_n = \{x_1, \ldots, x_n\}$. C restreint à $\mathcal{X}_n$ est un code uniquement déchiffrable. En vertu du théorème précédent, $\sum_{x \in \mathcal{X}_n} |\Sigma|^{-|C(x)|} \leq 1$. Par passage à limite, $\sum_{x \in \mathcal{X}} |\Sigma|^{-|C(x)|} \leq 1$.

Notons l'alphabet $\Sigma = \{0,1,\ldots,D-1\}$. Posons maintenant $l_i = |C(x_i)|$. On définit les mots du code préfixe par induction :

- $C(x_1) = 0^{l_1}$.
- Soit $C(x_i) = \alpha_1 \ldots \alpha_{l_i}$ et soit $j \leq l_i$ le plus grand indice t.q. $\alpha_j < D-1$ alors $C(x_{i+1}) = \alpha_1 \ldots \alpha_j + 1(0)^{l_{i+1}-j}$.

Si cette construction se termine, il s'agit d'un code préfixe. Si lors d'une étape la construction se bloque alors on aboutit à une contradiction en utilisant la preuve du cas fini pour l'alphabet réduit aux caractères examinés.

Nous décrivons aussi une autre preuve qui introduit l'idée du codage arithmétique. Associons au mot du code $\alpha_1 \ldots \alpha_{l_i}$ l'intervalle $I_i = [0.\alpha_1 \ldots \alpha_{l_i}, 0.\alpha_1 \ldots \alpha_{l_i} + D^{-l_i}[$ où les nombres sont écrits en base D. Les intervalles $I_1, I_2, \ldots$ sont disjoints mais contigus. Si la construction se bloque à l'étape i, alors $\biguplus_{j \leq i} I_j = [0,1[$. En sommant la mesure des intervalles, on obtient $\sum_{j \leq i} D^{-l_j} = 1$ contrairement à l'hypothèse.

c.q.f.d. ◊◊◊

Nous établissons maintenant une borne inférieure (très précise) de la longueur moyenne d'un code qui nous fournit une indication sur la conception de codes optimaux.

Théorème 3 *Soit C un code uniquement déchiffrable sur un alphabet de taille D pour X une v.a. à valeurs dans $\mathcal{X}$. Alors :*

$$L(C) \geq H_D(X)$$

De plus, il y a égalité ssi $D^{-l_i} = p_i$ pour tout $x_i \in \mathcal{X}$ avec p_i la probabilité d'occurrence de x_i et l_i la longueur du code associé.

Preuve
Notons $\mathcal{X} = \{x_1, \ldots, x_n\}$.
$L(C) - H_D(X) = \sum_{i \leq n} p_i l_i + \sum_{i \leq n} p_i \log_D(p_i)$
$= -\sum_{i \leq n} p_i \log_D(D^{-l_i}) + \sum_{i \leq n} p_i \log_D(p_i) = \sum_{i \leq n} p_i \log_D(\frac{p_i}{D^{-l_i}})$
Posons $r_i = \frac{D^{-l_i}}{\sum_{i \leq n} D^{-l_i}}$ et $c = \sum_{i \leq n} D^{-l_i}$.
$L(C) - H_D(X) = \sum_{i \leq n} p_i \log_D(\frac{p_i}{r_i}) - \log_D(c)$
$= D(\vec{p} \parallel \vec{r}) + \log_D(\frac{1}{c})$
≥ 0
en utilisant la positivité de l'entropie relative et l'inégalité de Kraft.
Pour qu'il y ait égalité il faut que $\vec{p} = \vec{r}$ (d'après les propriétés de l'entropie relative) et $c = 1$, ce qui établit $D^{-l_i} = p_i$.

c.q.f.d. $\Diamond\Diamond\Diamond$

3.3 Le codage de Shannon-Fano

Les résultats de la section précédente nous permettent d'obtenir facilement un codage quasi-optimal et une suite de codes asymptotiquement optimale. Soit l'alphabet d'entrée $\mathcal{X} = \{x_1, \ldots, x_n\}$, p_i la probabilité d'occurrence du caractère x_i et D la taille de l'alphabet de codage. Posons $l_i = \lceil -\log_D(p_i) \rceil$. On a :

$$\sum_{i \leq n} D^{-l_i} = \sum_{i \leq n} D^{-\lceil -\log_D(p_i) \rceil} \leq \sum_{i \leq n} D^{\log_D(p_i)} = 1$$

Par conséquent, les longueurs l_i vérifient l'inégalité de Kraft et en appliquant la procédure de la preuve du théorème 1, on obtient un code préfixe appelé code de Shannon-Fano.

Nous majorons (et minorons) la longueur moyenne du code de Shannon-Fano noté C.

$$H_D(p) \leq L(C) = \sum_{i \leq n} p_i l_i = \sum_{i \leq n} p_i \lceil -\log_D(p_i) \rceil < \sum_{i \leq n} p_i(-\log_D(p_i) + 1) = H_D(p) + 1$$

Il est possible d'améliorer la qualité de cette borne, en considérant le codage de Shannon-Fano appliqué à des blocs de k caractères. Soit C_k un tel code, on a :

$$H_D(X_1, \ldots, X_k) \leq L(C_k) < H_D(X_1, \ldots, X_k) + 1$$

Ici les $X_1, \ldots, X_k$ sont des v.a. indépendantes de distribution p. Par conséquent, la proposition 8 implique :
$H_D(X_1, \ldots, X_k) = H_D(X_1) + \ldots + H_D(X_k) = kH_D(p)$
Sachant que l'on code k caractères simultanément :

$$H_D(p) \leq \frac{L(C_k)}{k} < H_D(p) + \frac{1}{k}$$

Par passage à la limite,

$$\lim_{k \to \infty} \frac{L(C_k)}{k} = H_D(p)$$

Cette suite de codes est donc asymptotiquement optimale mais la taille de la table de codage croît exponentiellement en fonction de k.

3.4 Le codage de Huffman

Soit D la taille de l'alphabet de codage, le codage de Huffman consiste à construire un arbre D-aire des feuilles vers la racine. Nous le présentons sur deux exemples avant de développer formellement l'algorithme.

Exemple 1.
Soient $x_1, \ldots, x_5$ les caractères à coder de distribution 0.25,0.25,0.2,0.15,0.15 et choisissons $D = 2$. Les deux premières feuilles de l'arbre sont x_4 et x_5 qui correspondent aux probabilités les plus faibles. On affecte à leur père commun, disons y_1 la somme des probabilités de ses fils, ici 0.3. Nous sommes maintenant en présence de 4 sommets potentiels triés par poids décroissant : y_1,x_1,x_2,x_3. On itère le procédé jusqu'à ce qu'il ne reste plus qu'un sommet : la racine. La figure 3.2 présente l'arbre ainsi construit. Le code associé à chaque caractère est la suite des étiquettes des arêtes 0 ou 1.

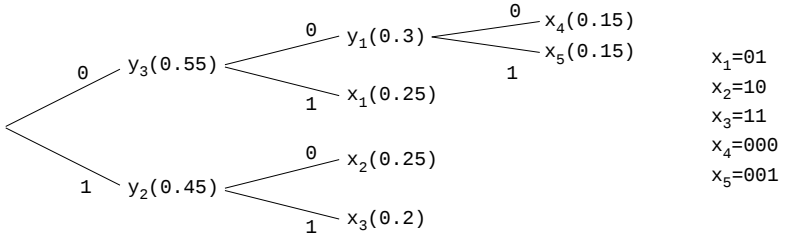


FIGURE 3.2: Un premier arbre de Huffman

Exemple 2.
Soient $x_1, \ldots, x_6$ les caractères à coder de distribution 0.25,0.25,0.2,0.1,0.1,0.1 et choisissons $D = 3$. Puisqu'à chaque étape nous diminuons la liste des sommets à examiner de $D - 1$ (ici 2) nous ne sommes pas certain d'arriver à un sommet

unique. Aussi nous ajoutons des caractères fictifs à l'alphabet d'entrée de probabilité d'occurrence 0. Le nombre de caractères à ajouter est précisément $1-n \mod D-1$ où n est la taille initiale de l'alphabet d'entrée. Dans notre exemple $n=6$, aussi on ajoute un septième caractère fictif. La figure 3.3 présente l'arbre construit.

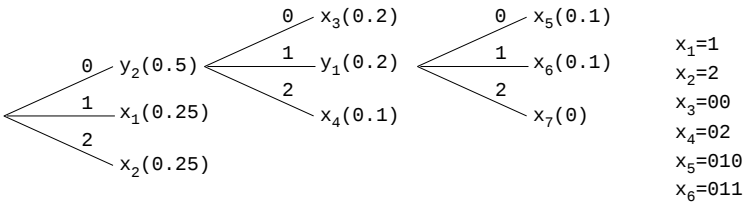


FIGURE 3.3: Un deuxième arbre de Huffman

L'algorithme 27 détaille la construction de l'arbre de Huffman. Les structures de données intermédiaires utilisées sont T' un tableau de sommets avec les fonctions CreerSommet, ModifierPoids et ModifierPere et H un tas binaire avec les fonctions Construire, Inserer, Extraire et Taille. L'algorithme initialise la table des sommets et le tas binaire (tous les deux avec l'attribut poids) à partir de T le tableau en entrée. Il complète éventuellement ces structures avec des sommets fictifs (de probabilité 0). Puis il itère le procédé décrit plus haut : prendre les D éléments de poids le plus faible, créer un sommet père qui cumule les probabilités et l'ajouter au tas binaire. La construction de l'arbre est obtenue par la fonction CreerArbre à partir de la table des sommets. La complexité de l'algorithme est dominée par la gestion du tas binaire en $O(n\log(n))$.

Par construction, le code ainsi défini est un code préfixe puisque les caractères codés sont les feuilles de l'arbre. Nous démontrons maintenant que ce code est optimal. D'après les résultats de la section précédente nous pouvons nous limiter aux codes préfixes.

Lemme 6 *Soit un arbre $\mathcal{A}$ dont les sommets internes ont $d>1$ fils. Alors n_f, le nombre de feuilles de $\mathcal{A}$, vérifie :*

$$n_f \mod d-1=1$$

Preuve
Par récurrence sur la taille de l'arbre. Si l'arbre a un unique sommet qui est une feuille c'est immédiat. Supposons que l'arbre ne soit pas réduit à un seul sommet. La racine de l'arbre $\mathcal{A}$ a d fils et notons $n_1,\ldots,n_d$ le nombre de feuilles des sous-arbres issus de ces fils. Par hypothèse de récurrence, $n_i \mod d-1=1$. D'où $n_f \mod d-1=\sum_i n_i \mod d-1=d \mod d-1=1$.

c.q.f.d. ◊◊◊

Algorithme 27: L'algorithme de Huffman

`Huffman`(T,n,d) : un arbre
Input : T, un tableau de probabilités de dimension n
Input : d, la taille de l'alphabet de sortie
Output : A, l'arbre de Huffman
Data : i,$count$ des indices, p,sp des probabilités
Data : T' un tableau de sommets contenant un père et une probabilité
Data : H un tas binaire

```
count ← n
for i from 1 to n do CreerSommet(T',i,T[i])
Construire(H,T)
for i from 1 to (1 − n) mod (d − 1) do
    count ← count + 1
    Inserer(H,(count,0)) ; CreerSommet(T',count,0)
end
while Taille(H) > 1 do
    count ← count + 1
    CreerSommet(T',count,0)
    sp ← 0
    for i from 1 to d do
        (j,p) ← Extraire(H) ; ModifierPere(T',j,count)
        sp ← sp + p
    end
    ModifierPoids(T',count,sp) ; Inserer(H,(count,sp)) ;
end
A ← CreerArbre(T')
return A
```

Lemme 7 *Soient les probabilités $p_1 \geq p_2 \geq \ldots \geq p_n$ avec $n \mod (d-1) = 1$, alors il existe un code préfixe optimal C sur un alphabet de taille d associé à ces probabilités qui vérifie (en notant l_i la longueur du code associé à p_i) :*

- *Si $p_i > p_j$ alors $l_i \leq l_j$.*
- *Les d mots les plus longs ont la même longueur.*
- *d mots associés aux probabilités les plus faibles partagent le même père.*

Preuve
Supposons que le code C ne vérifie pas la première propriété et échangeons les codes associés à p_i et p_j. Soit C' le nouveau code :

$$L(C) - L(C') = (p_i - p_j)l_i + (p_j - p_i)l_j = (p_i - p_j)(l_i - l_j) > 0$$

contrairement à l'hypothèse.

Soit k le nombre de mots les plus longs et notons l cette longueur.

Supposons que $k < d$. Puisque $k < d$ on peut regrouper ces mots sous un même père sans modifier l'optimalité et supprimer les noeuds dont le sous-arbre ne contient plus de mots.

Si $k = 1$ on « remonte » l'unique mot le plus long (ce n'est pas une contradiction car son poids peut être nul).

Par conséquent, $1 < k < d$. Supposons que tous les sommets internes autres que le père des mots les plus longs aient d fils. Considérons l'arbre dans lequel les mots les plus longs ont été supprimés. D'après le lemme précédent, son nombre de feuilles n' vérifie $n' \mod (d-1) = 1$. Par conséquent, $n \mod (d-1) = n'+k-1 \mod (d-1) = k-1 \neq 0$ contrairement à l'hypothèse. Par conséquent, l'un des autres sommets internes a moins de d fils. On peut donc remonter l'un des mots les plus longs. Ces remontées ne pouvant être effectuées indéfiniment, on obtient que pour un certain code optimal $k \geq d$ ce qui établit la deuxième proposition.

Puisque d mots associés aux probabilités les plus faibles ont même longueur, on peut par échange ou simple replacement garantir que ces d mots ont le même père sans modifier la longueur du code.

c.q.f.d. ◊◊◊

Théorème 4 *Le code de Huffman est optimal.*

Preuve
Soient les probabilités $p_1 \geq p_2 \geq \ldots \geq p_n$, étant donné un code optimal préfixe C qui vérifie les hypothèses du lemme précédent on peut construire un code préfixe C' pour la distribution $p_1, \ldots, p_{n-d}, \sum_{n-d<i\leq n} p_i$ en supprimant les d mots associés à $\{p_i\}_{n-d<i\leq n}$ et partageant le même père et en associant ce père à la probabilité $\sum_{n-d<i\leq n} p_i$.

Soit un code préfixe C'_1 pour $p_1, \ldots, p_{n-d}, \sum_{n-d<i\leq n} p_i$, on peut fabriquer un code préfixe C_1 pour $p_1 \geq p_2 \geq \ldots \geq p_n$ en remplaçant le sommet associé à $\sum_{n-d<i\leq n} p_i$ par un noeud interne et en lui associant comme fils les sommets associés à $\{p_i\}_{n-d<i\leq n}$. On a immédiatement que $l(C_1) = l(C'_1) + \sum_{n-d<i\leq n} p_i$. Par conséquent C' est un code optimal sinon on fabriquerait un code C_1 meilleur que C (voir la figure 3.4).

Autrement dit, on peut transformer à son tour C' pour qu'il vérifie les hypothèses du lemme. Mais ce procédé itératif est exactement l'algorithme de Huffman.

c.q.f.d. ◊◊◊

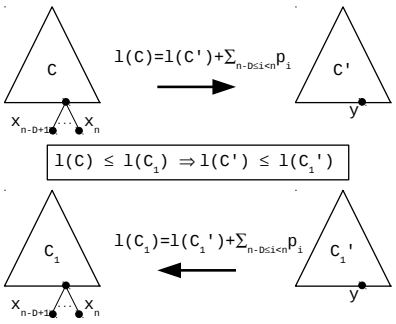


FIGURE 3.4: Optimalité de la construction de Huffman

3.5 Le codage de Shannon-Fano-Elias

Le codage introduit maintenant est « inférieur » au codage de Huffman. Il a cependant un avantage qui apparaîtra lors de la prochaine section. On note dans cette section l'alphabet à coder $\Sigma = \{1,2,\ldots,n\}$ et $p_i = p(i) > 0$ la probabilité d'occurrence de la lettre i. Par souci de simplification, nous choisissons un alphabet de sortie binaire.

On rappelle que la fonction de distribution est définie par $F(x) = \sum_{y \leq x} p(y)$ et on introduit une fonction de distribution modifiée $\bar{F}(x) = \sum_{y<x} p(y) + \frac{1}{2}p(x)$. Puisque les probabilités sont strictement positives, $x \neq y \Rightarrow \bar{F}(x) \neq \bar{F}(y)$. Ce qui signifie que $\bar{F}(x)$ pourrait être un code de x mais $\bar{F}(x)$ est un réel et son écriture dans une base déterminée (ici 2) est potentiellement infinie.

Aussi, nous allons remplacer $\bar{F}(x)$ par $\lfloor \bar{F}(x) \rfloor_{l(x)}$ où $\lfloor z \rfloor_l$ désigne la troncature de $z < 1$ à ces l premiers bits après la virgule. Par définition :

$$0 \leq \bar{F}(x) - \lfloor \bar{F}(x) \rfloor_{l(x)} < 2^{-l(x)}$$

Choisissons $l(x) = \lceil -\log(p(x)) \rceil + 1$. Alors :

$$0 \leq \bar{F}(x) - \lfloor \bar{F}(x) \rfloor_{l(x)} < 2^{-l(x)} \leq 2^{\log(p(x))-1} \leq \frac{p(x)}{2}$$

Par conséquent,

$$\lfloor \bar{F}(x) \rfloor_{l(x)} \in]F(x-1), \bar{F}(x)]$$

et les codes sont donc uniques.

Il reste à vérifier que ce code est un code préfixe. Soit $z_1 z_2 \dots z_l$ un mot (binaire) du code. On peut lui associer l'intervalle des mots dont il est préfixe à savoir $[z_1 z_2 \dots z_l, z_1 z_2 \dots z_l + 2^{-l}[$. Le code est préfixe ssi les intervalles associés sont disjoints. Or :

$$\lfloor \bar{F}(x) \rfloor_{l(x)} + 2^{-l(x)} \leq \bar{F}(x) + 2^{-l(x)} \leq F(x) < \lfloor \bar{F}(x+1) \rfloor_{l(x+1)}$$

et le code est préfixe.

La majoration de la longueur du code est immédiate.

$$L(C) = \sum_x p(x)(\lceil -\log(p(x)) \rceil + 1) < H(X) + 2$$

Exemple.
Soient $1, \dots, 5$ les caractères à coder de distribution 0.25,0.25,0.2,0.15,0.15. Si x est une suite de bits, nous notons $\overline{x}$ la suite infinie $xx \dots$. Le tableau ci-dessous explique la construction du code.

x	$p(x)$	$F(x)$	$\bar{F}(x)$	$\bar{F}(x)$ en binaire	$l(x)$	$\lfloor \bar{F}(x) \rfloor_{l(x)}$
1	0,25	0,25	0,125	0,001	3	001
2	0,25	0,50	0,375	0,011	3	011
3	0,20	0,70	0,600	$0,1\overline{0011}$	4	1001
4	0,15	0,85	0,775	$0,110\overline{0011}$	4	1100
5	0,15	1,00	0,925	$0,111\overline{0110}$	4	1110

En effet, $\overline{0011} = \frac{3}{16} \sum_{i \in \mathbb{N}} \left(\frac{1}{16}\right)^i = \frac{3}{16} \frac{16}{15} = \frac{1}{5}$ et par conséquent $\overline{0110} = \frac{2}{5}$.

3.6 Le codage arithmétique

Lors de la présentation du code Shanon-Fano, nous avons vu qu'en groupant les lettres à coder par blocs, il était possible de se rapprocher asymptotiquement de l'entropie. Cette remarque est valable pour tous les codes qui ont une borne supérieure de type $H(X) + \alpha$ avec α une constante.

Cependant la taille du code croît exponentiellement ($|\Sigma|^b$ avec b la taille du bloc) ce qui rend cette méthode impraticable en toute généralité. Le codage de Shannon-Fano-Elias se généralise de manière astucieuse et fournit un codage asymptotiquement optimal avec un coût de codage et de décodage « modeste ». Ce codage s'appelle le codage arithmétique.

Supposons que nous appliquions le codage de Shannon-Fano-Elias à un message de b caractères $x_1 x_2 \dots x_b$ sur un alphabet $\Sigma = \{1, \dots, n\}$. Les messages sont ordonnés par l'ordre lexicographique.

L'algorithme d'encodage maintient la valeur de $F^-(x_1 \dots x_i)$ (i.e. la probabilité qu'un bloc de i caractères soit strictement inférieur à $x_1 \dots x_i$) et $p(x_1 \dots x_i)$ la probabilité d'occurence du bloc $x_1 \dots x_i$. Chaque itération repose sur les équations suivantes :

$$F^-(x_1 \dots x_i x_{i+1}) = F^-(x_1 \dots x_i) + p(x_1 \dots x_i) F^-(x_{i+1})$$

$$p(x_1 \dots x_i x_{i+1}) = p(x_1 \dots x_i) p(x_{i+1})$$

Algorithme 28: L'algorithme de codage arithmétique

`CodageArith`(n,b,F^-,T) : une suite de bits
Input : n, la taille de l'alphabet d'entrée, b, la taille d'un bloc
Input : F^-, la fonction de répartition stricte de taille $n+1$ avec $F^-(n+1) = 1$
Input : T, le bloc de b caractères à coder
Output : s un réel dans [0,1[codé par sa partie fractionnaire en binaire
Data : i un indice, p,sp des probabilités
Data : l un entier

$sp \leftarrow 0$; $p \leftarrow 1$
for i **from** 1 **to** b **do**
  $sp \leftarrow sp + pF^-[T[i]]$
  $p \leftarrow p(F^-[T[i]+1] - F^-[T[i]])$
end
$sp \leftarrow sp + p/2$
$l \leftarrow \lceil -\log(p) \rceil + 1$
$s \leftarrow \lfloor sp \rfloor_l$
return s

L'algorithme 28 met en oeuvre cette itération. Remarquons que si l'on code le message entier par un réel, alors le fait que le code soit un code préfixe est superflu.

Afin de simplifier la présentation du décodage, nous supposons que le message est codé par un seul bloc. L'algorithme de décodage repose sur les inéquations :

$$F^-(x_1) \leq F^-(x_1x_2) \leq \ldots \leq F^-(x_1 \ldots x_b) < C(x_1 \ldots x_b)$$
$$C(x_1 \ldots x_b) < F^-(x_1 \ldots x_b + 1) \leq \ldots \leq F^-(x_1x_2 + 1) \leq F^-(x_1 + 1)$$

où $C(x_1 \ldots x_b)$ est le code associé à $x_1 \ldots x_b$
et $F^-(x_1 \ldots x_b + 1) = Pr(X_1 \ldots X_b \leq_{lex} x_1 \ldots x_b)$.

L'algorithme 29 consiste essentiellement à décoder les caractères un par un un en maintenant $F^-(x_1 \ldots x_i)$ et $p(x_1 \ldots x_i)$. Remarquons que dans l'algorithme de décodage, on fournit le nombre de caractères codés. Ce nombre représenté en binaire est de taille $\log(b)$. Puisque $\lim_{b\to\infty} \log(b)/b = 0$, ceci ne modifie pas l'optimalité asymptotique du codage.

Les implémentations réalistes du codage arithmétiques sont plus complexes en raison des problèmes de tailles des représentations numériques, de la réalisation des opérations arithmétiques et de la gestion des blocs.

3.7 L'algorithme de Huffman adaptatif

Dans cette section, nous nous limitons à un alphabet de sortie binaire. L'algorithme de Huffman nécessite de connaître la fréquence d'occurrence des caractères et de transmettre cette fréquence (ou l'arbre) afin que le récepteur puisse décoder la transmission. Si la transmission correspond à un fichier qui a été préalablement analysé cela ne pose pas de problème. Si par contre, il s'agit d'un

Algorithme 29: L'algorithme de décodage arithmétique

`DecodageArith`(n,F^-,r,b) : un message
Input : n, la taille de l'alphabet d'entrée
Input : F^-, la fonction de répartition stricte de taille $n+1$ avec $F^-(n+1) = 1$
Input : r, un réel (en binaire) correspondant au message codé
Input : b, le nombre de caractères codés
Output : T, un tableau de b caractères
Data : i,j des indice, p,sp des probabilités

$sp \leftarrow 0$; $p \leftarrow 1$
for i **from** 1 **to** b **do**
  $j \leftarrow 1$
  while $sp + pF^-(j+1) \leq r$ **do** $j \leftarrow j+1$
  $T[i] \leftarrow j$
  $sp \leftarrow sp + pF^-[j]$
  $p \leftarrow p(F^-[j+1] - F^-[j])$
end
return T

flux de données dont l'émetteur prend connaissance au fur et à mesure de la transmission, l'algorithme de Huffman ne peut s'appliquer.

Une solution conceptuellement simple consiste à maintenir l'arbre de Huffman correspondant au flux $x_1 \dots x_i$ et à l'appliquer au caractère x_{i+1}. Pour mettre en oeuvre cette solution, il faut résoudre deux problèmes d'importance différente :

— Obtenir le nouvel arbre de manière incrémentale à partir de l'ancien arbre.
— Gérer la première apparition de chaque caractère.

Puisque la construction d'Huffman est insensible au changement d'échelle, il est plus facile d'associer aux sommets de l'arbre le nombre d'occurrences du sous-ensemble de lettres associés que la fréquence. D'autre part, on peut établir la caractérisation suivante.

Proposition 14 *Soit un arbre associé à un code préfixe tel qu'un sommet u contienne $w(u) > 0$ le nombre d'occurrences des caractères qui se trouvent sur les feuilles de son sous-arbre. Alors cet arbre est un arbre de Huffman ssi il existe une numérotation des sommets $u_1, \dots, u_{2n-1}$ telle que :*

— $\forall i \leq j \ w(u_i) \leq w(u_j)$
— $\forall 1 \leq i < n$, u_{2i-1} *et* u_{2i} *sont des frères dans l'arbre.*

Preuve
Si on dispose de cette numérotation alors on applique l'algorithme de Huffman en choisissant les sommets 1,2, puis 3,4 et on obtient l'arbre original ce qui démontre qu'il s'agit d'un arbre de Huffman. Réciproquement lors de l'exécution de l'algorithme d'Huffman la numérotation des sommets suit l'ordre de traitement des paires de sommets (en respectant les poids respectifs des deux frères).

c.q.f.d. ◊◊◊

Supposons maintenant que nous ayons un arbre d'Huffman et que nous augmentions le nombre d'occurrences d'une feuille, disons u_i. Nous voulons maintenir la propriété énoncée dans la proposition précédente.

— Nous incrémentons $w(u_i)$ puis nous le comparons avec $w(u_{i+1})$. Si $w(u_i) > w(u_{i+1})$, on recherche le plus grand indice $i+k$ t.q. $w(u_{i+k})$ a l'ancienne valeur de $w(u_i)$ et on échange u_i et u_{i+k}. L'échange signifie que les sous-arbres issus de ces deux sommets sont échangés. Observons qu'aucun des sommets ne peut être un descendant de l'autre puisqu'avant la mise à jour les poids étaient identiques. D'autre part, la propriété est maintenant vérifiée de 1 à $i+k$.

— Puis on recommence avec le père de $w(u_i)$ qui doit être aussi incrémenté. La procédure s'arrête à la racine.

La figure 3.5 illustre ce procédé qu'implémente la fonction MajHuffman de l'algorithme 30.

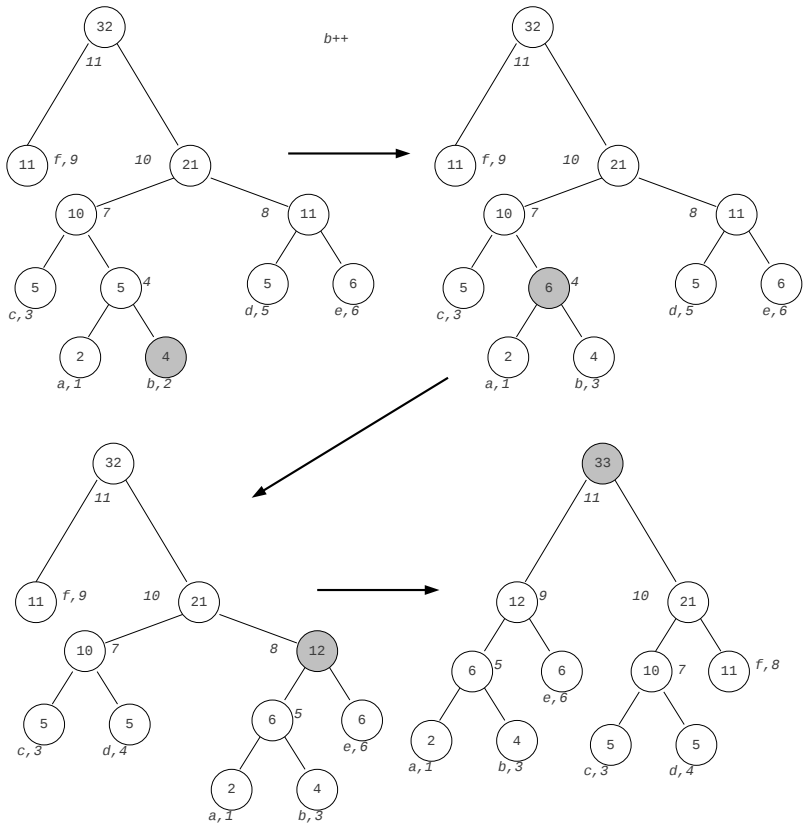


FIGURE 3.5: Mise à jour d'un arbre de Huffman

Puisqu'initialement aucun caractère n'est apparu, nous associons à chaque caractère un code de première apparition. Afin de gérer l'apparition d'un caractère manquant, nous ajoutons un caractère joker, disons ? avec aucune occurrence en même temps que le premier caractère qui apparaît (reconnu par son code de première apparition). Chaque fois qu'un nouveau caractère est ajouté il est obtenu par concaténation du code du ? et du code de première apparition. Le sommet associé au caractère joker devient un sommet interne avec deux fils : un sommet associé au caractère joker et le nouveau caractère. Lorsqu'il ne reste plus qu'un caractère à apparaître, on remplace le caractère joker par ce caractère. La figure 3.6 illustre ce procédé qui est décrit par la fonction AjoutHuffman de l'algorithme 30. Notons que la fonction MajHuffman tient compte du caractère ? afin d'éviter d'échanger le frère de la feuille associée à ? avec son père lors d'une incrémentation.

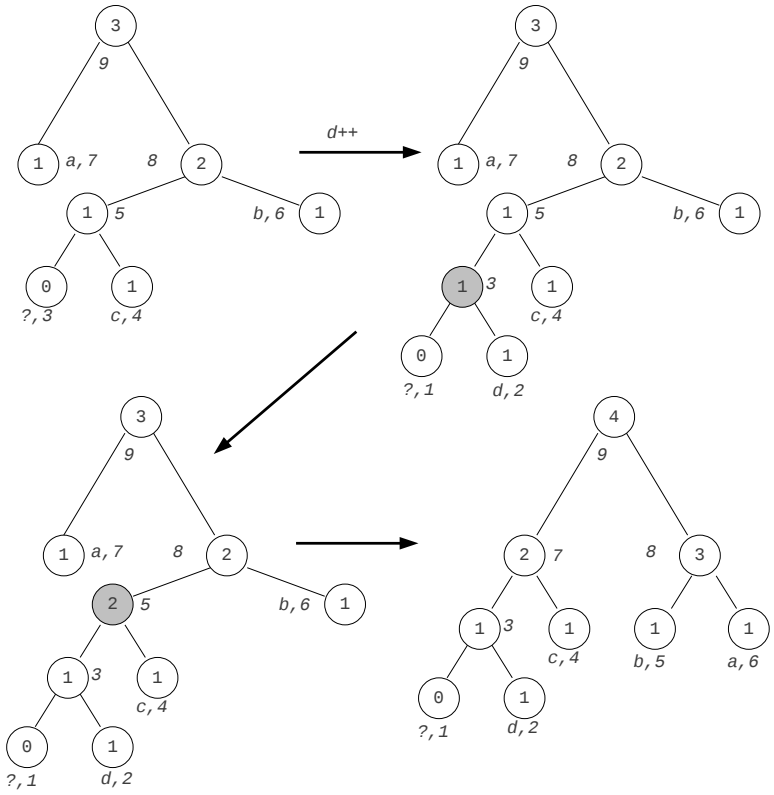


FIGURE 3.6: Apparition d'un nouveau caractère

Algorithme 30: Mise à jour de l'arbre de Huffman

```
MajHuffman(A,u)
Input : A, l'arbre de Huffman
Input : u, un sommet de l'arbre
Data : v un sommet, ex un booléen
if EstFeuille(u) ∧ Frere(u) = LireFeuille(A,?) then
    EcrireW(u,LireW(u) + 1) ; u ← Pere(u)
end
while u ≠ Racine(A) do
    EcrireW(u,LireW(u) + 1)
    v ← u
    while LireW(u) > LireW(Suiv(v)) do  v ← Suiv(v)
    if u ≠ v then Echange(u,v)
    u ← Pere(u)
end
EcrireW(u,LireW(u) + 1)

AjoutHuffman(A,a,n)
Input : A, l'arbre de Huffman
Input : a, un caractère, n le nombre de caractères de l'alphabet
Data : u,v,w des sommets
u ← LireFeuille(A,a)
if u = NULL then
    u ← LireFeuille(A,?)
    if NbCar(A) < n − 1 then
        v ← Creer(A,a,1)
        w ← Creer(A,?,0)
        AjoutFg(u,w) ; AjoutFd(u,v)
    else
        EcrireFeuille(A,u,a)
    end
end
MajHuffman(A,u)
```

3.8 Le codage de Lempel-Ziv

L'algorithme de Lempel-Ziv est aussi un algorithme adaptatif. Cependant son principe est très différent de l'algorithme de Huffman adaptatif. Nous le présentons d'abord avec un fonctionnement en deux passes (lecture du message puis codage) puis indiquons comment modifier l'algorithme pour qu'il travaille à la volée.

Soit le message m, l'algorithme découpe le message en mots tous différents les plus petits possibles. Par conséquent, un mot « reconnu » a forcément pour plus grand préfixe propre un mot déjà reconnu.

Afin de simplifier nos démonstrations, nous supposons que l'alphabet d'entrée est un alphabet binaire. Ces résultats se généralisent sans difficulté au cas d'un alphabet quelconque. On fait aussi l'hypothèse que le message se découpe entièrement (par ajout de bits si nécessaire).

Exemple 1.
Soit l'alphabet $\mathcal{X} = \{0,1\}$, le message 001010 donne lieu à un découpage en trois mots : 0, 01 et 010. En généralisant cet exemple, on obtient un découpage de $2n$ mots pour le message $0,01,\ldots,(01)^{n-1}0,(01)^n$ de longueur $n(2n+1)$. Cet exemple correspond au cas d'un ratio maximal entre le dernier mot et la longueur du message.

Exemple 2.
Soit l'alphabet $\mathcal{X} = \{0,1\}$, le message 0100011011 donne lieu à un découpage en six mots : 0, 1, 00, 01, 10 et 11. En généralisant cet exemple, on obtient un découpage de $2^{n+1}-2$ mots pour le message $0,1,00,\ldots,(1)^{n-1}0,1^n$ de longueur $(n-1)2^{n+1}+2$. Cet exemple correspond au cas d'un ratio maximal entre le nombre de mots obtenus et la longueur du message.

Soit m un message, nous notons $c(m)$ le nombre de mots obtenus à partir de m. Détaillons maintenant le codage de Lempel-Ziv. Le codeur et le décodeur gèrent « simultanément » une table (un dictionnaire) des mots, initialisée avec le mot vide comme première entrée. Le nombre de bits d'un index de cette table doit être $\lceil \log(c(m)+1) \rceil$. Cette quantité n'est connue qu'à la fin du message. Il est cependant très facile d'utiliser le nombre de bits correspondant à la taille courante de la table du codeur qui est la même lors de l'émission de l'index que celle du décodeur lors de la réception de l'index.

Chaque nouveau mot reconnu par l'émetteur est un mot de la table étendu par un bit. Aussi l'émetteur envoie l'index de la table suivi du dernier bit du mot. Puis il met à jour sa table. Le décodeur lit l'index dans sa table complète par le bit qui suit et reconstitue le mot ; il actualise à son tour la table.

Lorsque le flux se termine, la dernière partie du message ne correspond pas nécessairement à un nouveau mot. Aussi l'émetteur le complète par des bits pour en faire un nouveau mot et indique la fin du codage. Pour indiquer qu'il s'agit de la fin du codage, il suffit de répéter la première entrée (0,$m[1]$), soit $\lceil \log(c(m)+1) \rceil + 1 \leq \lceil \log(n)+1 \rceil + 1$ bits supplémentaires. On termine la transmission en envoyant le nombre n de bits significatifs du message. Calculons un majorant du codage du message. La longueur d'un nouveau mot est au plus égale à $\sqrt{2n}$ (voir l'exemple 1), la répétition de la première entrée et la transmission de la longueur du message conduit à $2\lceil \log(n)+1 \rceil + 1$ bits supplémentaires. Le majorant (appelé

B) est donc :

$$B = c(m)\lceil \log(c(m)+1)\rceil + \sqrt{2n} + 2\lceil \log(n)+1\rceil + 1$$

Remarquons que la taille de l'information de contrôle est en $o(n)$.

On suppose que l'occurrence des bits est régie par une distribution de probabilités p sur $\{0,1\}$. L'inégalité suivante est la clé de l'optimalité asymptotique du code.

Proposition 15 (Inégalité de Ziv) *Soit $m = x_1 \dots x_n$ un message binaire et $y_1, y_2, \dots, y_{c(m)}$ la suite de mots reconnus dans m. On note c_l le nombre de mots y_j de longueur l. Alors :*

$$\sum_{l\in\mathbb{N}} c_l \log(c_l) \leq -\sum_{i\leq n} \log(p(x_i))$$

Par conséquent :

$$\limsup_{n\to\infty} \frac{1}{n} \sum_{l\in\mathbb{N}} c_l \log(c_l) \leq H(X) \text{ avec probabilité 1}$$

Preuve
La probabilité d'occurrence d'un mot y_j est le produit des probabilités d'occurrence de ses lettres. Donc :
$-\sum_{i\leq n} \log(p(x_i)) = -\sum_{j\leq c(m)} \log(p(y_j))$
$= -\sum_{l\in\mathbb{N}} c_l \sum_{|y_j|=l} (1/c_l) \log(p(y_j)) \geq -\sum_{l\in\mathbb{N}} c_l \log(\sum_{|y_j|=l} (1/c_l) p(y_j))$
en vertu de la convexité de $-\log$

Or puisque tous les y_j sont distincts $\sum_{|y_j|=l} p(y_j) \leq 1$. Puisque la fonction $-\log$ est décroissante :
$-\sum_{i\leq n} \log(p(x_i)) \geq -\sum_{l\in\mathbb{N}} c_l \log(1/c_l) = \sum_{l\in\mathbb{N}} c_l \log(c_l)$
La deuxième inégalité s'en déduit à l'aide de la proposition 13.

c.q.f.d. ◊◊◊

Cette inégalité est remarquable car le code de Lempel-Ziv ne semble pas tenir compte de l'occurrence des bits. Remarquons déjà que le terme gauche est « proche » de la taille du message codé. La suite des propositions consiste à démontrer que ce terme est asymptotiquement égal à cette taille, ce qui établira l'optimalité asymptotique du codage.

Nous établissons d'abord une borne sur le nombre de mots obtenus en fonction de la taille du message à coder. Le point important est que la « production de mots par bit » tend vers zéro quand la longueur du message tend vers l'infini.

Proposition 16 *Soit m un message binaire de longueur n et $c(m)$ le nombre de mots reconnus dans m. Alors :*

$$c(m) \leq \frac{n}{(1-\varepsilon_n)\log(n)} \text{ avec } lim_{n\to\infty} \varepsilon_n = 0$$

D'où : $lim_{n\to\infty} \frac{c(m)}{n} = 0$

Preuve
Soit $n_k = \sum_{j=1}^{k} j2^j = (k-1)2^{k+1} + 2$. L'exemple 2 montre le pire découpage possible pour un mot m_k de longueur n_k. Dans ce cas :
$c(m_k) = 2^{k+1} - 2 < 2^{k+1} < \frac{n_k}{k-1}$.

Soit maintenant $n_k \leq n < n_{k+1}$. On pose $n = n_k + \Delta$ avec $\Delta < (k+1)2^{k+1}$. Alors le pire découpage est obtenu à partir de celui de n_k en ajoutant $\lfloor \Delta/(k+1) \rfloor$ mots. Par conséquent, $c(m) \leq \frac{n_k}{k-1} + \frac{\Delta}{k-1} = \frac{n}{k-1}$.
On cherche maintenant à estimer l'ordre de grandeur de k.
Tout d'abord $k \leq \log(n_k) \leq \log(n)$.
D'où :
$n < n_{k+1} = k2^{k+2} + 2 \leq (k+2)2^{k+2} \leq (\log(n)+2)2^{k+2}$
Par conséquent :
$k + 2 > \log(\frac{n}{\log(n)+2})$
D'où : $k - 1 > \log(n) - \log(\log(n)+2) - 3$
$= (1 - \frac{\log(\log(n)+2))+3}{\log(n)}) \log(n)$
$\leq (1 - \frac{\log(2\log(n))+3)}{\log(n)}) \log(n)$ pour $n \geq 4$
$\leq (1 - \frac{\log(\log(n))+4}{\log(n)}) \log(n)$
Il suffit alors de poser : $\varepsilon_n = \min(0.5, \frac{\log(\log(n))+4}{\log(n)})$
(0.5 est introduit afin d'éviter une division par zéro)

c.q.f.d. ◊◊◊

Le proposition suivante utilise la maximalité de la loi géométrique pour l'entropie à espérance constante.

Proposition 17 *Soit $m = x_1 \ldots x_n$ un message binaire et $y_1, y_2, \ldots, y_{c(m)}$ la suite de mots reconnus dans m. On note c_l le nombre de mots y_j de longueur l. Alors :*

$$\limsup_{n\to\infty} \frac{c(m)}{n} \log(c(m)) \leq \limsup_{n\to\infty} \frac{1}{n} \sum_{l\in\mathbb{N}} c_l \log(c_l)$$

Preuve
$\frac{1}{n} \sum_{l\in\mathbb{N}} c_l \log(c_l) = \frac{1}{n} \sum_{l\in\mathbb{N}} c_l \log(\frac{c(m)c_l}{c(m)})$
$= \frac{c(m)}{n} \log(c(m)) + \frac{c(m)}{n} \sum_{l\in\mathbb{N}} \frac{c_l}{c(m)} \log(\frac{c_l}{c(m)})$
Introduisons une v.a. U à valeurs sur les longueurs telle que $Pr(U = l) = \frac{c_l}{c(m)}$.
On observe que $E(U) = \sum_{l\in\mathbb{N}} l \frac{c_l}{c(m)} = \frac{n}{c(m)}$. D'après la proposition 12 :
$H(U) \leq (\frac{n}{c(m)} + 1) \log(\frac{n}{c(m)} + 1) - (\frac{n}{c(m)}) \log(\frac{n}{c(m)})$
$= (\frac{n}{c(m)} + 1)(\log(\frac{n}{c(m)} + 1) - \log(\frac{n}{c(m)})) + \log(\frac{n}{c(m)})$
$= (\frac{n}{c(m)} + 1) \log(\frac{c(m)}{n} + 1) + \log(\frac{n}{c(m)})$
En substituant $H(U)$ par sa borne :
$\frac{c(m)}{n} \log(c(m)) = \frac{1}{n} \sum_{l\in\mathbb{N}} c_l \log(c_l) + \frac{c(m)}{n} H(U)$
$\leq \frac{1}{n} \sum_{l\in\mathbb{N}} c_l \log(c_l) + (\frac{c(m)}{n} + 1) \log(\frac{c(m)}{n} + 1) + \frac{c(m)}{n} \log(\frac{n}{c(m)})$
Posons :
$x = \frac{c(m)}{n} \leq \frac{1}{(1-\varepsilon_n)\log(n)}$
On a :$\lim_{n\to\infty} x = 0$
Donc : $\lim_{n\to\infty} (x+1)\log(x+1) = 0$ et $\lim_{n\to\infty} -x\log(x) = 0$
Ce qui établit la proposition.

c.q.f.d. ◊◊◊

Proposition 18 *Le codage de Lempel-Ziv est asymptotiquement optimal. Autrement dit :*

$$\limsup_{n\to\infty} \frac{l(m)}{n} \leq H(X) \text{ avec probabilité 1}$$

avec X la v.a. associée à l'occurrence des bits, m un message aléatoire de longueur n et $l(m)$ la longueur du code associé à m.

Preuve
Le nombre de bits transmis pour un message m de longueur n est borné par :
$B = c(m)\lceil \log(c(m)+1)\rceil + \sqrt{2n} + 2\lceil \log(n)+1\rceil + 1$
D'une part, on observe que $\lim_{n\to\infty} \frac{B}{n} - \frac{c(m)}{n}\log(c(m)) = 0$.
D'autre part, en combinant les propositions 15 et 17, on a :
$\limsup_{n\to\infty} \frac{c(m)}{n}\log(c(m)) \leq H(X)$ avec probabilité 1

c.q.f.d. ◊◊◊

3.9 Loi forte des grands nombres

Rappelons quelques faits élémentaires sur les probabilités que nous utiliserons sans le préciser :

- Soit $\{A_i\}_{i\in I}$ une famille finie ou dénombrable d'événements alors $Pr(\bigcup_{i\in I} A_i) \leq \sum_{i\in I} Pr(A_i)$
- Soit $\{A_i\}_{i\in\mathbb{N}}$ une famille croissante (resp. décroissante) d'événements alors $Pr(\bigcup_{i\in I} A_i) = \lim_{i\to\infty} Pr(A_i)$ (resp. $Pr(\bigcap_{i\in I} A_i) = \lim_{i\to\infty} Pr(A_i)$).

Etant donnée des v.a. $X_1,\ldots,X_n,\ldots$ de même loi et mutuellement indépendantes, la loi forte des grands nombres affirme que l'ensemble des suites obtenues par réalisation des v.a. $M_n = (1/n)\sum_{k\leq n} X_k$ qui convergent vers m, la moyenne de la loi, est de mesure 1. Le lemme ci-dessous est une forme « probabiliste » de critère de Cauchy.

Lemme 8 *Soient $X_1,\ldots,X_n$ des v.a. mutuellement indépendantes avec $E(X_i) = m_i$ et $Var(X_i) = v_i$. Posons $S_k = \sum_{i\leq k} X_i$, $E(S_k) = sm_k = \sum_{i\leq k} m_i$ et $Var(S_k) = sv_k = \sum_{i\leq k} v_i$. Soit $t > 1$:*

$$Pr(\bigvee_{k\leq n} (S_k - sm_k)^2 \geq t^2 sv_n) \leq \frac{1}{t^2}$$

Preuve
Nous notons x la probabilité que l'on cherche à majorer. Nous exprimons x d'une autre façon. Soit Y_k la v.a. définie comme la fonction indicatrice de l'événement :

$$\bigvee_{k'<k} (S_{k'} - sm_{k'})^2 < t^2 sv_n \wedge (S_k - sm_k)^2 \geq t^2 sv_n$$

Autrement dit, Y_k indique que S_k est la première somme à vérifier l'inégalité du lemme. Par définition $\sum_{k\leq n} Y_k \in \{0,1\}$ et

$$x = Pr(\sum_{k\leq n} Y_k = 1) = E(\sum_{k\leq n} Y_k)$$

Puisque $(\sum_{k\leq n} Y_k)(S_n - sm_n)^2 \leq (S_n - sm_n)^2$, on a :

$$\sum_{k\leq n} E(Y_k(S_n - sm_n)^2) = E((\sum_{k\leq n} Y_k)(S_n - sm_n)^2) \leq sv_n \qquad (3.1)$$

Posons :

$$U_k = (S_n - sm_n) - (S_k - sm_k) = \sum_{k<k'\leq n} X_{k'} - m_{k'}$$

$E(Y_k(S_n - sm_n)^2) = E(Y_k(S_k - sm_k)^2) + 2E(Y_kU_k(S_k - sm_k)) + E(Y_kU_k^2)$
D'après la définition de U_k, cette variable est indépendante de $Y_k(S_k - m_k)$ et par conséquent $E(Y_kU_k(S_k - sm_k)) = E(U_k)E(Y_k(S_k - sm_k)) = 0$ puisque $E(U_k) = 0$. Donc : $E(Y_k(S_n - sm_n)^2) \geq E(Y_k(S_k - sm_k)^2)$.

Mais $Y_k \neq 0$ implique $(S_k - sm_k)^2 \geq t^2sv_n$. D'où : $Y_k(S_k - sm_k)^2 \geq Y_kt^2sv_n$ et par passage à l'espérance $E(Y_k(S_n - sm_n)^2) \geq E(Y_k(S_k - sm_k)^2) \geq E(Y_k)t^2sv_n$.

En sommant l'inégalité précédente et en utilisant l'équation 3.1, on obtient :
$sv_n \geq \sum_{k\leq n} E(Y_k(S_n - sm_n)^2) \geq \sum_{k\leq n} E(Y_k)t^2sv_n \geq xt^2sv_n$
ce qui permet de conclure.

c.q.f.d. ◊◊◊

Lemme 9 (Critère de Kolmogorov) *Soient $X_1, \ldots, X_n, \ldots$ des v.a. mutuellement indépendantes avec $E(X_i) = m_i$ et $Var(X_i) = v_i$. Posons $S_k = \sum_{i\leq k} X_i$, $E(S_k) = sm_k = \sum_{i\leq k} m_i$ et $Var(S_k) = sv_k = \sum_{i\leq k} v_i$.*
Supposons que $\sum_{k\in\mathbb{N}} \frac{v_k}{k^2} < \infty$. Alors :

$$\lim_{k\to\infty} \frac{S_k - sm_k}{k} = 0 \qquad \textit{presque sûrement}$$

Preuve
Notons ω une réalisation des v.a. $X_1, \ldots, X_n, \ldots$. Notons A l'ensemble des réalisations ω telles que la suite $\frac{S_k(\omega) - sm_k}{k}$ converge vers 0. $A = \bigcap_{d\in\mathbb{N}} A_d$ avec $A_d = \{\omega \mid \exists k \in \mathbb{N}\ \forall k' \geq k\ |\frac{S_k(\omega) - sm_k}{k}| < d^{-1}\}$. Pour parvenir à la conclusion du lemme, il nous suffit de démontrer que $Pr(A_d) = 1$. Introduisons maintenant $A_{d,k} = \{\omega \mid \forall k' > k\ |\frac{S_k(\omega) - sm_k}{k}| < d^{-1}\}$. On a : $A_d = \bigcup_{k\in\mathbb{N}} A_{d,k}$. Pour démontrer que que $Pr(A_d) = 1$, il nous suffit d'établir que pour tout $\varepsilon > 0$, il existe un k tel que $Pr(A_{d,k}) \geq 1 - \varepsilon$.

Appelons $B_h = \{\omega \mid \exists k \in \mathbb{N}\ 2^{h-1} < k \leq 2^h \wedge |\frac{S_k(\omega) - sm_k}{2^{h-1}}| \geq d^{-1}\}$. Puisqu'au dénominateur, on a substitué à k, $2^{h-1} < k$ on a $\left(\bigcup_{h'\geq h} B_{h'}\right)^c \subseteq A_{d,2^{h-1}}$. D'où $Pr(A_{d,2^{h-1}}) \geq 1 - \sum_{h'\geq h} Pr(B_{h'})$. Si nous démontrons que $\sum_{h\in\mathbb{N}} Pr(B_h) < \infty$, le lemme est établi.

Appliquons le lemme précédent avec $t = \frac{2^{h-1}}{d\sqrt{sv_{2^h}}}$.
On obtient $Pr(B_h) \leq d^2 2^{-2h+2} sv_{2^h}$. Par conséquent :
$\sum_{h\in\mathbb{N}} Pr(B_h) \leq \sum_{h\in\mathbb{N}} d^2 2^{-2h+2} \sum_{k\leq 2^h} v_k$
$= 4d^2 \sum_{k\in\mathbb{N}} v_k \sum_{2^h\geq k} 2^{-2h} \leq 8d^2 \sum_{k\in\mathbb{N}} \frac{v_k}{k^2} < \infty$.

c.q.f.d. ◊◊◊

Lorsque les variables X_k sont identiquement distribuées **et ont une variance finie** la loi forte des grands nombres est une conséquence immédiate du lemme précédent. Pour l'établir sans hypothèse sur la variance, nous nous appuyons sur le lemme suivant.

Lemme 10 (Borel-Cantelli) *Soient $A_1, \ldots, A_k, \ldots$ des événements qui vérifient $\sum_{k\in\mathbb{N}} Pr(A_k) < \infty$. Alors :*

$$Pr(\sum_{k\in\mathbb{N}} 1_{A_k} < \infty) = 1$$

Autrement dit, presque sûrement seul un nombre fini d'événements A_k est simultanément réalisé.

Preuve
Appelons B l'événement $\sum_{k\in\mathbb{N}} 1_{A_k} < \infty$ (un nombre fini d'événements A_k est simultanément réalisé).
Appelons $B_n = \bigcap_{k\geq n} A_k^c$ (aucun des événements A_k pour $k \geq n$ n'est réalisé).
On observe que B_n est une suite croisante et $B = \bigcup_{n\in\mathbb{N}} B_n$.
Il nous suffit donc de démontrer que pour tout $\varepsilon > 0$, il existe un n tel que $Pr(B_n) \geq 1 - \varepsilon$.
Or $Pr(B_n) \geq 1 - \sum_{k\geq n} Pr(A_k)$.
Puisque la série $\sum_{k\in\mathbb{N}} Pr(A_k)$ converge, la conclusion est immédiate.

c.q.f.d. ◊◊◊

Théorème 5 (Loi forte des grands nombres) *Soient $X_1, \ldots, X_n, \ldots$ des v.a. mutuellement indépendantes de même loi avec $E(X_i) = m$. Posons $S_k = \sum_{i\leq k} X_i$, alors :*

$$\lim_{k\to\infty} \frac{S_k}{k} - m = 0 \qquad \textit{presque sûrement}$$

Preuve
Introduisons deux nouvelles familles de variables U_k et V_k telles que $X_k = U_k + V_k$. $U_k = X_k$ si $|X_k| < k$ et $U_k = 0$ sinon. Démontrons que presque sûrement seul un nombre fini d'événements $A_k = \{V_k \neq 0\}$ est simultanément réalisé. La distribution des v.a. X_n est définie par la fonction de répartition F.

Notons $a_k = \int_{k-1}^{k^-} |x| dF\{x\}$.
Par définition, la moyenne existe si $E(|X|) < \infty$.
Or $E(|X|) = \sum_{k\geq 1} a_k$. Donc cette série converge.
$Pr(V_k \neq 0) = \int_{-\infty}^{-k} dF\{x\} + \int_k^{\infty} dF\{x\} \leq \sum_{i\geq k} \frac{a_{i+1}}{i}$
Par conséquent,
$\sum_{k\in\mathbb{N}} Pr(V_k \neq 0) \leq \sum_{k\in\mathbb{N}} \sum_{i\geq k} \frac{a_{i+1}}{i} = \sum_{i\in\mathbb{N}} \sum_{k\leq i} \frac{a_{i+1}}{i} = \sum_{i\in\mathbb{N}} a_{i+1} < \infty$
Le lemme de Borel-Cantelli nous permet alors d'affirmer que presque sûrement $X_k = U_k$ sauf pour un nombre fini d'indices.

Il nous suffit maintenant de démontrer que

$$\lim_{k\to\infty} \frac{\sum_{i\leq k} U_i}{k} - m = 0 \qquad \text{presque sûrement}$$

Posons v_k la variance de U_k.
Par définition des U_k et des a_k, $v_k \leq E(U_k^2) \leq a_1 + 2a_2 + \ldots + ka_k$.
Par conséquent,
$\sum_{k\geq 1} \frac{v_k}{k^2} \leq \sum_{k\geq 1} k^{-2} \sum_{i\leq k} ia_i = \sum_{i\leq k} a_i i \sum_{k\geq i} k^{-2} \leq 2\sum_{i\leq k} a_i < \infty$
On a utilisé ici la majoration standard :
$\sum_{k\geq i} k^{-2} \leq \sum_{k\geq i}(k-1)^{-1}k^{-1} = \sum_{k\geq i}(k-1)^{-1} - k^{-1} = 1/(i-1)$

Le critère de Kolmogorov s'applique et :

$$\lim_{k\to\infty} \frac{\sum_{i\leq k} U_i - m_i}{k} = 0 \qquad \text{presque sûrement avec } m_i = E(U_i)$$

Mais $m_i = \int_{-k^+}^{k^-} dF\{x\}$ converge vers m.
Il en est de même pour $m'_k = \frac{1}{k}\sum_{i\leq k} m_i$. Ce qui permet de conclure.

c.q.f.d. $\Diamond\Diamond\Diamond$

Chapitre 4

Algorithmes d'approximation

Ouvrage recommandé : [Vazirani]

4.1 Généralités

On considère dans ce chapitre des problèmes d'optimisation. Une instance d'un problème d'optimisation consiste en :

- un ensemble de solutions admissibles
- un coût (resp. une récompense) strictement positif $C(sol)$ associé à chaque solution *sol* dans le cas d'un problème de minimisation (resp. maximisation).

La nature des problèmes traités garantit qu'il existe une solution optimale sol^* de coût associé $C^* > 0$.

Etant donné un problème, un algorithme qui prend en entrée une instance de taille n et renvoie une solution *sol* fournit une *garantie de performance* $\rho(n)$ si $\frac{C(sol)}{C^*} \leq \rho(n)$ (resp. $\frac{C^*}{C(sol)} \leq \rho(n)$) dans le cas d'un problème de minimisation (resp. maximisation). On dit plus simplement qu'il s'agit d'un algorithme d'approximation $\rho(n)$.

Un *schéma d'approximation* est un algorithme d'approximation qui prend en entrée une instance et une valeur $\varepsilon > 0$ telle que pour tout ε fixé, l'algorithme est un algorithme d'approximation $1+\varepsilon$. Un *schéma d'approximation polynomial* est un schéma d'approximation qui, pour tout ε fixé, s'exécute en temps polynomial par rapport à la taille de l'instance.

Un *schéma d'approximation entièrement polynomial* est un schéma d'approximation qui s'exécute en temps polynomial par rapport à n et à $1/\varepsilon$.

4.2 Le problème de la couverture de sommets

Soit un graphe $G = (V,E)$, le problème de la couverture de sommets consiste à trouver un sous-ensemble de sommets V' de cardinalité minimale tel que toute arête soit adjacente à au moins un sommet de V'. Ce problème est la version optimisation du problème de décision NP-complet qui prend en entrée un graphe et une valeur entière et renvoie vrai s'il existe une couverture de taille inférieure ou égale à la valeur.

Nous proposons un algorithme qui procède ainsi. Il maintient un ensemble de sommets initialement vide qu'il enrichit itérativement comme suit. Tant qu'il existe une arête non adjacente à l'un des sommets de V', il ajoute les deux sommets adjacents à cette arête à V'. L'algorithme 31 formalise cette procédure `Extraire` extrait une arête arbitraire de l'ensemble considéré et des listes d'ajacence des sommets de l'arête. `Supprimer` extrait les arêtes adjacentes au sommet entrée de l'ensemble considéré.

Analysons la complexité de cet algorithme. Une arête est examinée une et une seule fois soit lorsqu'elle est sélectionnée pour ajouter ses sommets soit lorsqu'elle est supprimée des arêtes à considérer. Par conséquent le temps d'exécution est en $O(|A|)$ si on utilise des listes d'arêtes doublement chainées avec pointeur sur cellule partagée et pointeur retour depuis la cellule (ce qui permet de sélectionner et de supprimer une arête en $O(1)$).

Analysons son rapport d'approximation. Ceci peut sembler problématique de prime abord puisqu'on ne connaît pas le coût optimal. Mais comme assez souvent dans la suite nous allons minorer ce coût. Notons A_{sel} les arêtes sélectionnées. Elles ne partagent pas leurs sommets. Donc $C^* \geq |A_{sel}|$. Or la solution renvoyée a pour coût $2|A_{sel}|$. Autrement dit, l'algorithme est un algorithme de rapport 2.

Algorithme 31: Un algorithme approché pour le problème de couverture

`Couvrir`(G) : sous-ensemble de sommets
Input : $G = (V,E)$, un graphe non orienté
Output : V' une couverture des arêtes
Data : A' un sous-ensemble d'arêtes
$V' \leftarrow \emptyset$; $E' \leftarrow E$
while $E' \neq \emptyset$ **do**
    $(u,v) \leftarrow$ `Extraire`(E')
    $V' \leftarrow V' \cup \{u,v\}$
    `Supprimer`(E',u) ; `Supprimer`(E',v)
end
return V'

4.3 Le problème du voyageur de commerce

4.3.1 Inapproximabilité du problème général

Ce problème est intéressant car il nous permet d'établir un résultat d'inapproximabilité. Rappelons ce problème. On se donne un graphe complet dans les arêtes sont étiquetées par des coûts positifs ou nuls. Le coût d'un chemin est la somme du coût des arêtes. Il s'agit de trouver un cycle hamiltonien de coût minimal.

Théorème 6 *Si $P \neq NP$ alors pour toute constante $\rho \geq 1$, il n'existe pas d'algorithme d'approximation en temps polynomial et à garantie de performances ρ*

Preuve
Soit un graphe $G = (V,E)$, on construit le problème de voyageur suivant $V' = V$ et $c(\{u,v\}) = 1$ si $\{u,v\} \in E$ et $c(\{u,v\}) = \rho|V| + 1$ sinon. Cette construction se fait en temps polynomial en fonction de la taille de G.

Soit une solution *sol* qui comprend une arête n'appartenant pas à E, alors

$$C(sol) \geq \rho|V| + 1 + |V| - 1 > \rho|V| \tag{4.1}$$

Si G possède un cycle hamiltonien alors $C^* = |V|$ sinon d'après l'inégalité précédente, $C^* > \rho|V|$

Supposons qu'il existe un algorithme d'approximation en temps polynomial et à garantie de performances ρ. S'il existe un cycle hamiltonien dans G alors la garantie de performances et l'équation 4.1 implique que l'algorithme renvoie le cycle hamiltonien. S'il n'existe pas de cycle hamiltonien dans G alors la valeur trouvée est strictement plus grande que $\rho|V| > |V|$. Puisque cette réduction est polynomiale, le problème du cycle hamiltonien serait décidable en temps polynomial.

c.q.f.d. ◊◊◊

4.3.2 Le problème avec inégalité triangulaire

On introduit maintenant une restriction naturelle : le coût représente une distance et vérifie l'inégalité triangulaire suivante.

$$\forall u,v,w \in V \; c(\{u,w\}) \leq c(\{u,v\}) + c(\{v,w\})$$

Nous présentons l'algorithme 32 de conception très simple qui fournit un rapport d'approximation constant. Cet algorithme consiste à choisir un sommet r arbitraire, construire un arbre couvrant de poids minimal par l'algorithme de Prim (par construction itérative du l'arbre) ou de Kruskal (par réduction d'une forêt à l'arbre), puis de produire comme tournée la liste des sommets obtenus par parcours préfixe.

Nous avons illustré sur la figure 4.1, la tournée obtenue afin d'expliquer pourquoi cet algorithme garantit un ratio de 2. Les distances correspondent aux distances euclidiennes (ce qui nous évite de toutes les représenter) et vérifient par conséquent l'inégalité triangulaire. On rappelle qu'un parcours préfixe traverse deux fois chaque arc, une fois explicitement en venant du père et une fois implicitement lors du retour de fonction (en supposant une écriture récursive). La liste des noeuds ordonnés suit ce parcours excepté lorsque le noeud suivant le noeud courant nécessite de « remonter » au moins le long d'un arc mais dans ce cas on a substitué à un chemin, une arête. Aussi, en vertu de l'inégalité triangulaire, l'arête ne peut être plus longue que le chemin et la tournée obtenue a pour longueur au plus deux fois la somme des distances des arêtes de l'arbre couvrant. Le retour final à la racine est un cas particulier de ce type de raccourci.

Soit maintenant une tournée de distance minimale. En supprimant la dernière arête, on obtient un arbre couvrant. Par conséquent la longueur de la tournée est supérieure ou égale à la somme des distances des arêtes de l'arbre couvrant. Ce qui nous permet de conclure. Nous avons représenté sur la figure 4.2 la tournée de longueur minimale.

Algorithme 32: Un premier algorithme approché pour le voyageur de commerce

`Tournee`(G,c) : liste ordonnée des sommets
Input : $G = (V,E)$, un graphe non orienté et c un coût des arêtes

Output : L une liste ordonnée de V

Data : r un sommet
Data : T un arbre de sommets V

$r \leftarrow$ un sommet arbitraire de V
$T \leftarrow$ `Arbrecouvmin`(G,c,r)
$L \leftarrow$ la liste des sommets de T obtenue par un parcours préfixe
return L

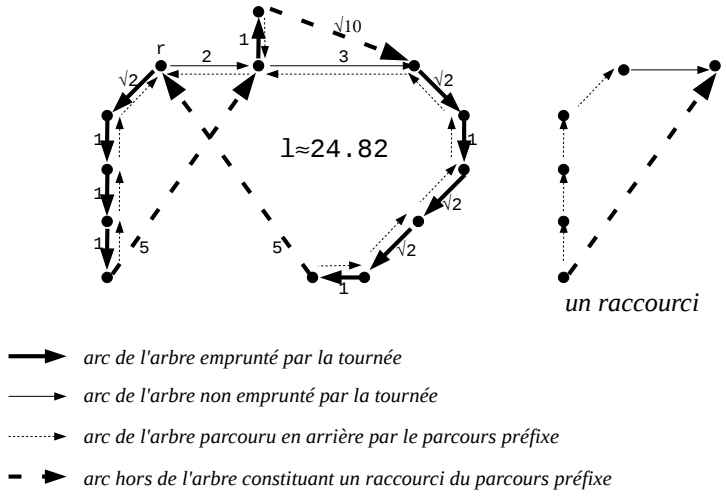


FIGURE 4.1: Une tournée obtenue par l'algorithme 32

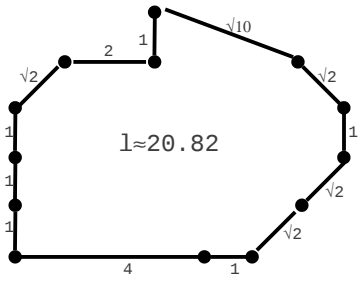


FIGURE 4.2: Une tournée de longueur minimale

4.4 Le problème du couplage maximal

Nous souhaitons améliorer le rapport d'approximation de l'algorithme précédent. Ceci nécessite la résolution en temps polynomial d'un problème qui a son importance propre : le couplage maximal. Afin d'aider à la compréhension de l'algorithme, nous résolvons d'abord le cas d'un graphe non pondéré.

4.4.1 Couplage maximal dans un graphe

Soit $G = (V,E)$ un graphe non orienté, un couplage est un sous-ensemble M de E tel que pour tout $e,e' \in M$ on a $e \cap e' = \emptyset$. Le problème du couplage maximal consiste à trouver un couplage M tel que $m = |M|$ soit maximal. Bien entendu, $m \leq \frac{|V|}{2}$.

Nous cherchons un moyen d'augmenter la taille d'un couplage si celui-ci n'est pas maximal. D'où la définition suivante.

Définition 11 *Soit M un couplage de G un graphe, un chemin (resp. circuit) élementaire $P = (v_0,v_1,\ldots,v_t)$ est dit un chemin M-augmentant (une M-fleur de racine v_0) si :*

- *t est impair*
- *$\{v_1,v_2\},\{v_3,v_4\},\ldots,\{v_{t-2},v_{t-1}\} \in M$*
- *v_0 et v_t ne sont pas couverts par M*

En notant, EP l'ensemble des arêtes de P, un chemin M-augmentant, on observe que $M' = M \Delta EP$ (avec Δ la différence symétrique) est un couplage tel que $|M'| = |M| + 1$. On a en fait un résultat plus fort.

Théorème 7 *Un couplage M est maximal ssi il n'existe pas de chemin M-augmentant.*

Preuve
La condition nécessaire est une conséquence de l'observation précédente. Considérons maintenant deux couplages M,M' avec $|M| < |M'|$. Soit le graphe $G' \equiv (V,M \cup M')$. Tout sommet a au plus deux arêtes adjacentes dans G'. Par conséquent tout composante connexe est soit un chemin élémentaire soit un circuit élémentaire. Puisque $|M| < |M'|$, il existe une composante $C = v_0,v_1,\ldots,v_n$ (notée comme un chemin ou un circuit) qui contient plus d'arêtes de M' que d'arêtes de M. Soit $\{v_i,v_{i+1}\} \in M' \setminus M$, alors en raisonnant par récurrence à partir de $i+1$ et récurrence inversée à partir de i, on déduit que C alterne les arêtes de $M' \setminus M$ et de $M \setminus M'$. Ce ne peut être un circuit car sinon M et M' aurait le même nombre d'arêtes sur C. Il s'agit donc d'un chemin M-augmentant.

c.q.f.d. ◊◊◊

On pourrait donc chercher directement un chemin M-augmentant. Cependant, ce problème ne se réduit pas simplement à un problème d'accessibilité. Aussi nous allons élargir notre recherche aux *promenades M-alternées.*

Définition 12 *Soit M un couplage de G un graphe, un chemin non nécessairement élémentaire $P = (v_0,v_1,\ldots,v_t)$ est dit une promenade M-alternée si :*

- *t est impair*

— $\{v_1,v_2\},\{v_3,v_4\},\ldots,\{v_{t-2},v_{t-1}\} \in M$
— v_0 *et* v_t *ne sont pas couverts par* M

Notons que les chemins M-augmentants et les M-fleurs sont des cas particuliers de promenades M-alternées.

Soient $G = (V,E)$ un graphe et M un couplage, le graphe orienté $D(G,M) = (V,A)$ est défini par $A = \{(u,v) \mid \exists\{u,w\} \in E \setminus M \wedge \exists\{w,v\} \in M\}$. Notons W l'ensemble des sommets non couverts par M et $W^\bullet$ les voisins de W dans G qui sont couverts par M. Un chemin de $u \in W$ à $v \in W^\bullet$ dans $D(G,M)$ fournit une promenade M-alternée en complétant le chemin par une arête de $\{v,w\}$ avec $w \in W$ et vice versa. Par conséquent, la recherche (en temps polynomial) d'un plus court chemin de W à $W^\bullet$ dans $D(G,M)$ soit nous fournit une plus courte promenade M-alternée soit établit qu'il n'en existe pas. De plus, un plus court chemin de W à $W^\bullet$ dans $D(G,M)$ a des propriétés particulières.

Proposition 19 *Soit un chemin* $P \equiv v_0,v_1,\ldots,v_t$ *dans* G *obtenu par « dépliage » d'un plus court chemin de* W *à* $W^\bullet$ *dans* $D(G,M)$. *Alors soit* P *est un chemin élémentaire, soit il existe* $i < j$ *tel que* i *est pair,* j *est impair,* $v_j = v_i$ *et* $P' = v_0,v_1,\ldots,v_{j-1}$ *est un chemin élémentaire.*

Preuve
Supposons que P ne soit pas un chemin élémentaire et notons j le premier indice tel qu'il existe $i < j$ avec $v_i = v_j$. Si $j - i$ était pair alors $P' \equiv v_0,v_1,\ldots,v_{i-1},v_{j+1},\ldots v_t$ serait plus court que P. Par conséquent $j - i$ est impair. Supposons que i est impair alors $\{v_i,v_{i+1}\}$ et $\{v_{j-1},v_j\}$ appartiendraient à M et par conséquent $v_{i+1} = v_{j-1}$ contredisant la définition de j.

c.q.f.d. ◊◊◊

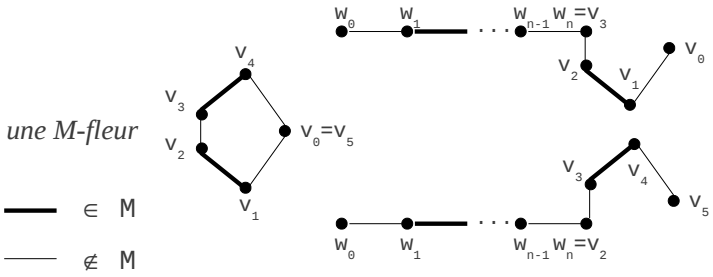
Résumons la situation après la recherche d'un plus court chemin W à $W^\bullet$ dans $D(G,M)$ et son dépliage en un chemin. Il y a quatre cas possibles.
— Il n'existe pas un tel chemin et donc pas de chemin M-augmentant. M est un couplage maximal.
— Ce chemin se complète en un chemin M-augmentant dans G.
— Ce chemin se complète en une M-fleur dans G.
— Ce chemin contient un préfixe de longueur paire $P' = v_0,v_1,\ldots,v_i$ qui est un chemin élémentaire suivi d'un circuit élémentaire de longueur impaire $P'' = v_i,v_1,\ldots,v_j$. Soit $M' = M\Delta EP'$ (EP' l'ensemble des arêtes de P'), M' est un couplage de même taille que M et P'' est une M'-fleur.

Il nous reste donc à traiter le cas de l'obtention d'une M-fleur (ou M'-fleur) lors de la recherche d'un chemin M-augmentant. A cette fin, nous introduisons une opération relative à deux ensembles X,Y que nous appelons le repliage de Y dans X, noté X/Y et défini par $X/Y \equiv X$ si $X \cap Y = \emptyset$ et $X/Y \equiv X \setminus Y \cup \{Y\}$ sinon. Cette opération s'étend aux graphes non orientés. Soit $G = (V,E)$ un graphe et C un sous-ensemble non vide de sommets, $G/C \equiv (V',E')$ avec $V' \equiv V/C$ et $E' \equiv \{\{u,v\} \mid \{u,v\} \in E \wedge \{u,v\} \subseteq V \setminus C\} \cup \{\{u,C\} \mid \{u,v\} \in E \wedge u \in V \setminus C \wedge v \in C\}$.

Théorème 8 *Soit* C *l'ensemble des sommets d'une* M*-fleur* $(v_0,v_1,\ldots,v_t)$, *alors* M *est un couplage maximal de* G *ssi* $M' \equiv M/C$ *est un couplage maximal de* $G' \equiv G/C$.

Preuve
Soit $P \equiv w_1, \ldots, w_n$ un chemin M-augmentant de G. Puisqu'il a deux extrémités, nous supposons qu'il ne démarre pas en v_0, la racine de la M-fleur. Si P ne rencontre pas C alors P est un chemin M-augmentant de G'. Supposons qu'il rencontre C pour la première fois en $w_j = v_i$ alors $w_0, \ldots, w_{j-1}$,C est un chemin M'-augmentant.

Soit $P \equiv w_1, \ldots, w_n$ un chemin M-augmentant de G'. Si P ne rencontre pas C alors P est un chemin M-augmentant de G. Supposons qu'il rencontre $w_j = C$ alors puisque C n'est pas couvert par M, le chemin se termine en C. L'arête $\{w_{n-1}C\}$ correspond à une arête $\{w_{n-1}, v_i\}$. Si i est impair, $w_0, \ldots, w_{n-1}, v_i, v_{i+1}, \ldots, v_t$ est un chemin M-augmentant sinon $w_0, \ldots, w_{n-1}, v_i, v_{i-1}, \ldots, v_0$ est un chemin augmentant.



c.q.f.d. ◊◊◊

Nous avons maintenant tous les éléments nécessaires à la conception de l'algorithme 33. L'algorithme augmente par pas unitaire la taille du couplage maximal. Afin de chercher un chemin M-augmentant (fonction `CheminAugmentant`), il construit le graphe orienté $D(G,M)$ et effectue une recherche de plus court chemin (fonction `PlusCourtChemin`) dans ce graphe entre les sommets non couverts par M et leurs voisins couverts par M. Si un tel chemin n'existe pas alors il n'existe pas de chemin M-augmentant. Sinon on le complète par une arête afin d'atteindre un sommet de W. Si le chemin obtenu est élémentaire alors ce chemin est un chemin M-augmentant et il est renvoyé par l'algorithme. Dans le cas contraire, on en extrait une M-fleur C moyennant un réajustement de M (fonction `DiffSym`). L'algorithme s'appelle alors récursivement sur le graphe G/C pour le couplage M/C et renvoie son résultat après un éventuel dépliage relatif à C (fonction `Depliage`).

Analysons la complexité de cet algorithme. La construction du graphe $D(G,M)$ et la recherche d'un plus court chemin se fait en temps linéaire (i.e. en $O(|V| + |E|)$). Il y a au plus $|V|$ appels récursifs durant une itération et au plus $\lfloor \frac{|V|}{2} \rfloor$ itérations. On obtient donc un algorithme en $O(|V|^2(|V| + |E|))$.

Le théorème qui suit établit un résultat de type $\min - \max$ (similaire au résultat de dualité en programmation linéaire) pour le couplage maximal.

Notations. $m(G)$ désigne la taille d'un couplage maximal de G. $o(G)$ désigne le nombre de composantes connexes de taille impaire de G. Soit U un sous-

Algorithme 33: Un algorithme de couplage maximal

`CouplageMax`(G) : couplage
Input : $G = (V,E)$, un graphe non orienté
Output : M, un couplage maximal de G
Data : P, un chemin
$M \leftarrow \emptyset$
while `True` **do**
  $P \leftarrow$ `CheminAugmentant`(G,M)
  if $P ==$ `NULL` **then return** M **else** $M \leftarrow$ `DiffSym`(M,P)
end
`CheminAugmentant`(G,M) : chemin
Input : $G = (V,E)$, un graphe non orienté, M un couplage de G
Output : P un chemin M-augmentant ou `NULL` s'il n'existe pas
Data : D un graphe orienté
Data : W,$W^\bullet$,C des sous-ensembles de sommets
Output : P' un chemin
$W \leftarrow \{v \mid \nexists u \in V\ \{u,v\} \in M\}$
$W^\bullet \leftarrow \{v \mid \exists u \in V \exists w \in W\ \{u,v\} \in M \wedge \{v,w\} \in E\}$
$D \leftarrow$ `ConstruireD`(G,M)
$P \leftarrow$ `PlusCourtChemin`(D,W,$W^\bullet$)
if $P ==$ `NULL` **then return** `NULL`
// Soit u le dernier sommet de P et $w \in W$ tel que $\{u,w\} \in E$
$P \leftarrow P \cdot w$
if P *est un chemin* M*-augmentant* **then return** P
// Soit $P \equiv v_0,\ldots,v_t$, $j \equiv \min\{k \mid \exists i < k\ v_i = v_k\}$
// et $i < j$ tel que $v_i = v_j$
$M \leftarrow$ `DiffSym`(M,$(v_0,\ldots,v_i)$)
$C \leftarrow \{v_i,\ldots,v_j\}$
$P' \leftarrow$ `CheminAugmentant`(G/C,M/C)
if $P' ==$ `NULL` **then return** `NULL`
$P \leftarrow$ `Depliage`(P',$(v_i,\ldots,v_j)$)
return P

ensemble de sommets de V, le graphe $G - U$ est obtenu par suppression des sommets de U et des arêtes adjacentes.

Théorème 9 (Tutte-Berge) *Pour tout graphe* $G = (V,E)$

$$m(G) = \min_{U \subseteq V} \frac{1}{2}(|V| + |U| - o(G - U)) \tag{4.2}$$

Preuve
Soit M' un couplage d'un graphe G', considérons l'intersection du couplage avec une composante $C = (V(C),E(C))$ $|E(C) \cap M'| \leq \lfloor |C|/2 \rfloor$. En regroupant les composantes selon la parité de leur cardinalité, on obtient $m(G') \leq \frac{1}{2}(|V'| - o(G'))$.

Un couplage de G se décompose en un couplage de $G-U$ et un ensemble d'arêtes dont chacune intersecte U. Par conséquent (en servant de l'inégalité précédente),

$$m(g) \leq |U| + m(G-U) \leq |U| + \frac{1}{2}(|V \setminus U| - o(G-U)) = \frac{1}{2}(|V| + |U| - o(G-U))$$

Nous démontrons l'inégalité inverse (i.e. l'égalité) par récurrence sur $|V|$. Le cas $|V| = 1$ est trivial. Nous supposons que G est connexe car sinon on applique l'hypothèse de récurrence sur chacune de ces composantes et le résultat s'obtient car le minimum d'une somme de termes est supérieur ou égal à la somme des termes minimums.

Supposons d'abord qu'il existe un sommet v couvert par tous les couplages de taille maximale, alors $m(G - \{v\}) = m(G) - 1$ et par hypothèse de récurrence, il existe $U' \subseteq V \setminus \{v\}$ avec

$$m(G) - 1 = m(G - \{v\}) = \frac{1}{2}(|V \setminus \{v\}| + |U'| - o(G - U' - \{v\}))$$

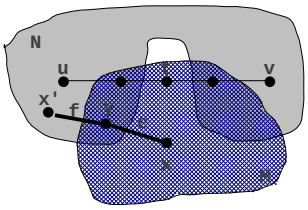
$$= \frac{1}{2}(|V| - 1 + |U' \cup \{v\})| - 1 - o(G - U' - \{v\})) = \frac{1}{2}(|V| + |U' \cup \{v\})| - o(G - U' - \{v\})) - 1$$

Par conséquent $U = U' \cup \{v\}$ convient.

Supposons maintenant qu'il n'existe pas un tel sommet. Par conséquent $m(G) < \frac{1}{2}|V|$. Nous montrons par l'absurde qu'il existe un couplage de taille $\frac{1}{2}(|V|-1)$ ce qui implique le résultat en prenant $U = \emptyset$ dans l'équation 4.2 et en remarquant que $|V|$ est impair..

Supposons que tout couplage maximal M ne couvre pas au moins deux sommets u,v. Choisissons parmi ces triplets (M,u,v) celui qui minimise la distance entre u et v. Cette distance ne peut être égale à 1 car sinon on augmente M avec l'arête $\{u,v\}$. Choisissons un sommet arbitraire t sur un chemin de longueur minimale joignant u à v. t est couvert par M en vertu de la minimalité.

Parmi les couplages de taille maximale qui ne couvrent pas t (il y en a forcément d'après nos hypothèses), choisissons-en un, disons N, qui maximise $|M \cap N|$. La minimalité relative à u,v implique que N couvre u et v. Puisque M et N ont même cardinalité, il existe un sommet $x \neq t$ couvert par M mais pas par N. Soit $e \equiv \{x,y\} \in M$. y est couvert par une arête f de N car sinon on pourrait augmenter N avec e. Dans ce cas $N' = N \setminus \{f\} \cup \{e\}$ est un couplage maximal qui ne couvre pas t, avec $|M \cap N'| > |M \cap N|$. D'où la contradiction.



c.q.f.d. ◊◊◊

4.4.2 Couplage maximal dans un graphe pondéré

Nous considérons maintenant un graphe $G = (V,E)$ dont les arêtes sont pondérées par un coût c. Nous recherchons un couplage M tel que $\sum_{e\in M} c(e)$ soit maximal. Nous allons d'abord transformer le problème. En ajoutant éventuellement un sommet et des arêtes de coût nul, on se limite à la recherche d'un couplage *parfait*, i.e. tel que tout sommet soit couvert. Puis en multipliant par -1 les coûts, on recherche un couplage parfait qui minimise $\sum_{e\in M} c(e)$. Enfin en ajoutant une constante suffisamment grande, on se restreint au cas des coûts positifs ou nuls.

L'algorithme que nous allons développer manipule Ω un ensemble de sous-ensembles de cardinalité impaire qui vérifie :

$$\forall U,U' \in \Omega\ U \cap U' = \emptyset \vee U \subset U' \vee U' \subset U$$

On peut imaginer Ω comme un arbre dont la racine est V et les autres sommets sont les ensembles de Ω avec la contrainte U' est fils de U implique $U' \subset U$. De plus, on requiert que pour tout sommet v de V, on ait $\{v\} \in \Omega$, autrement dit les feuilles de cet arbre sont les sommets.

Notations. Soit U un sous-ensemble de sommets, on définit le sous-ensemble d'arêtes $\delta(U) \equiv \{\{u,v\} \mid u \in U \wedge v \notin U\}$. Soit N un sous-ensemble d'arêtes, on définit le coût de ce sous-ensemble $c(N) \equiv \sum_{e\in N} c(e)$.

L'algorithme maintient aussi une fonction de *potentiel* $\pi : \Omega \mapsto \mathbb{Q}$ qui vérifie les propriétés suivantes :

1. $\forall U \in \Omega\ |U| \geq 3 \Rightarrow \pi(U) \geq 0$
2. $\forall e \in E\ \sum_{U\in\Omega, e\in\delta(U)} \pi(U) \leq c(e)$

Soit N un couplage parfait et π une fonction de potentiel. N et π vérifient les inéquations suivantes :

$$c(N) \geq \sum_{e\in N} \sum_{U\in\Omega, e\in\delta(U)} \pi(U) = \sum_{U\in\Omega} \pi(U)|N \cap \delta(U)| \geq \sum_{U\in\Omega} \pi(U)$$

La dernière inégalité est obtenue en décomposant la somme selon que $|U|$ est égal ou supérieur à 1 et en remarquant que puisque $|U|$ est impair, on a $|N\cap\delta(U)| \geq 1$. De plus,

— $\forall e \in N\ \sum_{U\in\Omega, e\in\delta(U)} \pi(U) = c(e)$ transforme la première inégalité en égalité ;

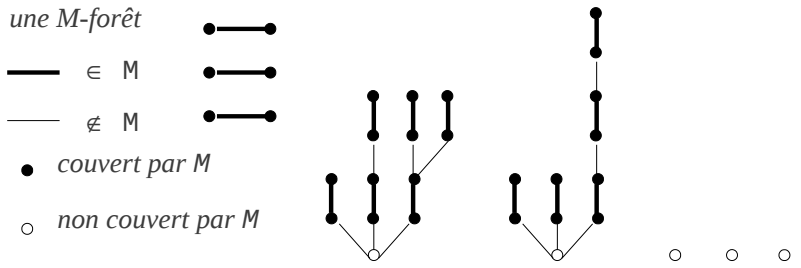
— et $\forall U \in \Omega\ |N \cap \delta(U)| = 1$ transforme la deuxième inégalité en égalité.
L'objectif de l'algorithme est d'obtenir un couplage parfait qui réalise l'égalité. Ainsi on est assuré de posséder un couplage parfait de coût minimal.

Notations. Soit π un potentiel, le coût d'une arête relatif à π est $c_\pi(e) = c(e) - \sum_{U \in \Omega, e \in \delta(U)} \pi(U)$. Le graphe G/Ω est obtenu en repliant les ensembles maximaux de Ω (l'ensemble de ces ensembles est noté $\Omega^{\max}$). Attention, il s'agit ici d'un *multigraphe* car on a besoin de conserver les arêtes et leurs coûts. Soit $U \in \Omega$ avec $|U| \geq 3$, le (multi-)graphe $G|U$, dont l'ensemble des sommets est U, est obtenu en repliant les sous-ensembles maximaux de Ω contenus dans U.

L'algorithme garantit que tout $G|U$ contient un circuit hamiltonien C_U d'arêtes avec $c_\pi(e) = 0$ pour tout $e \in C_U$.

Enfin, l'algorithme maintient un couplage M sur G/Ω et un sous-graphe de G/Ω appelée une M-forêt, notée dans la suite F. Une illustration en est donnée à la figure ci-dessous.

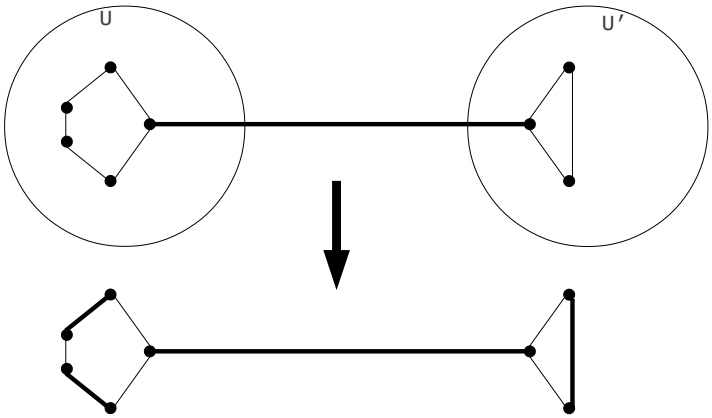
- Les sommets de F sont ceux de G/Ω.
- Toutes les arêtes de M sont des arêtes de F. Toutes les arêtes $e \in F$ vérifient $c_\pi(e) = 0$.
- Les composantes connexes de F sont de deux types : une arête de M et ses deux sommets ou un arbre dont un seul sommet n'est pas couvert par M que nous appellerons racine. Cet arbre orienté à partir de la racine alterne sur tout chemin, des arêtes de M et des arêtes qui n'appartiennent pas à M. Les feuilles sont à une distance paire de la racine.
- On notera $Pair(F)$ (resp. $Impair(F)$) les sommets d'un arbre à distance paire (resp. impaire) de la racine. Observez que certains sommets n'appartiennent ni à $Pair(F)$ ni à $Impair(F)$



Nous avons maintenant tous les éléments pour décrire l'algorithme.

Initialisation. Ω est constitué des singletons $\{v\}$ associés aux sommets avec $\pi(\{v\}) = 0$. M est vide. F est l'ensemble des sommets isolés.

Terminaison. M est un couplage parfait de G/Ω. On transforme M en un couplage parfait de G qui vérifie $\forall e \in M\ c_\pi(e) = 0$ et $\forall U \in \Omega\ |M \cap \delta(U)| = 1$ en « dépliant » les ensembles de Ω par une approche descendante illustrée par la figure ci-dessous. D'après le raisonnement fait plus haut, ces propriétés garantissent qu'on a affaire à un couplage minimal.



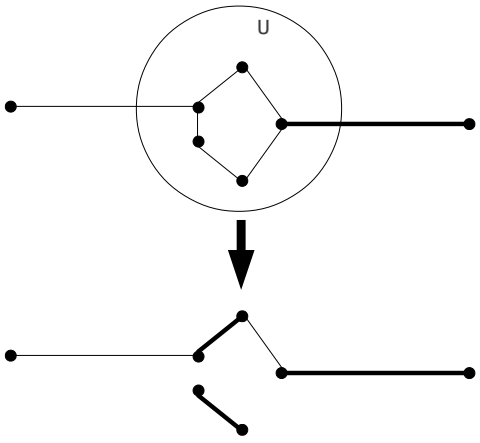
Itération. On met à jour π : $\forall U \in Impair(F)\ \pi(U) \leftarrow \pi(U) - \alpha$ et $\forall U \in Pair(F)\ \pi(U) \leftarrow \pi(U) + \alpha$ avec α la plus grande valeur possible qui garantit que π vérifie les contraintes exigées par l'algorithme. Autrement dit :

$$\alpha \equiv \min(\min(\{\pi(U) \mid |U| \geq 3 \wedge U \in Impair(F)\}),$$

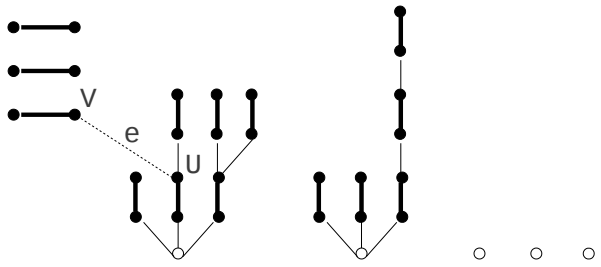
$$\min(\{c_\pi(\{U,V\}) \mid \{U,V\} \in E \wedge U \in Pair(F) \wedge V \notin Impair(F)\}))$$

Remarquons que α est bien défini puisque $\sum_{U\in\Omega} \pi(U)$ est borné par le coût d'un couplage parfait et puisque $|Pair(F)| > |Impair(F)|$ (M n'est pas un couplage parfait). Examinons les différentes situations possibles.

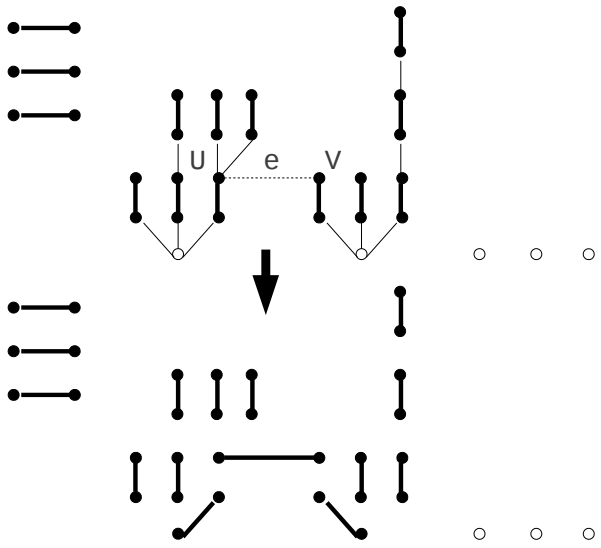
Cas $\exists U \in Impair(F)\ |U| \geq 3 \wedge \pi(U) = 0$**.** Dans ce cas, on retire U de Ω (c_π est inchangé) et on complète de la façon indiquée par la figure ci-dessus en divisant le circuit C_U en deux parties selon les sommets de $G|U$ attachés aux arêtes adjacentes à U dans G/Ω.



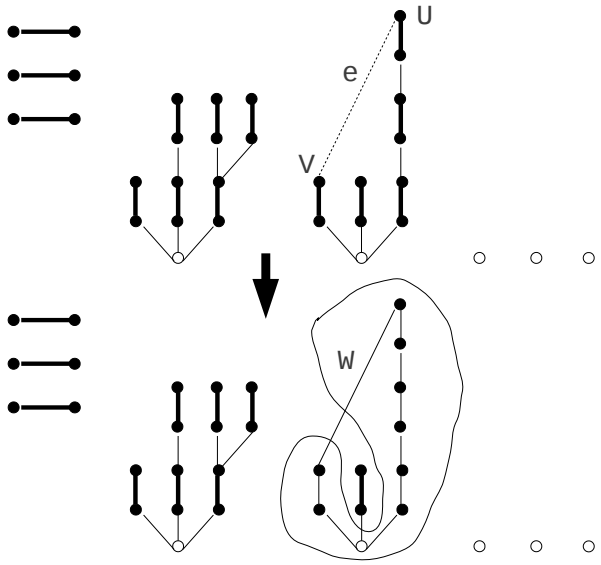
Cas $\exists e = \{U,V\} \in E\ c_\pi(e) = 0 \wedge U \in Pair(F) \wedge V \notin Pair(F) \cup Impair(F)$**.** Dans ce cas, on étend un arbre de F avec l'arête e comme indiqué par la figure ci-dessous.



Cas $\exists e = \{U,V\} \in E\ c_\pi(e) = 0 \wedge U,V \in Pair(F)$ **et** $F \cup \{e\}$ **ne contient pas de circuit.** Dans ce cas, on $F \cup \{e\}$ contient un chemin M-augmentant qu'on utilise pour augmenter la taille de M comme indiqué par la figure ci-dessous.



Cas $\exists e = \{U,V\} \in E\ c_\pi(e) = 0 \wedge U,V \in Pair(F)$ **et** $F \cup \{e\}$ **contient un circuit.** Soit W l'ensemble des sommets du circuit, on ajoute à Ω l'ensemble W, on pose $\pi(W) = 0$ et on met à jour de manière appropriée G/Ω, F et M comme indiqué par la figure ci-dessous.



Preuve de terminaison et estimation de la complexité. Remarquons que la quantité $|V(G/\Omega)| - 2|M|$ reste invariante lors du traitement des cas 1, 2 et 4 et décroît de 2 lors du traitement du cas 3. Il y a donc au plus $\frac{|V|}{2}$ diminutions de $|V(G/\Omega)| - 2|M|$ correspondant aux occurrences du cas 3.

Définissons V_{pair} l'ensemble des sommets de V repliés (même indirectement) dans un sommet de $Pair(F)$ et Ω_0 l'ensemble des sommets de G/Ω qui n'appartiennent pas à $Pair(F)$. Examinons la quantité $2|V_{pair}| + |\Omega_0|$.

- Lors du traitement du cas 1, $|V_{pair}|$ ne diminue pas et $|\Omega_0|$ augmente.
- Lors du traitement du cas 2, $|V_{pair}|$ croît d'au moins 1 et $|\Omega_0|$ décroît de 1.
- Lors du traitement du cas 4, pour chaque sommet de $w \in W$ anciennement dans $Impair(F)$ les sommets de V qu'il contenait sont maintenant dans V_{pair}. Donc $2|V_{pair}| + |\Omega_0|$ croît d'une quantité au moins égale au nombre de sommets anciennement dans $Impair(F)$.

Comme la quantité $2|V_{pair}|+|\Omega_0|$ est bornée par $2|V|$. Il y au plus $2|V|$ itérations entre chaque augmentation de $|M|$ et donc au plus $|V|^2$ itérations. Il reste à vérifier que chaque itération et la terminaison se font en temps polynomial. Le seul point délicat est la taille de l'ensemble Ω. Démontrons que $|\Omega| \leq 2|V|$. prouvons-le par récurrence sur le nombre n de $U \in \Omega$ tel que $|U| \geq 3$. Si $n = 0$ alors $|\Omega| = |V|$. Si $n \geq 1$ alors $\exists U \in \Omega$ tel que U ne contient que des singletons disons $\{\{v_1\}, \ldots, \{v_m\}\}$ avec $m \geq 3$. Posons $V' = V \setminus \{v_1, \ldots, v_m\} \cup \{U\}$ et Ω' obtenu en supprimant de Ω l'ensemble $\{\{v_1\}, \ldots, \{v_m\}\}$. Par récurrence, $|\Omega| - m = |\Omega'| \leq 2|V'| = 2(|V| - m + 1)$. D'où $|\Omega| \leq 2|V| - m + 2 \leq 2|V|$ (l'argument fonctionne aussi avec $m = 2$).

4.4.3 Le problème du voyageur de commerce révisé

A l'aide de l'algorithme de couplage parfait de coût minimal, nous allons proposer un algorithme fournissant une meilleure garantie de performances dans le cas des inégalités triangulaires. Pour cela nous nous appuyons sur un résultat classique.

Définition 13 *Soit $MG = (V,E)$ un multigraphe (i.e. plusieurs arêtes peuvent avoir les mêmes extrémités). Une promenade eulérienne est un ordonnancement $\{u_0,v_0\},\ldots,\{u_{n-1},v_{n-1}\}$ des arêtes de E telle que $\forall 0 \leq i < n\ u_i = v_{i+1 \mod n}$.*

Proposition 20 *Soit $MG = (V,E)$ un multigraphe connexe dont tous les sommets ont un degré pair tel que $|E| > 0$. Alors MG admet une promenade eulérienne.*

Preuve
On choisit arbitrairement un sommet v_0. Puisque G est connexe, il admet une arête d'extrémité $\{v_0,v_1\}$, on choisit cette arête et on poursuit itérativement jusqu'a ce qu'on atteigne un sommet qui n'a plus d'arête. La seule façon de s'arrêter est de revenir en $v_n = v_0$ en raison du degré pair des sommets. Si toutes les arêtes ont été traversées, on a une promenade eulérienne.

Sinon par connexité, il existe un sommet v_i qui dispose d'encore une arête (en fait au moins deux). On étend par le même procédé la promenade $v_i,v_{i+1},\ldots,v_{n-1}$, $v_0,\ldots,v_i$. Cette procédure se termine nécessairement fournissant ainsi une promenade eulérienne.

c.q.f.d. ◊◊◊

Le nouvel algorithme bâtit aussi un arbre couvrant de coût minimal T. L'ensemble V_{pair} des sommets de degré impair de cet arbre est de cardinalité paire. Soit le sous-graphe complet engendré par V_{pair}, il admet un couplage parfait. Donc on peut calculer M un couplage parfait de coût minimal.

Soit la tournée optimale $TO = v_1,\sigma_1,v_2,\sigma_2,\ldots,v_{2m-1},\sigma_{2m-1},v_{2m},\sigma_{2m}$ avec $V_{pair} = \{v_1,\ldots,v_{2m}\}$. On construit les deux couplages :
$M_1 = \{\{v_1,v_2\},\ldots,\{v_{2m-1},v_{2m}\}\}$ et $M_2 = \{\{v_2,v_3\},\ldots,\{v_{2m},v_1\}\}$
En vertu de l'inégalité triangulaire, $c(M_1) + c(M_2) \leq c(TO)$.
Par conséquent, $c(M) \leq c(TO)/2$.

Soit le multigraphe constitué de T et de M. Il est connexe et tous ses sommets sont de degré pair. Il admet une promenade eulérienne $w_1,\sigma_1,w_2,\sigma_2,\ldots,w_n,\sigma_n$ avec $V = \{w_1,\ldots,w_n\}$. De cette promenade eulérienne, on extrait le circuit hamiltonien $w_1,w_2,\ldots,w_n$ de coût inférieur ou égal (par l'inégalité triangulaire) à $c(T) + c(M)$ donc à $\frac{3}{2}c(TO)$. Le résultat de l'algorithme est illustré par la figure 4.3 avec le plus mauvais choix de raccourci.

4.4.4 Le problème du sac à dos

On dispose d'un sac à dos qui peut contenir jusqu'à t kilos et d'un ensemble de n paquets dont les poids entiers non nuls sont notés $X \equiv \{x_1,\ldots,x_n\}$. Il s'agit de sélectionner un sous-ensemble de paquets d'indice I qui maximise la

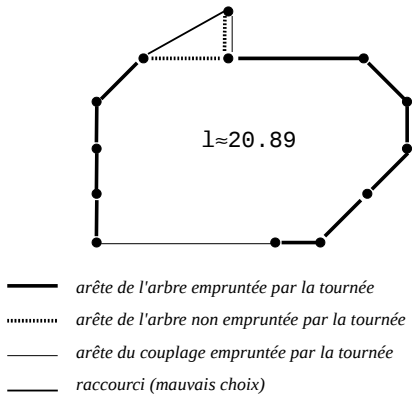


FIGURE 4.3: Une tournée obtenue par l'algorithme révisé

somme des poids sous la contrainte que cette somme n'excède pas t. Autrement dit, on cherche :

$$\max(\sum_{i\in I} x_i \mid I \subseteq \{1,\ldots,n\} \wedge \sum_{i\in I} x_i \leq t)$$

Ce problème est NP-complet ce qui justifie l'emploi d'algorithmes d'approximation.

Afin de concevoir un algorithme d'approximation, nous proposons d'abord un algorithme exact qui s'exécute en temps exponentiel. On note pour $J \subseteq \{1,\ldots,n\}$, $s_J = \sum_{j\in J} x_j$. Au début de l'itération k de l'algorithme 34, L contient la liste (ordonnée par s_J) des sous-ensembles J tels que $J \subseteq \{1,\ldots,k-1\}$ et $s_J \leq t$. Une itération consiste à produire une liste des sous-ensembles étendus avec l'indice k (fonction `Ajout`), fusionner les deux listes (fonction `Fusion`) et purger la liste des sous-ensembles dont la somme associée excède t (fonction `Purge`). A la sortie de la boucle, il suffit de renvoyer un sous-ensemble dont la somme associée est maximale (fonction `Max`).

Observons que les opérations se font toutes en temps polynomial en fonction de la taille des listes mais que cette taille peut croître de façon exponentielle.

L'idée qui sous-tend l'algorithme d'approximation est de conserver une liste de taille polynomiale. Fixons nous un paramètre d'épuration δ. Sachant que la liste est ordonnée, on parcourt la liste par ordre croissant et on supprime un élement de somme s si l'élement précédent a pour somme s' avec $s \leq (1+\delta)s'$. La taille de la liste est inférieure ou égale à $\lfloor \frac{\log(t)}{\log(1+\delta)} \rfloor + 2$. Notons les sommes de la liste $0, s_1, \ldots, s_k$. On a $s_1 \geq 1$ et par induction $s_i \geq (1+\delta)^{i-1}$. Par conséquent $(1+\delta)^{k-1} \leq s_k \leq t$ et le résultat s'obtient en passant au logarithme.

Nous démontrons le lemme suivant qui nous garantit le rapport d'approximation.

Algorithme 34: Un algorithme exact pour le problème du sac à dos

`SacADos`(X,t) : sous-ensemble d'indices et une valeur
Input : X, un ensemble d'entiers indicé par $\{1,\ldots,n\}$ et t un seuil
Output : I un sous-ensemble d'indices et $s = \sum_{i\in I} x_I$
Data : L,L' des listes de paires (J,s) avec $s = \sum_{j\in J} x_j$
Data : k un indice

$L \leftarrow \langle(\emptyset,0)\rangle$
for k **from** 1 **to** n **do**
  $L' \leftarrow$ `Ajout`(k,L)
  $L \leftarrow$ `Fusion`(L,L')
  $L \leftarrow$ `Purge`(t,L)
end
return `Max`(L)

Lemme 11 *L'algorithme d'approximation par épuration avec paramètre $\delta \equiv \varepsilon/2n$ (avec $\varepsilon \leq 1$) garantit un rapport d'approximation ε.*

Preuve
Démontrons qu'à la fin de l'itération k pour tout sous-ensemble $J \subseteq \{1,\ldots,k\}$, il existe un sous-ensemble J' de la liste tel que $\frac{s_J}{(1+\varepsilon/2n)^k} \leq s_{J'} \leq s_J$.

A la fin de la première itération, c'est exactement le résultat de l'épuration. Soit un ensemble $J \subseteq \{1,\ldots,k\}$ et posons $J'' \equiv J \setminus \{k\}$. D'après l'hypothèse de récurrence, il existe J^* élément de la liste à la fin de l'itération $k-1$ tel que $\frac{s_{J''}}{(1+\varepsilon/2n)^{k-1}} \leq s_{J^*} \leq s_{J''}$. Posons $J^+ \equiv J^* \cup \{k\}$ si $k \in J$ et $J^+ \equiv J^*$ sinon. Dans le premier cas on obtient

$$\frac{s_{J''} + x_k}{(1+\varepsilon/2n)^{k-1}} \leq \frac{s_{J''}}{(1+\varepsilon/2n)^{k-1}} + x_k \leq s_{J^*} + x_k \leq s_{J''} + x_k$$

D'où :

$$\frac{s_J}{(1+\varepsilon/2n)^{k-1}} \leq s_{J^+} \leq s_J$$

Dans le deuxième cas, l'encadrement ci-dessus est simplement l'hypothèse de récurrence.
Après épuration il existe un J' tel que

$$\frac{s_J}{(1+\varepsilon/2n)^k} \leq \frac{s_{J^+}}{(1+\varepsilon/2n)} \leq s_{J'} \leq s_{J^+} \leq s_J$$

A la fin de l'algorithme, on a donc pour J de somme s_J maximale et inférieure ou égale à t, un J' dans la liste tel que $\frac{s_J}{(1+\varepsilon/2n)^n} \leq s_{J'} \leq s_J$.
Or $n\log(1+\varepsilon/2n) \leq \varepsilon/2$
Donc $(1+\varepsilon/2n)^n \leq e^{\varepsilon/2}$
Pour $\varepsilon \leq 1$, $e^{\varepsilon/2} \leq 1+\varepsilon$
Ce qui permet de conclure.

c.q.f.d. ◊◊◊

Nous voulons démontrer que cet algorithme est un schéma d'approximation entièrement polynomial. Il nous suffit donc de borner la taille de la liste manipulée.

$\lfloor \frac{\log(t)}{\log(1+\varepsilon/2n)} \rfloor + 2$
$\leq \frac{\log(t)}{\log(1+\varepsilon/2n)} + 2$
$\leq \frac{2n(1+\varepsilon/2n)\log(t)}{\varepsilon} + 2$ *(car* $\log(1+x) \geq \frac{x}{1+x}$ *pour* $x \geq 0$*)*
$\leq \frac{4n\log(t)}{\varepsilon} + 2$ *(dès que* $\varepsilon \leq 1$*)*

Chapitre 5

Algorithmes et probabilités

Ouvrage recommandé : [Mitzenmacher et Upfal]

5.1 Gestion de données dynamiques

5.1.1 Listes à trous

Nous nous intéressons au problème classique d'insérer, de supprimer et de rechercher des données dotées d'une clef à valeurs dans un ensemble totalement ordonné. Les solutions les plus efficaces reposent sur des arbres binaires de recherche « équilibrés ». Il existe plusieurs possibilités d'équilibrer les arbres mais elles requièrent toutes une certaine ingéniosité. La complexité des opérations est alors au pire des cas en $O(\log(n))$ où n est le nombre d'éléments présents dans la structure au moment de l'opération.

Si on se passe de ce mécanisme d'équilibrage, alors moyennant des contraintes d'équiprobabilité sur l'occurrence des clefs et sur l'absence de suppression, on démontre que le temps moyen des opérations se fait aussi en $O(\log(n))$.

Nous allons illustrer l'intérêt des algorithmes probabilistes en montrant qu'on arrive au même résultat (et même mieux) sans aucune hypothèse sur les données et les opérations.

La structure de données que nous étudions s'appelle liste à trous (en anglais *skip list*). Elle est représentée sur la figure 5.1. Il s'agit d'un ensemble de listes doublement chainées superposées par double chaînage. Ces listes vérifient les propriétés suivantes :

— Toutes les listes sont triées.
— La liste la plus basse contient tous les éléments de la structure ainsi que les valeurs bornantes $-\infty, +\infty$.
— Toute autre liste contient un sous-ensemble non nécessairement strict des éléments de la liste en-dessous d'elle ainsi que les valeurs bornantes $-\infty, +\infty$.
— Seule la liste la plus haute est réduite aux valeurs bornantes $-\infty, +\infty$.

Lorsqu'un nouvel élément doit être inséré, on détermine sa position dans la liste la plus basse par le mécanisme de recherche que nous détaillons plus loin. On l'insère dans cette liste, puis par un tirage aléatoire équiprobabiliste, on choisit ou non de l'insérer dans la liste supérieure. Le mécanisme de recherche

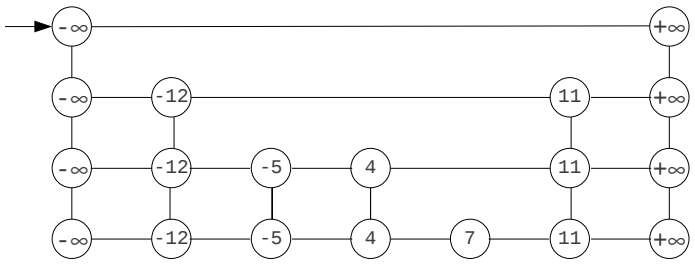


FIGURE 5.1: Listes à trous

garantit que cette insertion se fait en temps constant. Si l'élement a été inséré alors on réitère le procédé. Lorsqu'on insère dans la liste la plus haute, on crée une liste au-dessus. Notons que l'on peut être amené à créer plusieurs listes par l'insertion d'un unique élément.

5.1.2 Analyse de complexité

Dans cette section, tous les log sont en base 2. On utilisera fréquemment l'inégalité *union-somme* : $Pr(\bigcup_{i=1}^{n} E_i) \leq \sum_{i=1}^{n} Pr(E_i)$ et l'inégalité *au moins un* : $Pr(\sum_{i=1}^{n} X_i \geq k) \leq \sum_{i=1}^{n} Pr(X_i \geq \frac{k}{n})$.

Afin d'estimer les temps d'opérations nous établissons d'abord un résultat sur le nombre de listes superposées.

Proposition 21 *Soit H la v.a. associée au nombre de listes. Alors dans une structure à $n > 0$ éléments on a $Pr(H \geq 3\log(n) + 2) \leq \frac{1}{n^2}$*

Preuve
La probabilité qu'il y ait $h+2$ listes correspond à la probabilité qu'au moins un élément de la liste ait été dupliqué h fois. En appliquant la majoration « union-somme », on obtient $Pr(H \geq h + 2) \leq \frac{n}{2^h}$. Par conséquent,
$Pr(H \geq 3\log(n) + 2) \leq \frac{n}{2^{3\log(n)}} = \frac{1}{n^2}$.

c.q.f.d. ◊◊◊

Nous décrivons maintenant les opérations. La recherche d'un élément se pratique ainsi. On parcourt la liste la plus haute jusqu'à ce qu'on rencontre l'élément recherché ou que l'élément suivant soit strictement plus grand que l'élément recherché. Dans le premier cas on descend à la verticale (en vue d'une éventuelle suppression) et dans le deuxième cas on descend à la liste inférieure et on réitère le procédé. A la fin de la procédure, si l'élément est absent, on dispose sur toutes les listes d'un pointeur sur le plus grand élément qui le précède. Nous avons illustré sur la figure 5.2 la recherche de l'élément de clef 8.

Proposition 22 *Soit R la v.a. associée au nombre de parcours de pointeurs lors d'une recherche. Alors dans une structure à $n \geq 16$ éléments on a :*
$Pr(R > \log(n)(12\log(n) + 8)) \leq \frac{2}{n}$ *et* $E(R) \leq 6\log(n) + 6$

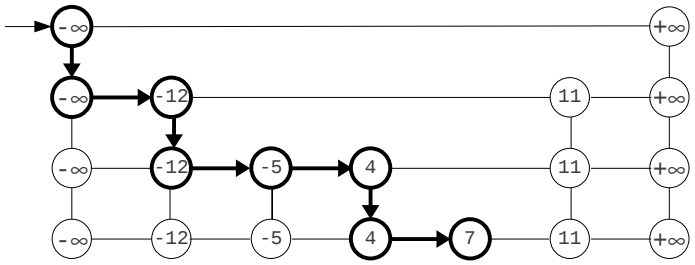


FIGURE 5.2: Recherche de la clef 8

Preuve
Dans la formule ci-dessous on intègre le parcours du pointeur vertical qui accède à la liste i et N_i le nombre de parcours de pointeurs horizontaux de la liste i.
$Pr(R > \log(n)(12\log(n)+8))$
$= Pr(\sum_{1\leq i} 1_{H\geq i}(1+N_i) > \log(n)(12\log(n)+8))$
$\leq Pr(\sum_{1\leq i\leq 3\log(n)+2}(1+N_i) + \sum_{3\log(n)+2<i} 1_{H\geq i}(1+N_i) > \log(n)(12\log(n)+8))$
$\leq Pr(\sum_{1\leq i\leq 3\log(n)+2}(1+N_i) > \log(n)(6\log(n)+4))$
$+Pr(\sum_{3\log(n)+2<i} 1_{H\geq i}(1+N_i) > \log(n)(6\log(n)+4))$
$\leq Pr(\sum_{1\leq i\leq 3\log(n)+2}(1+N_i) > \log(n)(6\log(n)+4)) + Pr(H > 3\log(n)+2)$
$\leq \sum_{1\leq i\leq 3\log(n)+2} Pr(1+N_i > 2\log(n)) + \frac{1}{n^2}$
Pour étudier le parcours de recherche d'un point de vue probabiliste, rien n'empêche d'imaginer qu'il démarre du dernier élément du parcours et remonte jusqu'à l'origine des listes à trous. Soit un élément examiné $e \neq -\infty$ de la liste i, l'élément précédemment examiné peut être soit un certain $e' < e$ sur la liste i, soit e sur la liste $i+1$. Le deuxième cas intervient avec une probabilité $1/2$, d'après le mécanisme d'insertion. Dans le cas où $e = -\infty$, cette probabilité est égale à 1. Par conséquent pour tout i, la v.a. N_i est t.q. $Pr(N_i \geq m) \leq \frac{1}{2^m}$. En reportant dans la dernière inéquation, on obtient :
$Pr(R > \log(n)(12\log(n)+8)) \leq \frac{3\log(n)+2}{n^2} + \frac{1}{n^2}$
$\leq \frac{2}{n}$ dès que $n \geq 16$.
Pour calculer l'espérance de R, on décompose dans le cas défavorable en parcours de pointeurs horizontaux et verticaux. Remarquons que le nombre de parcours de pointeurs horizontaux est inférieur ou égal à n. On se sert aussi de l'inégalité $3\log(n+1) \leq 3\log(n)+1$ dès que $n \geq 4$.
$E(R) = E(\sum_{1\leq i} 1_{H\geq i}(1+N_i))$
$\leq E(\sum_{1\leq i\leq 3\log(n)+2}(1+N_i) + nPr(H > 3\log(n)+2) + \sum_{3\log(n)+2<i} Pr(H \geq i)$
$\leq E(\sum_{1\leq i\leq 3\log(n)+2}(1+N_i)) + 1 + \sum_{3\log(n)+2<i} \frac{1}{i^2}$
$\leq \sum_{1\leq i\leq 3\log(n)+2} E(1+N_i) + 2$
$\leq 6\log(n) + 6$

c.q.f.d. ◊◊◊

La suppression se fait en effectuant une recherche puis en supprimant l'élément dans toutes les listes en partant de la liste la plus basse (ceci peut entraîner

des suppressions des listes les plus hautes). L'insertion se fait en effectuant une recherche puis en insérant l'élément comme indiqué précédemment, la recherche ayant permis d'obtenir les pointeurs appropriés. Dans les deux cas, le surcoût est indépendant de n et suit une loi binomiale de paramètre $1/2$.

5.2 Le problème de la coupe maximale

5.2.1 NP-complétude de la coupe maximale

Le problème de la coupe maximale consiste à partitionner les sommets d'un graphe en deux sous-ensembles tels que le nombre d'arêtes reliant les deux ensembles soit maximal. Contrairement au problème de la coupe minimale qui se résout en temps polynomial (par une recherche d'un flot maximal), ce problème est NP-complet. Afin de le prouver, nous allons procéder en deux temps.

Définition 14 (`MAX2SAT`) *Le problème* `MAX2SAT` *a pour entrées un ensemble de p clauses (avec éventuellement des répétitions) d'au plus deux littéraux et un entier $k \geq 0$. Il consiste à trouver une interprétation des variables tel qu'au moins k clauses soient satisfaites.*

On rappelle que `3SAT` consiste, étant donné un ensemble de clauses d'au plus trois littéraux, à trouver une interprétation des variables tel que toutes les clauses soient satisfaites. Ce problème est `NP`-complet.

Proposition 23 *Le problème* `MAX2SAT` *est* `NP`*-complet.*

Preuve
Nous procédons par réduction en temps polynomial de `3SAT`. En dupliquant si nécessaire certains littéraux, on suppose que toutes les clauses ont exactement trois littéraux.

Nous notons l'ensemble des clauses de l'instance I du problème `3SAT` $\{a_i \vee b_i \vee c_i\}_{1 \leq i \leq p}$. L'instance correspondante I' du problème `MAX2SAT` est définie par les clauses :

$$\bigcup_{i=1}^{p} \{a_i, b_i, c_i, d_i, \neg a_i \vee \neg b_i, \neg a_i \vee \neg c_i, \neg b_i \vee \neg c_i, a_i \vee \neg d_i, b_i \vee \neg d_i, c_i \vee \neg d_i,\}$$

et l'entier $k = 7p$
Ici d_i est une nouvelle variable.

Si I est satisfaisable examinons les différents cas (en tenant compte des symétries) de satisfaction d'une clause :

- $a_i =$ `V` et $b_i = c_i =$ `F` en choisissant $d_i =$ `F` (resp. $d_i =$ `V`), on obtient 7 (resp. 6) clauses satisfaisables.
- $a_i = b_i =$ `V` et $c_i =$ `F` en choisissant $d_i =$ `F` ou $d_i =$ `V`, on obtient 7 clauses satisfaisables.
- $a_i = b_i = c_i =$ `V` en choisissant $d_i =$ `V` (resp. $d_i =$ `F`), on obtient 7 (resp. 6) clauses satisfaisables.

Par conséquent par un choix approprié on obtient $7p$ clauses satisfaites. Remarquons aussi qu'il n'est pas possible d'obtenir plus de 7 clauses satisfaisables par clause de I.

Si I n'est pas satisfaisable alors pour toute interprétation il existe une clause t.q. $a_i = b_i = c_i = \texttt{F}$. Dans ce cas, si $d_i = \texttt{V}$ (resp. $d_i = \texttt{F}$), on obtient 4 (resp. 6) clauses satisfaites. Il n'est donc pas possible d'atteindre l'objectif $7p$.

c.q.f.d. ◊◊◊

Définition 15 (`MAXCUT`) *Le problème* `MAXCUT` *a pour entrées un graphe* $G = (V,E)$ *et un entier* $K > 0$. *Il consiste à trouver une partition des sommets* $V = V_1 \uplus V_2$ *tel que* $|\{\{u,v\} \mid \{u,v\} \in E \wedge u \in V_1 \wedge v \in V_2\}| \geq K$.

Proposition 24 *Le problème* `MAXCUT` *est NP-complet.*

Preuve
Nous procédons par réduction en temps polynomial de `MAX2SAT`. En dupliquant si nécessaire certains littéraux, on suppose que toutes les clauses ont exactement deux littéraux. De même si une clause est de la forme $x \vee \neg x$, on la supprime et on décrémente k. Enfin si $k = 0$, la réduction consiste à associer à I, un graphe constitué d'un unique sommet et d'un objectif $K = 0$.

Soit I une instance du problème `MAX2SAT`. Les clauses de ce problème sont : $\{a_j \vee b_j\}_{1 \leq j \leq p}$ et l'entier est k. Les variables de ce problème sont $x_1, \ldots, x_n$.

La réduction repose sur une construction qui garantit que deux sous-ensembles de sommets sont séparés par toute coupe maximale. Les sommets de l'instance I' correspondante sont :

$$V = \{F\} \cup \{x_i^j, \overline{x}_i^j\}_{1 \leq i \leq n, 0 \leq j \leq 2p}$$

Les arêtes de l'instance I' sont définies par deux sous-ensembles $E = E_1 \uplus E_2$.

$$E_1 = \{\{x_i^j, \overline{x}_i^{j'}\}\}_{1 \leq i \leq n, 0 \leq j,j' \leq 2p}$$

Supposons que nous ayons une partition de $V = V_1 \uplus V_2$. Si pour tout i, tous les x_i^j appartiennent à V_m ($m \in \{1,2\}$) et tous les $\overline{x}_i^j$ appartiennent à V_{3-m} alors toutes les arêtes de E_1 appartiennent à la coupe.

Supposons que l'une des conditions ne soit pas remplie. S'il existe x_i^j et $x_i^{j'}$ n'appartenant pas au même V_m ($m \in \{1,2\}$ dépendant de i) alors $2p+1$ arêtes ne sont pas couvertes par la coupe car pour tout j'' l'une des arêtes $\{\overline{x}_i^{j''}, x_i^j\}, \{\overline{x}_i^{j''}, x_i^{j'}\}$ n'est pas couverte par la coupe. Il en est de même pour toute paire $\overline{x}_i^j$ et $\overline{x}_i^{j'}$. Si maintenant tous les x_i^j et tous les $\overline{x}_i^{j'}$ appartiennent au même V_m alors $(2p+1)^2 \geq 2p+1$ arêtes ne sont pas couvertes par la coupe.

E_2 est un ensemble de p « triangles » dont le seul sommet commun est F. Chaque triangle a 0 ou 2 arêtes couvertes par une coupe selon que les sommets du triangle appartiennent au même ensemble ou pas.

$$E_2 = \{\{a_j^{2j-1}, b_j^{2j}\}, \{a_j^{2j-1}, F\}, \{b_j^{2j}, F\}\}_{1 \leq j \leq p}$$

On choisit pour objectif $K = |E_1| + 2k$. Observons que $|E_1| = n(2p+1)^2$.

Supposons que nous ayons une partition $V = V_1 \uplus V_2$ dont la coupe atteint cet objectif K. D'après nos observations, cet objectif ne peut être atteint que si toutes les arêtes de E_1 sont couvertes par la coupe (sinon $2p+1$ arêtes ne sont pas couvertes) et au moins k triangles n'ont pas leurs sommets dans un même

sous-ensemble de la partition. On définit l'interprétation par $x_i = \mathtt{V}$ si x_i^j et F n'appartiennent pas au même V_m. Cette définition ne dépend pas de j puisque tous les x_i^j appartiennent au même V_m. De plus toujours d'après la contrainte de couvrir les arêtes de E_1, $\overline{x}_i^j$ et F n'appartiennent pas au même ensemble ssi $x_i = \mathtt{F}$ pour cette interprétation. La contrainte sur les k triangles garantit alors que, pour au moins k « j », soit a_j soit b_j est vrai.

Supposons que I admette une interprétation qui satisfait au moins k clauses. Nous construisons la partition suivante.

$$V_1 = \{x_i^j\}_{x_i=\mathtt{V}} \cup \{\overline{x}_i^j\}_{x_i=\mathtt{F}} \text{ et } V_2 = V \setminus V_1$$

Nous laissons le soin au lecteur de vérifier que la coupure associée à cette partition atteint bien l'objectif.

c.q.f.d. ◊◊◊

5.2.2 Algorithmes d'approximation probabiliste et déterministe

Nous cherchons à donc trouver une coupe la plus grande possible. Etudions l'algorithme probabilisé naïf qui consiste à construire une partition en choisissant avec probabilité 1/2 et de manière indépendante l'appartenance d'un sommet à V_1 ou à V_2. Notons X_u la v.a. égale à 1 (resp. 2) si le sommet u appartient à V_1 (resp. V_2). Soit une arête $\{u,v\}$, la probabilité qu'elle appartienne à la coupe est égale à :
$Pr(X_u \neq X_v) = Pr(X_u = 1 \wedge X_v = 2) + Pr(X_u = 2 \wedge X_v = 1)$
$= Pr(X_u = 1)Pr(X_v = 2) + Pr(X_u = 2)Pr(X_v = 1) = 1/4 + 1/4 = 1/2$
Par conséquent la valeur moyenne de la coupe (dont la v.a. est notée X_C) est égale à :
$E(X_C) = E(\sum_{\{u,v\}\in A} 1_{X_u \neq X_v}) = \sum_{\{u,v\}\in A} Pr(X_u \neq X_v) = m/2$
où m est le nombre d'arêtes du graphe.
Cet algorithme a donc une garantie de performances moyenne de 2.

En approfondissant l'analyse probabiliste nous allons déterminiser (« derandomization » en anglais) cet algorithme tout en conservant la garantie de performances. Notons les sommets $v_1, \ldots, v_n$ et supposons que le placement (pour l'instant aléatoire) des sommets se fasse selon l'ordre des indices. Nous notons $E(X_C \mid x_1, \ldots, x_k) \equiv E(X_C \mid X_{v_1} = x_1 \wedge \ldots \wedge X_{v_k} = x_k)$ l'espérance conditionnelle de X_C connaissant les k premiers placements.

Nous allons montrer qu'il est possible de choisir x_{k+1} t.q. $E(X_C \mid x_1, \ldots, x_k) \leq E(X_C \mid x_1, \ldots, x_k, x_{k+1})$. Par définition de l'espérance conditionnelle,
$E(X_C \mid x_1, \ldots, x_k) = \sum_{i\in\mathbb{N}} iPr(X_C = i \mid x_1, \ldots, x_k)$
$= \sum_{i\in\mathbb{N}} i\frac{Pr(X_C=i\wedge X_{v_1}=x_1\wedge\ldots\wedge X_{v_k}=x_k)}{Pr(X_{v_1}=x_1\wedge\ldots\wedge X_{v_k}=x_k)}$
$= \sum_{i\in\mathbb{N}} i(\frac{Pr(X_C=i\wedge X_{v_1}=x_1\wedge\ldots\wedge X_{v_k}=x_k\wedge X_{v_{k+1}}=1)}{Pr(X_{v_1}=x_1\wedge\ldots\wedge X_{v_k}=x_k)}$
$+\frac{Pr(X_C=i\wedge X_{v_1}=x_1\wedge\ldots\wedge X_{v_k}=x_k\wedge X_{v_{k+1}}=2)}{Pr(X_{v_1}=x_1\wedge\ldots\wedge X_{v_k}=x_k)})$
$= Pr(X_{v_{k+1}} = 1 \mid X_{v_1} = x_1 \wedge \ldots \wedge X_{v_k} = x_k)E(X_C \mid x_1, \ldots, x_k, 1)$
$+Pr(X_{v_{k+1}} = 2 \mid X_{v_1} = x_1 \wedge \ldots \wedge X_{v_k} = x_k)E(X_C \mid x_1, \ldots, x_k, 2)$
$= 1/2(E(X_C \mid x_1, \ldots, x_k, 1) + E(X_C \mid x_1, \ldots, x_k, 2))$

Si $k = 0$ alors par symétrie $E(X_C \mid 1) = E(X_C \mid 2) = E(X_C)$. Autrement dit, le choix de x_1 est arbitraire.

D'après l'égalité vue plus haut,
$E(X_C \mid x_1, \ldots, x_k) \leq \max((E(X_C \mid x_1, \ldots, x_k,1),E(X_C \mid x_1, \ldots, x_k,2)))$
Autrement dit, le x_k à sélectionner est celui qui maximise $E(X_C \mid x_1, \ldots, x_k,x_{k+1})$.
Cette quantité se calcule ainsi :
$E(X_C \mid x_1, \ldots, x_k,x_{k+1}) = \sum_{\{u,v\} \in A} Pr(X_u \neq X_v \mid x_1, \ldots, x_k,x_{k+1})$
$= |\{\{v_i,v_j\} \in A \mid \{v_i,v_j\} \subseteq \{v_1, \ldots,v_k\} \wedge x_i \neq x_j\}|$
$+1/2|\{\{v_i,v_j\} \in A \mid \{v_i,v_j\} \cap \{v_{k+2}, \ldots,v_n\} \neq \emptyset\}|$
$+|\{\{v_i,v_{k+1}\} \in A \mid v_i \in \{v_1, \ldots,v_k\} \wedge x_i \neq x_{k+1}\}|$
Le deuxième terme provient du fait que si au moins un des sommets d'une arête n'est pas encore placé alors sa probabilité d'être dans la coupe est $1/2$ (indépendamment des placements déjà effectués). Le troisième terme est le seul qui dépend de x_{k+1}. Autrement dit, pour maximiser $E(X_C \mid x_1, \ldots, x_k,x_{k+1})$ il suffit de choisir de placer v_{k+1} pour maximiser le nombre d'arêtes $\{v_i,v_{k+1}\}$ avec $i \leq k$ appartenant à la coupure. Par récurrence, ce choix assure que $m/2 = E(X_C) \leq E(X_C \mid x_1, \ldots, x_n)$. On a ainsi obtenu un algorithme (déterministe) en temps polynomial qui garantit un rapport d'approximation 2.

5.3 Bornes de Chernoff

Les bornes de Chernoff fournissent un moyen efficace d'établir des garanties d'approximation pour des algorithmes probabilistes. Elles reposent sur deux hypothèses : les choix probabilistes sont indépendants et à valeurs dans 0,1.

L'inégalité de Markov est un résultat élémentaire qui est le point de départ des bornes de Chernoff.

Lemme 12 (Inégalité de Markov) *Soit X une v.a. positive alors*

$$Pr(X \geq a) \leq \frac{E(X)}{a}$$

Preuve
Soit Y une v.a. définie par $Y = a \cdot 1_{X \geq a}$. Y est toujours inférieure ou égale à X. D'où $E(X) \geq E(Y) = aPr(X \geq a)$.

c.q.f.d. ◊◊◊

Ce lemme a pour objectif de rendre exploitables les bornes de Chernoff.

Lemme 13 *Soit $0 \leq \varepsilon < 1$ alors*

$$\frac{e^{\varepsilon}}{(1+\varepsilon)^{1+\varepsilon}} \leq e^{-\frac{\varepsilon^2}{3}} \text{ et } \frac{e^{-\varepsilon}}{(1-\varepsilon)^{1-\varepsilon}} \leq e^{-\frac{\varepsilon^2}{2}}$$

Preuve
En passant au logarithme, la première inégalité est équivalente à :

$$f(\varepsilon) \stackrel{\text{def}}{=} \varepsilon - (1+\varepsilon)\log(1+\varepsilon) + \frac{\varepsilon^2}{3} \leq 0$$

Calculons les dérivées successives de f :
$f'(\varepsilon) = -\log(1+\varepsilon) + \frac{2\varepsilon}{3}$ et $f''(\varepsilon) = -\frac{1}{1+\varepsilon} + \frac{2}{3}$
f'' est négative ou nulle sur [0,1/2] et positive ensuite. Donc f' décroît puis croît. Puisque $f'(0) = 0$ et $f'(1) \approx -0.02 < 0$, f' est négative ou nulle sur [0,1]. Donc f décroît sur cet intervalle. Puisque $f(0) = 0$ l'inégalité s'en déduit.

En passant au logarithme, la deuxième inégalité est équivalente à :

$$g(\varepsilon) \equiv -\varepsilon - (1-\varepsilon)\log(1-\varepsilon) + \frac{\varepsilon^2}{2} \leq 0$$

Calculons les dérivées successives de g :
$g'(\varepsilon) = \log(1-\varepsilon) + \varepsilon$ et $g''(\varepsilon) = -\frac{1}{1-\varepsilon} + 1$
g'' est négative ou nulle sur [0,1[. Donc g' décroît sur cet intervalle. Puisque $g'(0) = 0$, g' est négative ou nulle sur [0,1[. Donc g décroît sur cet intervalle. Puisque $g(0) = 0$ l'inégalité s'en déduit.

c.q.f.d. ◊◊◊

Théorème 10 (Bornes de Chernoff) *Soient $X_1, \ldots, X_n$ des v.a. indépendantes à valeurs dans {0,1} t.q. $Pr(X_i = 1) = p_i$. Soient $X \stackrel{def}{=} \sum_{i \leq n} X_i$ et $\mu \stackrel{def}{=} E(X) = \sum_{i \leq n} p_i$. Soit $0 \leq \varepsilon < 1$. Alors :*

$$Pr(X \geq (1+\varepsilon)\mu) \leq \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{1+\varepsilon}}\right)^{\mu} \leq e^{-\frac{\mu\varepsilon^2}{3}}$$

$$Pr(X \leq (1-\varepsilon)\mu) \leq \left(\frac{e^{-\varepsilon}}{(1-\varepsilon)^{1-\varepsilon}}\right)^{\mu} \leq e^{-\frac{\mu\varepsilon^2}{2}}$$

Preuve
On suppose $\varepsilon > 0$ car le cas $\varepsilon = 0$ est trivial. Pour tout $t > 0$,
$Pr(X \geq (1+\varepsilon)\mu) = Pr(e^{tX} \geq e^{t(1+\varepsilon)\mu}) \leq \frac{E(e^{tX})}{e^{t(1+\varepsilon)\mu}}$
d'après l'inégalité de Markov.
D'autre part,
$E(e^{tX}) = E(e^{t\sum_{i \leq n} X_i}) = E(\prod_{i \leq n} e^{tX_i}) = \prod_{i \leq n} E(e^{tX_i})$
d'après l'indépendance.
$E(e^{tX_i}) = 1 + p_i(e^t - 1) \leq e^{p_i(e^t-1)}$
D'où : $E(e^{tX}) \leq \prod_{i \leq n} e^{p_i(e^t-1)} = e^{\mu(e^t-1)}$
Posons $t = \log(1+\varepsilon)$, on obtient :
$Pr(X \geq (1+\varepsilon)\mu) \leq \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{1+\varepsilon}}\right)^{\mu}$
Pour tout $t < 0$,
$Pr(X \leq (1-\varepsilon)\mu) = Pr(e^{tX} \geq e^{t(1-\varepsilon)\mu}) \leq \frac{E(e^{tX})}{e^{t(1-\varepsilon)\mu}}$
d'après l'inégalité de Markov.
Comme précédemment : $E(e^{tX}) \leq e^{\mu(e^t-1)}$
Posons $t = \log(1-\varepsilon)$, on obtient :
$Pr(X \leq (1-\varepsilon)\mu) \leq \left(\frac{e^{-\varepsilon}}{(1-\varepsilon)^{1-\varepsilon}}\right)^{\mu}$
Les autres inégalités sont des conséquences du lemme précédent.

c.q.f.d. ◊◊◊

Le corollaire qui suit correspond à un cas courant d'utilisation des bornes de Chernoff à savoir un échantillonnage répété d'une même variable aléatoire. (voir la section suivante).

Corollaire 1 *Soient* $0 < p,\delta,\varepsilon \leq 1$ *et* $k \geq \frac{3\log(2/\delta)}{p\varepsilon^2}$. *Soient* $X_1,\ldots,X_k$ *des v.a. indépendantes à valeurs dans* {0,1} *t.q.* $Pr(X_i = 1) = p$. *Alors*

$$Pr(|(1/k)\sum_{i\leq k} X_i - p| < \varepsilon p) \geq 1 - \delta$$

Preuve
$Pr(|(1/k)\sum_{i\leq k} X_i - p| \geq \varepsilon p) = Pr(|\sum_{i\leq m} X_i - kp| \geq \varepsilon kp)$
$= Pr(\sum_{i\leq k} X_i \geq (1+\varepsilon)kp) + Pr(\sum_{i\leq k} X_i \leq (1-\varepsilon)kp)$
$\leq e^{-\frac{kp\varepsilon^2}{3}} + e^{-\frac{kp\varepsilon^2}{2}} \leq 2e^{-\frac{kp\varepsilon^2}{3}} \leq \delta$
d'après le théorème 10

c.q.f.d. ◊◊◊

5.4 Comptage d'interprétations d'une formule DNF

5.4.1 Un algorithme d'approximation non polynomial

Une formule propositionnelle est sous forme normale disjonctive si elle est la disjonction de clauses conjonctives comme dans la formule ci-dessous.

$$(x_1 \wedge \neg x_2 \wedge x_3) \vee (x_2 \wedge x_4) \vee (\neg x_1 \wedge x_3 \wedge x_4)$$

La satisfaisabilité d'une formule DNF est triviale puisqu'il suffit de satisfaire une clause. Un problème plus difficile est de compter le nombre d'interprétations qui satisfont la formule. En effet, etant donnée φ une formule CNF à n variables, sa négation $\neg\varphi$ se réécrit en temps linéaire sous forme DNF disons φ'. φ est satisfaisable ssi φ' admet moins de 2^n interprétations qui la satisfont (notez que la représentation binaire de 2^n est linéaire en fonction de φ). Par conséquent, le problème de savoir si une formule admet au moins (ou exactement) k interprétations est NP-complet.

Afin de formaliser notre objectif, nous introduisons la notion de *schéma d'approximation entièrement polynomial probabilisé*. Nous considérons dans ce cadre des problèmes qui étant donnée une entrée x ont pour résultat une valeur $V(x)$. Un v.a. X est une (ε,δ)-approximation d'une valeur v ssi :

$$Pr(|X - v| \leq \varepsilon v) \geq 1 - \delta$$

Un schéma d'approximation entièrement polynomial probabilisé est un algorithme probabilisé qui prend en entrée une instance x et deux paramètres $0 < \varepsilon,\delta < 1$ t.q. le résultat de l'algorithme est une (ε,δ)-approximation d'une valeur $V(x)$ et qui est polynomial par rapport à la taille de $|x|$, $1/\varepsilon$ et $\log(\delta^{-1})$.

Examinons l'algorithme 35 et notons $c(\varphi)$ le nombre d'interprétations qui satisfont φ. Notons Y_i la v.a. correspondant au résultat de l'interprétation i, $E(Y_i) = c(\varphi)/(2^n)$. Puisque $X = (2^n/k)\sum_{1\leq i\leq m} Y_i$, $E(X) = c(\varphi)$. En appliquant le corollaire 1, on obtient que si $k \geq 3(2^n)\log(2/\delta)/(\varepsilon^2 c(\varphi))$ alors X

Algorithme 35: Un algorithme probabiliste approché pour le comptage d'interprétations

`Compter`(φ) : un entier
Input : φ, une formule DNF à n variables
Output : X une estimation du nombre d'interprétations satisfaisant φ
Data : Y,i des entiers
Data : k une constante entière

$Y \leftarrow 0$
for i **from** 1 **to** k **do**
  Générer de manière équiprobable une interprétation des variables de φ
  if cette interprétation satisfait φ **then** $Y \leftarrow Y + 1$
end
return $2^n(Y/k)$

est une (ε,δ)-approximation de $c(\varphi)$. Malheureusement cet algorithme n'est pas polynomial par rapport à $|\varphi|$ en raison du 2^n au numérateur. Remarquons cependant que si $2^n/c(\varphi)$ est polynomial par rapport à $|\varphi|$ alors on a un schéma d'approximation entièrement polynomial probabilisé (mais bien sûr $c(\varphi)$ n'est pas connu).

5.4.2 Un algorithme d'approximation polynomial

L'insuffisance de l'algorithme précédent tient au fait que son espace d'échantillonnage ne tient pas compte de la formule φ. Ecrivons $\varphi \stackrel{\text{def}}{=} C_1 \vee \ldots \vee C_m$ avec les C_i des clauses conjonctives. On fait l'hypothèse non restrictive qu'une variable et sa négation ne peuvent apparaître dans une clause (sinon on supprime la clause). Soit l_i le nombre de littéraux de la clause C_i, il y a 2^{n-l_i} interprétations qui satisfont C_i. Appelons cet ensemble d'interprétations SC_i et définissons l'ensemble $U \equiv \{(a,i) \mid a \in SC_i\}$. Remarquons que pour tout i, $|SC_i|$ se calcule en temps linéaire et que $|U| = \sum_{i \leq m} |SC_i|$ se calcule en temps polynomial. Enfin le fait qu'une interprétation appartient SC_i se teste aussi en temps linéaire.

Définissons le sous-ensemble S de U par :

$$S \equiv \{(a,i) \mid a \in SC_i \wedge \forall j < i \ (a,j) \notin SC_j\}$$

Cet ensemble est en bijection avec l'ensemble des interprétations satisfaisant φ. Nous avons maintenant tous les ingrédients pour concevoir un meilleur algorithme.

Examinons l'algorithme 36. Notons Y_i la v.a. correspondant au résultat de l'interprétation i, $E(Y_i) = |S|/|U| = c(\varphi)/|U|$.

Puisque $X = (|U|/k) \sum_{1 \leq i \leq k} Y_i$, $E(X) = c(\varphi)$. En appliquant les bornes de Chernoff, on obtient que si $k \geq 3|U| \log(2/\delta)/(\varepsilon^2 c(\varphi))$ alors X est une (ε,δ)-approximation de $c(\varphi)$. On fait maintenant l'observation essentielle que puisqu'une interprétation peut satisfaire au plus t clauses $|U|/c(\varphi) \leq m$ et par conséquent il suffit que $k \geq 3m \log(2/\delta)/\varepsilon^2$ pour que X soit une (ε,δ)-approximation

Algorithme 36: Un autre algorithme probabiliste approché pour le comptage d'interprétations

`Compter`(φ) : un entier
Input : φ, une formule DNF à n variables
Output : X une estimation du nombre d'interprétations satisfaisant φ
Data : Y,j des entiers
Data : k une constante entière

$Y \leftarrow 0$
for j **from** 1 **to** k **do**
  Choisir aléatoirement un indice $i \in \{1, \ldots, m\}$
  avec une probabilité $\frac{|SC_i|}{\sum_{i'} |SC_{i'}|}$
  Générer de manière équiprobable une interprétation de SC_i
  if cette interprétation appartient à S **then** $Y \leftarrow Y + 1$
end
return $|U|(Y/k)$

de $c(\varphi)$. On a donc obtenu un schéma d'approximation entièrement polynomial probabilisé.

Chapitre 6

Programmation linéaire

Ouvrages recommandés : [Roos et al], [Papadimitriou et Steigliz], [Cormen et al, chapitre 29]

6.1 Introduction

Le *temps d'exécution* d'un programme est un facteur primordial. Certains problèmes, bien que disposant d'algorithmes de résolution, ne peuvent être traités que pour des instances de petite taille. Il en est ainsi si l'algorithme a un temps d'exécution exponentiel en fonction de la taille du problème. La *programmation linéaire* (qui a pour but de minimiser une fonction objectif linéaire dans un espace de solutions contraintes par des équations et/ou inéquations linéaires) est intéressante à double titre. Ses applications sont très nombreuses (gestion, économie, réseaux, etc.) et elle dispose d'un algorithme classique appelé *algorithme du simplexe* qui bien qu'ayant une complexité théorique exponentielle s'exécute rapidement dans la plupart des cas pratiques [Cormen et al].

L'espace des solutions, vu comme un objet géométrique, est un polyèdre. Si ce polyèdre est non vide et si la fonction objectif admet un minorant sur l'espace des solutions alors la théorie affirme qu'un des sommets de ce polyèdre est une solution optimale. Etant donné un sommet, le simplexe décide s'il est optimal et dans le cas contraire soit détecte que la fonction n'est pas minorée soit, via une transformation linéaire, lui substitue un sommet qui, s'il est différent de l'ancien sommet, réduit la fonction objectif. Substituer un sommet à un autre s'appelle pivoter. Il y a deux difficultés techniques à résoudre. Lorsqu'on pivote on peut retomber sur le même sommet. Par un choix astucieux de « pivotage » ceci ne peut se répéter indéfiniment. Par conséquent puisque le nombre de sommets est fini, cet algorithme se termine. Il faut aussi trouver un sommet initial. Ce problème se résout en appliquant le simplexe à une variante du problème pour laquelle l'origine est un sommet.

Les *problèmes de flots* constituent un champ d'application naturel pour la programmation linéaire. Ceux-ci se modélisent à l'aide d'un graphe muni d'une source et d'un puits et dont les arêtes sont étiquetées par des capacités de transport. Il s'agit alors de maximiser le flot allant de la source au puits en respectant l'équilibre des flux entrants et sortants en un autre noeud. Ce problème fait l'objet de nombreuses déclinaisons. Par exemple on peut ajouter une information de

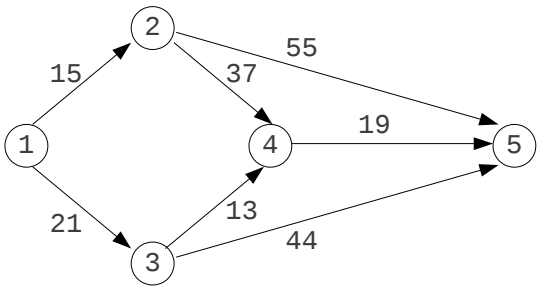


FIGURE 6.1: Un problème de flot maximal

type coût unitaire sur les arêtes. Il s'agit maintenant pour un flot donné de minimiser le coût de transport. Nous avons donc modélisé une structure de graphe. Puis pour chaque problème étudié, nous avons transformé automatiquement le graphe en un programme linéaire équivalent.

Il y a de multiples algorithmes pour résoudre un programme linéaire. Ainsi un problème de minimisation admet un problème dual de maximisation dont la solution fournit moyennant une transformation une solution au problème initial. L'algorithme « primal-dual » [Papadimitriou et Steigliz] est basé sur des idées similaires. Une approche différente est adoptée par les méthodes intérieures [Roos et al] qui déplacent un point à l'intérieur du polyèdre afin d'atteindre une solution optimale. La complexité de ces méthodes est polynomiale même si elles sont parfois moins efficaces que le simplexe en pratique.

6.2 D'un problème de flots à un programme linéaire

Soit le graphe décrit à la figure 6.1. Ce graphe représente une source d'approvisionnement en gaz, le noeud 1, une destination d'exploitation de ce gaz, le noeud 5, un ensemble de canaux de distribution intermédiaires qui ont une capacité maximale et d'embranchements entre les canaux, les noeuds 2, 3 et 4. L'objectif est de maximiser l'arrivée en gaz à la destination en respectant l'équilibre entre les entrées et les sorties aux embranchements. On fait ici l'hypothèse que le débit maximal de la source est supérieur ou égal à la somme des capacités de ses canaux sortants. Ce problème s'exprime sous forme mathématique de la façon suivante (la variable $x_{i,j}$ représente la capacité du canal de i à j utilisée par la solution) :

$$\texttt{Maximiser } x_{1,2} + x_{1,3} \texttt{ tel que :}$$

$$0 \leq x_{1,2} \leq 15 \wedge 0 \leq x_{1,3} \leq 21 \wedge 0 \leq x_{2,4} \leq 37 \wedge 0 \leq x_{3,4} \leq 13$$

$$0 \leq x_{2,5} \leq 55 \wedge 0 \leq x_{3,5} \leq 44 \wedge 0 \leq x_{4,5} \leq 19$$

$$x_{1,2} = x_{2,4} + x_{2,5}, x_{1,3} = x_{3,4} + x_{3,5}, x_{2,4} + x_{3,4} = x_{4,5}$$

Remarquons qu'en ajoutant les variables complémentaires $y_{i,j}$ qui représentent les capacités des canaux non utilisées, on obtient le problème équivalent suivant :

$$\texttt{Maximiser } x_{1,2} + x_{1,3} \texttt{ tel que :}$$

$$\forall (i,j), 0 \leq x_{i,j} \wedge 0 \leq y_{i,j}$$

$$x_{1,2} + y_{1,2} = 15 \wedge x_{1,3} + y_{1,3} = 21 \wedge x_{2,4} + y_{2,4} = 37 \wedge x_{3,4} + y_{3,4} = 13$$

$$x_{2,5} + y_{2,5} = 55 \wedge x_{3,5} + y_{3,5} = 44 \wedge x_{4,5} + y_{4,5} = 19$$

$$x_{1,2} - x_{2,4} - x_{2,5} = 0 \wedge x_{1,3} - x_{3,4} - x_{3,5} = 0 \wedge x_{2,4} + x_{3,4} - x_{4,5} = 0$$

Exercice. Le problème de flots à coût minimal s'énonce ainsi. Les canaux portent ici deux informations : la capacité maximale et le coût unitaire de transport. On se fixe une valeur de flot à transporter de la source à la destination et il s'agit de répartir ce flot sur le réseau afin de minimiser le coût global du transport. La figure 6.2 illustre une instance de ce problème. Ecrire ce problème sous forme d'un programme linéaire.

6.3 Différentes formulations du problème de programmation linéaire

Comme nous l'avons remarqué à propos du problème de flots, différentes formulations caractérisent le même problème. Nous allons le mettre en évidence dans un cadre général.

Notations. Dans la suite, nous parlons de problème PL pour un problème de programmation linéaire. « · » dénote le produit matrice-matrice et ses cas particuliers : matrice-vecteur, vecteur-matrice et le produit scalaire d'un vecteur ligne par un vecteur colonne.

Définition 16 (PL général) *Une instance du problème PL* général *est donnée par les éléments suivants :*

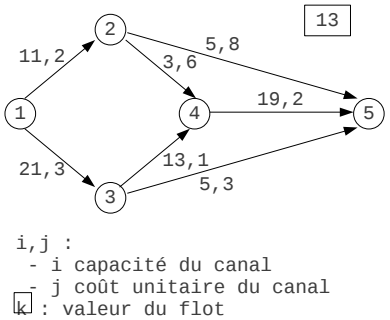


FIGURE 6.2: Un problème de flot à coût minimal

— *I,I', des ensembles d'indices de ligne tels que $I \cap I' = \emptyset$ correspondant aux contraintes du problème. I est associé aux égalités et I' aux inégalités.*
— *J,J', des ensembles d'indices de colonne tels que $J \cap J' = \emptyset$ correspondant aux variables du problème. J est associé aux variables positives et J' aux variables réelles.*
— *A, une matrice $I \cup I' \times J \cup J'$ à coefficients rationnels qui correspond aux combinaisons linéaires des variables associées aux contraintes. $A[i,-]$ (resp. $A[-,j]$) désigne le vecteur ligne $\{1\} \times J \cup J'$ (resp. colonne $I \cup I' \times \{1\}$) de la matrice A d'indice i (resp. j).*
— *b, un vecteur colonne $I \times \{1\}$ à coefficients rationnels correspondant aux valeurs à comparer à ces combinaisons linéaires. On note b_i, la composante d'indice i de ce vecteur.*
— *c, un vecteur ligne par $\{1\} \times J$ à coefficients rationnels correspondant à la combinaison linéaire qui définit la fonction objectif. On note c_j, la composante d'indice j de ce vecteur.*
— *$\{x_j\}_{j \in J \cup J'}$, un ensemble de variables réelles et x est le vecteur colonne $J \cup J' \times \{1\}$ de ces variables.*

Il s'énonce ainsi :

$$\texttt{Minimiser } c \cdot x \texttt{ tel que :}$$
$$\forall i \in I, A[i,-] \cdot x = b_i$$
$$\forall i \in I', A[i,-] \cdot x \geq b_i$$
$$\forall j \in J, x_j \geq 0$$

Nous définissons deux formes restreintes de problème PL.

Définition 17 (PL canonique, PL standard) *Soit un problème PL,*
— *Ce problème est un problème PL* canonique *si $I = \emptyset$ et $J' = \emptyset$. Autrement dit, il est constitué d'inégalités et toutes ses variables sont positives.*
— *Ce problème est un un problème PL* standard *si $I' = \emptyset$ et $J' = \emptyset$. Autrement dit, il est constitué d'égalités et toutes ses variables sont positives.*

La proposition suivante démontre que tout problème PL peut se transformer en un problème canonique ou standard équivalent.

Proposition 25 *Soit un problème PL général, alors il existe un problème PL canonique et un problème PL standard qui lui sont équivalents.*

Preuve
Transformation en problème canonique. Nous procédons en deux étapes. Nous substituons à toute égalité $A[i,-].x = b_i$ les deux inégalités $A[i,-].x \geq b_i$ et $-A[i,-].x \geq -b_i$. Pour chaque variable x_j avec $j \in J'$, nous ajoutons deux nouvelles variables x_{j^+},x_{j^-} telles que j^+,$j^- \in J$ et nous substituons $x_{j^+} - x_{j^-}$ à x_j dans toutes les équations.
Transformation en problème standard. Nous procédons également en deux étapes. Comme précédemment, pour chaque variable x_j avec $j \in J'$, nous ajoutons deux nouvelles variables x_{j^+},x_{j^-} telles que j^+,$j^- \in J$ et nous substituons $x_{j^+} - x_{j^-}$ à x_j dans toutes les équations. Puis nous ajoutons l'ensemble des variables $\{x_{i^s}\}_{i \in I'}$ avec $i^s \in J$ et nous substituons à toute inégalité $A[i,-].x \geq b_i$, l'égalité $A[i,-].x - x_{i^s} = b_i$.

c.q.f.d. ◊◊◊

6.4 L'algorithme du simplexe

L'algorithme du simplexe résout les problèmes PL standard ce qui nous conduit aux notations suivantes. $Sol(A,b) = \{x \in \mathbb{R}^J \mid A \cdot x = b\}$, $Sol^+(A,b) = \{x \geq 0 \mid A \cdot x = b\}$. $I = \{1,\ldots,m\}$ et $J = \{1,\ldots,n\}$. Nous notons la fonction objectif $f(x) \equiv c \cdot x$.

Le problème PL standard noté (A,b,c) s'énonce alors :

$$\texttt{Minimiser } c \cdot x \texttt{ tel que } x \in Sol^+(A,b)$$

Remarquons qu'en prenant $c' = -c$, on transforme un problème de minimisation en un problème de maximisation et vice versa.

6.4.1 Schéma général de l'algorithme

L'algorithme procède en plusieurs étapes. Cependant la preuve de correction de l'algorithme ne suit pas les étapes de cet algorithme. Aussi nous présentons d'abord le schéma général de l'algorithme en nous appuyant sur des résultats que nous démontrerons dans un ordre différent de leur utilisation.

Première étape. Il s'agit de transformer le problème de façon à ce que $rang(A) = m$. Cette étape s'exécute en temps polynomial et est très similaire à l'algorithme d'élimination de Gauss. Comme l'indique la proposition suivante, il y a deux résultats possibles à l'issue de cette étape.

Proposition 26 *Etant donné un programme linéaire (A,b,c) alors :*
- *Soit on peut détecter que ce problème n'a pas de solution.*
- *Soit on peut le transformer en un problème (A',b',c) tel que $rang(A') = m'$.*

Illustration. La figure ci-dessous se lit ainsi. La première ligne décrit la contrainte est $x_1 + 3x_4 + 2x_5 = 1$ (idem pour les deux lignes suivantes). La dernière ligne décrit la fonction objectif $x_1 + x_2 + x_3 + x_4 + x_5 + 0$.

A	x_1	x_2	x_3	x_4	x_5	b
	1	0	0	3	2	1
	1	1	0	5	1	3
	1	1	1	4	4	6
c	1	1	1	1	1	0

Il est évident que le rang de A est 3. Imaginons que nous voulions nous en assurer en faisant apparaître une matrice identité 3×3. Sans changer le problème, nous soustrayons la première ligne aux deux autres ce qui nous donne :

A	x_1	x_2	x_3	x_4	x_5	b
	1	0	0	3	2	1
	0	1	0	2	-1	2
	0	1	1	1	2	5
c	1	1	1	1	1	0

Puis nous retirons la deuxième ligne à la troisième ligne.

A	x_1	x_2	x_3	x_4	x_5	b
	1	0	0	3	2	1
	0	1	0	2	-1	2
	0	0	1	-1	3	3
c	1	1	1	1	1	0

On supposera dans la suite que $rang(A) = m$. Par conséquent $n \geq m$. On considère maintenant une partition (dynamique) des indices de $J = B \uplus N$ avec $Card(B) = m$. On note A_B (resp. A_N) la sous-matrice de A constituée des colonnes d'indice appartenant à B (resp. N). On fait de même avec les vecteurs c et x.

Deuxième étape. Il s'agit de trouver une solution initiale au problème et plus précisément une *base* initiale.

Définition 18 (Base d'un problème PL) *Une* base B *est un sous-ensemble de m indices de J telle que A_B est inversible et $A_B^{-1} \cdot b$ est positif ou nul. La solution associée à la base B est définie par :*

$$\forall j \in B, x_j = (A_B^{-1} \cdot b)_j \text{ et } \forall j \notin B, x_j = 0$$

Illustration. Dans notre exemple $B = \{1,2,3\}$ constitue bien une base. En effet, nous avons « composé » notre problème par A_B^{-1} et le vecteur $A_B^{-1} \cdot b$ se trouve sur la dernière colonne. Comme il est positif cela nous fournit la solution initiale $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, $x_4 = x_5 = 0$ et $f(x) = 6$.

Le principe de cette recherche est de construire un autre problème qui dispose d'une base initiale évidente et dont le résultat soit indique que le problème original n'a pas de solution soit permet de retrouver une base initiale.

Proposition 27 *Etant donné un programme linéaire (A,b,c) alors*
- *Soit on peut détecter que ce problème n'a pas de solution.*
- *Soit on peut trouver une base B.*

Troisième étape. Il s'agit d'améliorer la solution courante en remplaçant un indice de la base par un indice hors base pour obtenir une nouvelle base dont la solution est au moins aussi *bonne* que la solution précédente. La proposition suivant nous assure que cela est toujours possible à moins que : soit la solution courant soit optimale, soit on détecte qu'il n'existe pas de solution optimale.

Proposition 28 *Etant donné un programme linéaire (A,b,c) et B une base alors :*
- *Soit on peut détecter que la solution de B est optimale.*
- *Soit on peut détecter que ce problème n'est pas minoré.*
- *Soit on peut trouver une base B' avec $B' \neq B, B' = B \setminus \{i\} \cup \{j\}$ pour un certain couple (i,j) telle que $c_{B'} \cdot A_{B'}^{-1} \cdot b \leq c_B \cdot A_B^{-1} \cdot b$, l'égalité n'étant réalisée que si les solutions associées sont identiques.*

Nous dirons que B' est obtenue par « pivotage » à partir de B.

Illustration. Afin de décrire le pivotage, nous allons exprimer notre fonction objectif en tenant compte de la base. Chacune des trois contraintes permet d'exprimer une variable de la base en fonction des variables hors base. Par exemple, $x_1 = 1 - 3x_4 - 2x_5$. On peut substituer ces expressions dans la fonction objectif et nous obtenons $f(x) = 6 - 3x_4 - 3x_5$ (valide pour tout $x \in Sol(A,b)$). Le tableau se réécrit ainsi.

A	x_1	x_2	x_3	x_4	x_5	b
	1	0	0	3	2	1
	0	1	0	2	-1	2
	0	0	1	-1	3	3
c	0	0	0	-3	-3	6

On s'aperçoit alors que si on augmente x_4 ou x_5 tout en maintenant la positivité de la solution, on diminue la fonction objectif. Choisissons d'augmenter x_5. Puisque $x_1 = 1 - 3x_4 - 2x_5$, x_5 ne peut dépasser $1/2$; puisque $x_2 = 2 - 2x_4 + x_5$ cette équation ne contraint pas l'augmentation de x_5 ; puisque $x_3 = 3 + x_4 - 3x_5$ x_5 ne peut dépasser 1. La variable qui sortira de la base est donc x_1. Le nouveau tableau est obtenu en réécrivant la première ligne $x_5 + 3/2x_4 + 1/2x_1 = 1/2$ et en remplaçant dans les deux autres lignes et dans la fonction objectif x_5 par $1/2 - 3/2x_4 - 1/2x_1$. Ce qui nous donne :

A	x_5	x_2	x_3	x_4	x_1	b
	1	0	0	$\frac{3}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
	0	1	0	$\frac{7}{2}$	$\frac{1}{2}$	$\frac{5}{2}$
	0	0	1	$-\frac{11}{2}$	$-\frac{3}{2}$	$\frac{3}{2}$
c	0	0	0	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{9}{2}$

Nous constatons que la fonction objectif s'écrit $f(x) = 9/2 + 3/2x_4 + 3/2x_5$. Autrement dit, la nouvelle base fournit une solution optimale.

Quatrième étape. L'algorithme reposant sur la propriété précédente ne se termine pas si et seulement si il rencontre deux fois une même base. Cependant avec une stratégie appropriée d'échanges d'indices, ce cas ne peut arriver.

Proposition 29 *Il existe une stratégie de pivotage telle que pour toute suite de bases $B_1, \ldots, B_K$ vérifiant que $\forall 1 \leq k < K, B_{k+1}$ est obtenue par pivotage de B_k, on a $B_K \neq B_1$.*

6.4.2 Preuve de correction de l'algorithme

Il nous suffit pour cela d'établir les propositions de la section précédente.

Preuve de la proposition 26 (*relative au rang de A*)
Supposons que le rang de la matrice A noté r vérifie $r < m$.
Soit I' un sous-ensemble d'indices de r vecteurs lignes indépendants de A.
Soit maintenant $i \notin I'$, alors $\exists \{\lambda_{i'}\}_{i' \in I'}$, tels que $A[i,-] = \sum_{i' \in I'} \lambda_{i'} A[i',-]$. Il y a alors deux cas à considérer :

— Si $b_i = \sum_{i' \in I'} \lambda_{i'} b_{i'}$ alors la contrainte $A[i,-] \cdot x = b_i$ est redondante par rapport à celles spécifiées par I' et elle peut être omise.
— Sinon le problème n'admet pas de solution.

Dans le cas où la deuxième branche de l'alternative n'est jamais rencontrée, on restreint la matrice A aux lignes de I' de cardinal $r = rang(A)$ avec équivalence des problèmes.

c.q.f.d. ◊◊◊

Preuve de la proposition 28 (*relative au pivotage*)
On note :
— $b_{BN} = A_B^{-1} \cdot b$ vecteur indicé par B,
— I_B la matrice identité indicée par $B \times B$
— $A_{BN} = A_B^{-1} \cdot A_N$ matrice indicée par $B \times N$

Nous observons que :
— La solution courante associée à B est définie par $\forall j \in B, x_j = b_{BN}[j]$ et $\forall j \in N, x_j = 0$. On la note s_B.
— Puisque $A_B \cdot A_{BN} = A_N$, $\forall k \in N, A[-,k] = \sum_{j \in B} A_{BN}[j,k] \cdot A[-,j]$.

L'équation des contraintes peut se réécrire $A_B \cdot x_B + A_N \cdot x_N = b$ ce qui est équivalent à $x_B + A_B^{-1} \cdot A_N \cdot x_N = A_B^{-1} \cdot b$ qui se réécrit finalement

$$I_B \cdot x_B + A_{BN} \cdot x_N = b_{BN} \tag{6.1}$$

L'équation 6.1 signifie que pour une valeur de x_N (resp. $x_N \geq 0$), il y a exactement (resp. au plus) une valeur de x_B (resp. $x_B \geq 0$) telle que $x \in Sol(A,b)$ (resp. $x \in Sol^+(A,b)$).

Intéressons-nous maintenant à la fonction objectif $f(x) = c \cdot x$.
Lorsque $x \in Sol(A,b)$, nous pouvons la réécrire :
$f(x) = c_N \cdot x_N + c_B \cdot x_B = c_N \cdot x_N + c_B \cdot (A_B^{-1} \cdot b - A_B^{-1} \cdot A_N \cdot x_N)$
$= c_B \cdot A_B^{-1} \cdot b + (c_N - c_B \cdot A_B^{-1} \cdot A_N) \cdot x_N$
On note $f(B) = c_B \cdot b_{BN}$ et $c_{BN} = c_N - c_B \cdot A_{BN}$. Alors la fonction objectif se réécrit :

$$\forall x \in Sol(A,b), \quad f(x) = f(B) + c_{BN} \cdot x_N \tag{6.2}$$

Cas n°1. Si maintenant $c_{BN} \geq 0$, alors pour tout $x \in Sol(A,b)$ tel que $x_N \geq 0$ $f(x) \geq f(B)$. **Par conséquent, le minimum est atteint pour $x = s_B$ et a pour valeur $c_B \cdot b_B$.**

Cas n°2. Sinon ceci signifie qu'il existe un $j \in N$ tel que $c_{BN}[j] < 0$.
On note $N_e^B = \{j \in N \mid c_{BN}[j] < 0\}$ et $j_e^B = \max(N_e^B)$ (d'autres stratégies de choix sont possibles).

Soit maintenant x_θ défini pour $\theta \geq 0$ par :
— $\forall j \in B, x_\theta[j] = b_{BN}[j] - \theta A_{BN}[j,j_e^B]$
— $\forall j \in N \setminus \{j_e^B\}, x_\theta[j] = 0$
— $x_\theta[j_e^B] = \theta$

En vertu de l'équation 6.1, $x_\theta \in Sol(A,b)$. Par conséquent, $f(x_\theta) = f(B) + \theta c_{BN}[j_e^B]$. Soit maintenant $B_c^B = \{j \in B \mid A_{BN}[j,j_e^B] > 0\}$.

Cas n°2.1. Si $B_c^B = \emptyset$ alors $\forall \theta \in R^+, x_\theta \geq 0$ et puisque $\lim_{\theta \to \infty} f(x_\theta) = -\infty$, **il n'y a donc pas de solution qui minimise la fonction de coût f.**

Cas n°2.2. Sinon on note $B_s^B = \{j \in B_c^B \mid \forall k \in B_c^B, \frac{b_{BN}[j]}{A_{BN}[j,j_e^{(B)}]} \leq \frac{b_{BN}[k]}{A_{BN}[k,j_e^{(B)}]}\}$. B_s^B est l'ensemble des indices de la base B tels que $x_\theta[j]$ atteint en premier zéro lorsque θ croît. On pose $j_s^B = \max(B_s^B)$ (d'autres stratégies de choix sont possibles) et $\theta^B = \frac{b_{BN}[j_s^B]}{A_{BN}[j_s^B,j_e^B]}$.

Par construction, $x_{\theta^B} \geq 0$ et $x_{\theta^B}[j_s^B] = 0$. Le nouvel ensemble d'indices est $B' = B \setminus \{j_s^B\} \cup \{j_e^B\}$. Rappelons que $A[-,j_e^B] = \sum_{j \in B} A_{BN}[j,j_e^B]A(-,j)$. Puisque $A_{BN}[j_s^B,j_e^B] \neq 0$, les vecteurs colonnes de A indicés par B' sont bien indépendants.

Puisque $f(B') = f(B) + \theta^B A_{BN}[j_s^B,j_e^B]$, on en déduit que $f(B') \leq f(B)$ **avec inégalité stricte si** $\theta^B > 0$.

La preuve est terminée mais nous indiquons comment calculer $A_{B'N'}$ et $b_{B'N'}$ et $c_{B'N'}$ car ces calculs sont repris dans l'algorithme 40.
$A[-,j_s^B] = -\sum_{j \in B, j \neq j_s^B} \frac{A_{BN}[j,j_e^B]}{A_{BN}[j_s^B,j_e^B]} A[-,j] + \frac{1}{A_{BN}[j_s^B,j_e^B]} A[-,j_e^B]$.
Ce qu'on peut aussi écrire :
$A[-,j_s^B] = \sum_{j \in B'} A_{B'N'}[j,j_s^B]A[-,j]$
Il reste à compléter le calcul de $A_{B'N'}$:
$\forall k \in N \setminus \{j_s^B\}$,
$A[-,k]$
$= \sum_{j \in B} A_{BN}[j,k]A[-,j]$
$= \sum_{j \in B, j \neq j_s^B} A_{BN}[j,k]A[-,j] + A_{BN}[j_s^B,k](\sum_{j \in B'} A_{B'N'}[j,j_s^B]A[-,j])$
$= \sum_{j \in B', j \neq j_e^B} (A_{BN}[j,k] + A_{BN}[j_s^B,k]A_{B'N'}[j,j_s^B])A[-,j] + A_{BN}[j_s^B,k]A_{B'N'}[j_e^B,j_s^B]A[-,j_e^B]$
Par construction $b_{B'N'} = x_{\theta^B}$.

Enfin, il faut calculer $c_{B'N'}$. Pour $x \in Sol(A,b)$, on a :
$x[j_e^B] = b_{B'N'}[j_e^B] - \sum_{j \in N'} A_{B'N'}[j_e^B,j]x[j]$
Par conséquent,
$f(x) = f(B) + \sum_{j \in N} c_{BN}[j]x[j]$
$= f(B) + \sum_{j \in N \setminus \{j_e^B\}} c_{BN}[j]x[j] + c_{BN}[j_e^B](b_{B'}[j_e^B] - \sum_{j \in N'} A_{B'N'}[j_e^B,j]x[j])$
$= f(B') + \sum_{j \in N \setminus \{j_e^B\}} (c_{BN}[j] - c_{BN}[j_e^B]A_{B'N'}[j_e^B,j])x[j] - A_{B'N'}[j_e^B,j_s^B]x[j_s^B]$

c.q.f.d. ◊◊◊

Preuve de la proposition 29 (*relative à l'occurrence des bases*)
Nous appliquons *la règle de Bland* qui consiste à choisir le plus petit indice entrant et le plus petit indice sortant. Nous démontrons cette proposition par l'absurde. Supposons que l'algorithme construise une suite de bases $B_1, \ldots B_K$ avec $B_1 = B_K$.

Si nous supprimons tous les indices appartenant à $\bigcap_{1 \leq k < K} B_k$ ainsi que les vecteurs lignes correspondants dans les matrices $A_{B_k N_k}$ et les composantes correspondantes dans les vecteurs $b_{B_k N_k}$ alors l'algorithme se comporte de la même façon puisque ces indices ne sont jamais sélectionnés pour sortir de la base.

Si nous supprimons tous les indices appartenant à $\bigcap_{1 \leq k < K} N_k$ ainsi que les vecteurs colonnes correspondants dans les matrices $A_{B_k N_k}$ et les composantes correspondantes dans les vecteurs $c_{B_k N_k}$ alors l'algorithme se comporte de la même façon puisque ces indices ne sont jamais sélectionnés pour entrer dans la base.

Nous pouvons donc supposer que tout indice entre et sort de la base au moins une fois durant la suite de « pivotages ». Par conséquent puisque la solution courante reste identique durant tout le processus, on a $\forall i,k, b_{B_k N_k}[i] = 0$ (sinon $\theta_{B_k} > 0$ pour au moins une base B_k). Autrement dit la solution courante est le vecteur nul et la fonction objectif est nulle pour cette solution.

Soit maintenant *im* l'indice maximum restant, B_k une base telle que $im \in N_k \cap B_{k+1}$ (*i.e. im* entre dans la base B_{k+1}) et $B_{k'}$ une base telle que $im \in B_{k'} \cap N_{k'+1}$ (*i.e. im* sort de la base $B_{k'}$). On note *in* l'indice qui entre dans la base $B_{k'+1}$.

Définissons le vecteur y par :

— $y[in] = 1$
— $\forall i \in B_{k'}$, $y[i] = -A_{B_{k'} N_{k'}}[i,in]$
— $\forall i \in N_{k'} \setminus \{in\}$, $y[i] = 0$

Par construction, $y \in Sol(A,b)$ et par conséquent $f(y) = f(0) + c_{B_{k'} N_{k'}}[in] = c_{B_{k'} N_{k'}}[in] < 0$ puisque *in* entre dans la base. D'autre part $A_{B_{k'} N_{k'}}[im,in] > 0$ puisque *im* sort de la base et $\forall i \in B_{k'} \setminus \{im\}, A_{B_{k'} N_{k'}}[i,in] \leq 0$ d'après la règle de choix de l'indice sortant. Autrement dit, $y[i] < 0 \Leftrightarrow i = im$.

Puisque *im* entre dans B_{k+1}, $c_{B_k N_k}[im] < 0$. D'après la règle de choix de l'indice entrant $\forall i \in N_k \setminus \{im\}, c_{B_k N_k}[i] \geq 0$. Evaluons $f(y)$ en fonction de la base B_k.
$f(y) = \sum_{i \in N_k} c_{B_k N_k}[i] \cdot y[i]$
$= \sum_{i \in N_k \setminus \{im\}} c_{B_k N_k}[i] \cdot y[i] + c_{B_k N_k}[im] \cdot y[im] \geq c_{B_k N_k}[im] \cdot y[im] > 0.$
D'où une contradiction.

c.q.f.d. ◊◊◊

Le choix des indices entrants et sortants peut toujours se faire sans respecter la règle de Bland. Il suffit simplement de l'appliquer lorsque la solution ne progresse plus. Ceci est particulièrement intéressant si on veut appliquer des règles heuristiques de choix de ces indices. D'autre part il existe une autre stratégie qui permet de garantir la terminaison de l'algorithme du simplexe que nous expliquons maintenant. Cette stratégie repose sur l'ordre (total) lexicographique entre vecteurs.

Définition 19 *Soient v et v' deux vecteurs de $\mathbb{R}^n$, v est dit plus petit lexicographiquement que v' (noté $v <_l v'$) ssi $\exists 1 \leq i \leq n, v[i] < v'[i] \wedge \forall 1 \leq j < i, v[j] = v'[j]$. On note également $v \leq_l v'$ ssi $v <_l v' \vee v = v'$.*

On suppose que J est muni d'un ordre total sur les indices et ne contient pas 0 considéré comme plus petit que tout indice de J. On note bA la matrice $I \times \{0\} \cup J$ définie par $bA[-,0] = b$ et $\forall j \in J, bA[-,j] = A[-,j]$. De manière similaire, soit B une base, on note $BA_{BN} = A_B^{-1} \cdot bA$. On note aussi c^+_{BN} le vecteur ligne indicé par $\{0\} \cup J$ tel que $c^+_{BN}[0] = -f(B)$ et $\forall j \in J, c^+_{BN}[j] = c_{BN}[j]$. L'intérêt de cette écriture est de mettre en évidence la stratégie que l'on veut adopter et dont la correction est assurée par le lemme suivant.

b	A		x_1	x_2	x_3	x_4	x_5
1			1	0	0	3	2
2			0	1	0	2	-1
3			0	0	1	-1	3
-6	c		0	0	0	-3	-3

Lemme 14 *Soit une base B, soit j_e un indice entrant quelconque (i.e. tel que $c_{BN}[j_e] < 0$), alors :*
Si $j_s \in B$ est défini par $bA_{BN}[j_s,j_e] > 0 \wedge$
$\forall j \in B, j \neq j_s, bA_{BN}[j,j_e] > 0 \Rightarrow \frac{1}{bA_{BN}[j_s,j_e]} bA_{BN}[j_s,-] <_l \frac{1}{bA_{BN}[j,j_e]} bA_{BN}[j,-]$
alors $B' = B \setminus \{j_s\} \cup \{j_e\}$ est bien une base.
Si de plus $\forall j \in B, bA_{BN}[j,-] >_l 0$
alors $\forall j \in B', bA_{B'N'}[j,-] >_l 0$ et $c^+_{B'N'} >_l c^+_{BN}$.

Preuve
Le fait que j_s soit unique résulte du fait que $rang(A) = m$.
La première partie du lemme est évidente car en examinant l'indice 0 de la matrice bA, nécessairement j_s est choisi parmi les j qui minimisent $\frac{b_{BN}[j]}{A_{BN}[j,j_s]}$ sachant que $A_{BN}[j,j_s] > 0$.
Supposons l'hypothèse complémentaire vérifiée et examinons maintenant les lignes de la matrices $bA_{B'N'}$.
$bA_{B'N'}[j_e,-] = \frac{1}{bA_{BN}[j_s,j_e]} bA_{BN}[j_s,-] >_l 0$.
Soit $j \in B \setminus \{j_s\}$.
$bA_{B'N'}[j,-] = bA_{BN}[j,-] - \frac{bA_{BN}[j,j_e]}{bA_{BN}[j_s,j_e]} bA_{BN}[j_s,-]$
Il y a deux cas à examiner.
Cas n° 1. $A_{BN}[j,j_e] > 0$. Alors on réécrit
$bA_{B'N'}[j,-] = bA_{BN}[j,j_e](\frac{1}{bA_{BN}[j,j_e]} bA_{BN}[j,-] - \frac{1}{bA_{BN}[j_s,j_e]} bA_{BN}[j_s,-]) >_l 0$
d'après le choix de l'ordre lexicographique.
Cas n° 2. $A_{BN}[j,j_e] \leq 0$. Alors $bA_{B'N'}[j,-] \geq_l bA_{BN}[j,-] >_l 0$ puisque l'ordre lexicographique est « compatible » avec l'addition : $v >_l 0 \Rightarrow v + v' >_l v'$.
Examinons maintenant $c^+_{B'N'}$. On a :
$c^+_{B'N'} = c^+_{BN} - \frac{c^+_{BN}[j_e]}{bA_{BN}[j_s,j_e]} bA_{BN}[j_s,-] >_l c^+_{BN}$
pour la même raison que celle qui nous a fourni l'inéquation précédente.

c.q.f.d. ◊◊◊

Deuxième preuve de la proposition 29 (*relative à l'occurrence des bases*)
En appliquant la stratégie du lemme précédent, il est impossible d'obtenir deux fois la même base puisque le vecteur $c^+_{B'N'}$ croît strictement lexicographiquement. Pour assurer la condition initiale, il suffit d'ordonner J de telle façon que si B_0 est la base initiale alors $\forall j \in B_0, \forall j' \in N_0, j < j'$. Ainsi sur chaque ligne, on a un coefficient non négatif suivi éventuellement de quelques 0 puis d'un 1.

c.q.f.d. ◊◊◊

Une fois établies les trois propositions précédentes, nous disposons d'un algorithme pour le cas particulier où une base initiale est connue.

Preuve de la proposition 27 (*relative à la base initiale*)
Tout d'abord nous supposons que b est un vecteur positif ou nul. En effet, pour chaque composante négative, il suffit de multiplier par -1 cette composante et le vecteur ligne de A correspondant sans changer le problème. Nous augmentons alors le problème avec m variables $\{x_{n+1}, \ldots, x_{n+m}\}$ et la matrice A par $\forall 1 \leq i \leq m, A[i,n+i] = 1 \wedge \forall n+1 \leq j \neq n+i \leq n+m, A[i,j] = 0$.

La base initiale est $B = \{n+1, \ldots, n+m\}$ (la sous-matrice associée est l'identité), $b_{BN} = b$. On minimise la fonction objectif $\sum_{n<i\leq n+m} x_i$. Autrement dit, $\forall 1 \leq j \leq n, c_{BN}[i] = -\sum_{1\leq i\leq m} A[i,j]$.
Puisque la fonction objectif est minorée par 0, lorsque l'algorithme se termine, il renvoie une solution optimale.

Cas n°1. Si la fonction objectif est non nulle, alors le problème initial n'admet pas de solution.

Cas n°2. Dans le cas contraire, la base correspondante inclut peut-être des indices des variables auxiliaires (qui sont nulles). La matrice $(Id \mid A_{BN})$ restreinte aux composantes des variables originales est la transformée par A_B^{-1} de la matrice originale (nous la notons A'). A' est de rang m. Par conséquent, soit i l'indice d'une variable auxiliaire dans la base B, alors il existe au moins un indice j d'une variable originale dans N tel que $A_{BN}[i,j] \neq 0$. Sinon le vecteur ligne d'indice i de A' est nul et $rang(A') < m$. On pivote i et j sans tenir compte de la fonction objectif puisque $x_i = x_j = 0$. En itérant cette opération, on obtient une base composée uniquement des indices de variables originales et on détruit les vecteurs colonnes des variables auxiliaires.

c.q.f.d. ◊◊◊

6.4.3 L'algorithme détaillé

L'algorithme 37 élimine les contraintes redondantes afin d'assurer que le rang de la matrice soit égal au nombre de contraintes. La variable *newm* fournit le nombre de contraintes finales (éventuellement 0 si la matrice A est nulle). La boucle principale examine la ligne d'indice i. Au début de cet examen, *newm* lignes ont été conservées et déplacées en première position et les *newm* premières colonnes constituent une matrice triangulaire supérieure de diagonale sans éléments nuls. Si la ligne d'indice i est non nulle, elle est conservée car l'indice j de son composant non nul est forcément compris entre $newm + 1$ et n augmentant ainsi le rang de la sous-matrice. On incrémente *newm* et on inverse les colonnes d'indice j et *newm* (et les composants correspondants du vecteur c) pour faire apparaître le composant non nul en colonne *newm*. On combine linéairement la ligne i avec les lignes $i + 1, \ldots, m$ afin de faire apparaître des zéros sous ce composant. Ces combinaisons linéaires s'appliquent aussi au vecteur b. Puis on déplace la ligne (et le composant b_i) en position *newm*. Si la ligne i est nulle alors soit b_i est nul et la ligne est redondante ; soit b_i est non nul et le problème n'admet pas de solution. Le tableau *ind* fournit l'indice de colonne en fonction de l'indice de la variable.

L'algorithme 38 résout le problème lorsqu'il dispose d'une base initiale. Il itère un pivotage à l'aide de deux algorithmes : `Sélectionner` qui fournit les indices entrants et sortants (si un pivotage est possible) et `Permuter` qui pratique le pivotage en fonction des indices fournis. L'itération s'arrête si `Sélectionner` découvre que la base est optimale ou que le problème est non minoré.

Dans l'algorithme 39, le tableau *ind* ordonne les indices des variables de telle sorte qu'un indice j appartient à la base ssi $ind[j] \leq m$. La matrice A de cet algorithme correspond à la matrice A_{BN}, et le vecteur c à c_{BN}. Cette matrice et ce vecteur ont $n - m$ colonnes ; une variable hors base d'indice j y apparaît à la colonne d'indice $ind[j] - m$. Le vecteur b correspond à b_{BN}. La matrice A et le vecteur b ont m lignes ; une variable de base d'indice j y

Algorithme 37: Eliminer les contraintes redondantes

`Eliminer`(A,b,c,ind,m,n) : statut
Input : n, le nombre de variables

InputOutput : m, le nombre de contraintes éventuellement diminué
InputOutput : A, une matrice $m \times n$
InputOutput : b, un vecteur de dimension m
InputOutput : c, un vecteur de dimension n
InputOutput : ind, un tableau $1..n$ repérant la place des variables

Output : le statut de l'opération

Data : *statut* une variable de type statut, i,j,k,$newm$,$itemp$ des indices
Data : $temp$,$pivot$ des valeurs, *trouve* un booléen

$newm \leftarrow 0$
for $i \leftarrow 1$ **to** n **do** $invind[i] \leftarrow i$
for $i \leftarrow 1$ **to** m **do** *(ligne i conservée si indépendante des précédentes)*
  $j \leftarrow newm + 1$; $trouve \leftarrow$ **false**
  repeat
    $trouve \leftarrow A[i,j] \neq 0$
    if not *trouve* **then** $j \leftarrow j + 1$
  until $j > n$ **or** *trouve*
  if *trouve* **then**
    $newm \leftarrow newm + 1$
    for $k \leftarrow 1$ **to** m **do**
      $temp \leftarrow A[k,newm]$; $A[k,newm] \leftarrow A[k,j]$; $A[k,j] \leftarrow temp$
    end
    $temp \leftarrow c[newm]$; $c[newm] \leftarrow c[j]$; $c[j] \leftarrow temp$
    $itemp \leftarrow invind[newm]$; $invind[newm] \leftarrow invind[j]$;
    $invind[j] \leftarrow itemp$
    for $j \leftarrow i + 1$ **to** m **do**
      $pivot \leftarrow A[j,newm]/A[i,newm]$; $A[j,newm] \leftarrow 0$
      for $k \leftarrow newm + 1$ **to** n **do**
        $A[j,k] \leftarrow A[j,k] - pivot \cdot A[i,k]$
      end
      $b[j] \leftarrow b[j] - pivot \cdot b[i]$
    end
    for $j \leftarrow 1$ **to** n **do** $A[newm,j] \leftarrow A[i,j]$
    $b[newm] \leftarrow b[i]$
  else if $b[i] \neq 0$ **then return** `vide`
end
if $newm = 0$ **then return** `matricenulle`
for $i \leftarrow 1$ **to** n **do** $ind[invind[i]] \leftarrow i$
$m \leftarrow newm$; **return** `rangm`

Algorithme 38: Minimiser avec une solution initiale

`MinimiserInit`(A,b,c,ind,m,n) : statut
Input : n, le nombre de variables
Input : m, le nombre de contraintes

InputOutput : A, une matrice $m \times n - m$
InputOutput : b, un vecteur de dimension m
InputOutput : c, un vecteur de dimension $n - m$
InputOutput : ind, un tableau $1..n$ repérant la place des variables
Output : le statut de l'opération

Data : *statut* une variable de type statut, i,j des indices

statut,i,$j \leftarrow$ `Sélectionner`(A,b,c,ind,m,n)
while *statut* = `sélectionné` **do**
  `Permuter`(A,b,c,ind,m,n,i,j)
  statut,i,$j \leftarrow$ `Sélectionner`(A,b,c,ind,m,n)
end
return *statut*

apparaît à la ligne d'indice $ind[j]$. La sélection des indices à permuter se fait suivant la règle de Bland. La recherche de l'indice entrant se fait en repérant les indices hors base ($ind[j] > m$) dont la composante c_{BN} est strictement négative ($c[ind[j] - m] < 0$). Le premier indice trouvé est l'indice entrant. Si aucun indice n'est trouvé alors la solution est optimale. Le recherche de l'indice sortant se fait parmi ceux qui contraignent la diminution de la solution ($A[ind[i],jN] > 0$) et parmi ceux-là ceux qui la contraignent le plus ($\frac{b[ind[i]]}{A[ind[i],jN]}$ minimal). Le plus petit indice parmi les plus contraignants est renvoyé. Si un tel indice n'est pas trouvé alors le problème n'est pas minoré.

L'algorithme 40 utilise les mêmes variables que l'algorithme précédent. Le « pivotage » se fait en deux étapes. Il met d'abord à jour la colonne qui va correspondre à l'indice sortant. Pour ce faire, il doit multiplier l'ancien coefficient par $-\frac{1}{A[iB,jN]}$ sur toutes les lignes différentes de l'indice sortant (y compris pour le coefficient de c). Le coefficient sur la ligne de l'indice sortant est égal à $\frac{1}{A[iB,jN]}$ et le coefficient sur b correspond à la nouvelle valeur de la variable entrante $\frac{b[iB]}{A[iB,jN]}$. Une fois ceci fait, il met à jour les colonnes en tenant compte du fait que la colonne de l'indice sortant a déjà été mise à jour ce qui facilite les calculs. Enfin les indices sont intervertis.

L'algorithme 41 consiste à trouver si possible une base initiale. Il rend positifs les coefficients de b négatifs. En raison de l'insertion des variables auxiliaires, la matrice A correspond exactement à la matrice A_{BN}. Le vecteur c est choisi de manière appropriée. *Indbis* est le tableau de position des indices en tenant compte de l'insertion des variables auxiliaires. L'algorithme appelle ensuite `MinimiserInit` pour trouver une solution optimale au problème intermédiaire. Dans la boucle qui suit, il examine les variables auxiliaires. Si une variable auxiliaire est non nulle alors le problème n'a pas de solution. Si malgré tout elle est dans la base alors il la pivote avec une variable originale hors base. Enfin il élimine les colonnes superflues en décalant les colonnes restantes et mettant à

Algorithme 39: Sélectionner les indices à permuter

`Sélectionner`(A,b,c,ind,m,n) : statut,entier,entier
Input : n, le nombre de variables
Input : m, le nombre de contraintes
InputOutput : A, une matrice $m \times n - m$
InputOutput : b, un vecteur de dimension m
InputOutput : c, un vecteur de dimension $n - m$
InputOutput : ind, un tableau $1..n$ repérant la place des variables
Output : le statut de l'opération, les indices sortants et entrants
Data : i,j,is,je,iB,jN des indices, $pivot$,$inverse$ des rationnels

$je \leftarrow 0$; $j \leftarrow 0$
while $j < n$ **and** $je = 0$ **do** *(Recherche de l'indice entrant)*
  $j \leftarrow j + 1$
  if $ind[j] > m$ **then if** $c[ind[j] - m] < 0$ **then** $je \leftarrow j$
end
if $je = 0$ **then return** `optimal`,0,0
$jN \leftarrow ind[j] - m$
$is \leftarrow 0$; $pivot \leftarrow \infty$
for $i \leftarrow 1$ **to** n **do** *(Recherche de l'indice sortant)*
  if $ind[i] \leq m$ **then**
    if $A[ind[i],jN] > 0$ **then**
      if $\frac{b[ind[i]]}{A[ind[i],jN]} < pivot$ **then**
        $is \leftarrow i$; $iB \leftarrow ind[i]$
        $inverse \leftarrow \frac{1}{A[iB,jN]}$
        $pivot \leftarrow b[iB] \cdot inverse$
      end
    end
  end
end
if $is = 0$ **then return** `nonminoré`,0,0
return `sélectionné`,is,je

Algorithme 40: Permuter

`Permuter`(A,b,c,ind,m,n,is,je)
Input : n, le nombre de variables
Input : m, le nombre de contraintes
Input : is,je, les indices entrants et sortants

InputOutput : A, une matrice $m \times n - m$
InputOutput : b, un vecteur de dimension m
InputOutput : c, un vecteur de dimension $n - m$
InputOutput : ind, un tableau $1..n$ repérant la place des variables

Data : i,j,iB,jN des indices, $pivot$,$inverse$ des rationnels

$iB \leftarrow ind[is]$; $jN \leftarrow ind[je] - m$
$inverse \leftarrow \frac{1}{A[iB,jN]}$; $pivot \leftarrow b[iB] \cdot inverse$
for $i \leftarrow 1$ **to** m **do**
  if $i \neq iB$ **then**
    $b[i] \leftarrow b[i] - pivot \cdot A[i,jN]$
    $A[i,jN] \leftarrow -A[i,jN] \cdot inverse$
  end
end
$c[jN] \leftarrow -inverse \cdot c[jN]$
$b[iB] \leftarrow pivot$; $A[iB,jN] \leftarrow inverse$
for $j \leftarrow 1$ **to** $n - m$ **do**
  if $j \neq jN$ **then**
    for $i \leftarrow 1$ **to** m **do**
      if $i \neq iB$ **then**
        $A[i,j] \leftarrow A[i,j] + A[i,jN] \cdot A[iB,j]$
      end
    end
    $A[iB,j] \leftarrow inverse \cdot A[iB,j]$
  end
end
for $j \leftarrow 1$ **to** $n - m$ **do**
  if $j \neq jN$ **then** $c[j] \leftarrow c[j] - A[iB,j] \cdot c[jN]$
end
$ind[is] \leftarrow jN + m$; $ind[je] \leftarrow iB$;

jour les indices.

L'algorithme 42 qui implémente la méthode du simplexe enchaîne l'algorithme d'élimination de lignes, de recherche de base initiale et de recherche de solution optimale pour un problème initialisé. Le seul point délicat est la mise à jour des coefficients de la fonction objectif en tenant compte de la base initiale.

6.5 La dualité

De nombreuses méthodes alternatives reposent sur la dualité que nous présentons maintenant. Supposons que nous ayons une combinaison linéaire des vecteurs lignes de A, $d = \sum_{i \in I} y_i A[i,-]$ telle que $d \leq c$, alors pour tout $x \in Sol^+(A,b)$, $\sum_{i \in I} y_i b_i = d \cdot x \leq c \cdot x$. Autrement dit, $\sum_{i \in I} y_i b_i$ est un minorant de la valeur optimale. En recherchant le plus grand minorant possible par cette méthode, on obtient le problème *dual* suivant :

$$\texttt{Maximiser } y \cdot b \texttt{ tel que } y \in \mathbb{R}^I \wedge y \cdot A \leq c$$

Nous généralisons cette notion à un problème quelconque.

Définition 20 (Problème dual) *Soit* **P** *un problème PL appelé problème* primal *donné par les équation suivantes :*

$$\texttt{Minimiser } c \cdot x \texttt{ tel que :}$$

$$\forall i \in I, A[i,-] \cdot x = b_i$$
$$\forall i \in I', A[i,-] \cdot x \geq b_i$$
$$\forall j \in J, x_j \geq 0$$

Alors **D** *le problème* dual *de* **P** *est défini par les équations suivantes où y est un vecteur ligne de variables de support $I \cup I'$:*

$$\texttt{Maximiser } y \cdot b \texttt{ tel que :}$$

$$\forall j \in J, y \cdot A[-,j] \leq c_j$$
$$\forall j \in J', y \cdot A[-,j] = c_j$$
$$\forall i \in I', y_i \geq 0$$

Le lemme suivant généralise le raisonnement du début de la section.

Lemme 15 *Soit* **P** *un problème PL et* **D** *son dual. Soient x une solution (non nécessairement optimale) de* **P** *et y une solution (non nécessairement optimale) de* **D**. *Alors $y \cdot b \leq y \cdot A \cdot x \leq c \cdot x$.*

Preuve
$\forall j \in J, (y \cdot A[-,j])x_j \leq c_j x_j$ puisque x_j est positif et
$\forall j \in J', (y \cdot A[-,j])x_j = c_j x_j$.
D'où : $\sum_{j \in J \cup J'} (y \cdot A[-,j])x_j \leq \sum_{j \in J \cup J'} c_j x_j$
Autrement dit : $y \cdot A \cdot x \leq c \cdot x$
$\forall i \in I', y_i(A[i,-] \cdot x) \geq y_i b_i$ puisque y_i est positif et
$\forall i \in I, y_i(A[i,-] \cdot x) = y_i b_i$
D'où : $\sum_{i \in I \cup I'} y_i(A[i,-] \cdot x) \geq \sum_{i \in I \cup I'} y_i b_i$
Autrement dit : $y \cdot A \cdot x \geq y \cdot b$

Algorithme 41: Trouver une solution initiale

`Initialiser`(A,b,ind,m,n) : statut
Input : m, le nombre de contraintes
Input : n, le nombre de variables

InputOutput : A, une matrice $m \times n$,
au retour seule la sous-matrice $m \times n - m$ est significative
InputOutput : b, un vecteur de dimension m
InputOutput : ind, un tableau $1..n$ repérant la place des variables

Output : le statut de l'opération

Data : i,j,k,$next$ des indices
Data : c un tableau de dimension n des coefficients d'accroissement
Data : $indbis$,$invind$, deux tableaux $1..n+m$
repérant la place des variables et vice versa

```
for i ← 1 to m do (inverser les b si nécessaire)
    if b[i] < 0 then
        b[i] ← −b[i] ; for j ← 1 to n do A[i,j] ← −A[i,j]
    end
end
for i ← 1 to n do (Initialisation de c)
    c[i] ← 0 ; for j ← 1 to m do c[i] ← c[i] − A[i,j]
end
// Initialisation de indbis
for i ← 1 to n do indbis[i] ← m + ind[i]
for i ← 1 to m do indbis[n + i] ← i
MinimiserInit(A,b,c,indbis,m,m + n)
for i ← n + 1 to n + m do (sortir les variables aux. de la base si possible)
    if indbis[i] ≤ m then
        j ← indbis[i] ; if b[j] > 0 then return vide
        trouve ← false ; k ← 0
        repeat
            k ← k + 1
            if indbis[k] > m and A[j,indbis[k]] ≠ 0 then
                Permuter(A,b,c,indbis,m,m + n,i,k)
                trouve ← true
            end
        until trouve = true
    end
end
// Initialisation de invind
for i ← 1 to n + m do invind[indbis[i]] ← i
for i ← 1 to m do ind[invind[i]] ← i
i ← m ; next ← 0
while next < n − m do (élimination des colonnes superflues)
    i ← i + 1
    if invind[i] ≤ n then
        next ← next + 1 ; ind[invind[i]] ← next + m
        for k ← 1 to m do A[k,next] ← A[k,i − m]
    end
end
return unebase
```

Algorithme 42: Le simplexe

`Simplexe`(A,b,c,m,n) : statut
Input : m le nombre initial de contraintes, n le nombre de variables
Input : A une matrice rationnelle $m \times n$
Input : b un vecteur rationnel de dimension m
Input : c un vecteur rationnel de dimension n

Output : le statut de l'opération

Data : ind un tableau $1..n$ repérant la place courante d'une variable
Data : cBN un vecteur rationnel $1..n-m$,
coefficients d'accroissement courant des variables hors base
Data : *statut* une variable de type statut, i un indice, *opt* une valeur

for $i \leftarrow 1$ **to** n **do** $ind[i] \leftarrow i$
statut $\leftarrow$ `Eliminer`(A,b,ind,m,n)
if *statut* $\neq$ `rangm` **then return** *statut*
statut $\leftarrow$ `Initialiser`(A,b,ind,m,n)
if *statut* $\neq$ `unebase` **then return** *statut*
for $i \leftarrow 1$ **to** n **do** *(maj des coefficients en fonction de la base trouvée)*
  if $ind[i] > m$ **then**
    $k \leftarrow ind[i] - m$; $cBN[k] \leftarrow c[i]$
    for $j \leftarrow 1$ **to** n **do**
      if $ind[j] \leq m$ **then** $cBN[k] \leftarrow cBN[k] - A[ind[j],k]c[j]$
    end
  end
end
statut $\leftarrow$ `MinimiserInit`(A,b,cBN,ind,m,n)
if *statut* $\neq$ `optimal` **then return** *statut*
print « `Solution optimale` »
$opt \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
  print « `Variable` », i
  if $ind[i] > m$ **then print** 0
  else print $b[ind[i]]$; $opt \leftarrow opt + c[i] \cdot b[ind[i]]$
end
print « `Valeur optimale` », *opt*
return `trouvé`

c.q.f.d. ◊◊◊

Le lemme suivant justifie le choix du mot « dual ».

Lemme 16 *Soit* **P** *un problème PL et* **D** *son dual. Si* **D** *est exprimé comme un problème de minimisation alors son dual est* **P**.

Preuve
Soit **D** réécrit en problème PL :

$$\texttt{Minimiser } -b^T \cdot y^T \texttt{ tel que :}$$

$$\forall j \in J, -A^T[j,-] \cdot y^T \geq -c_j$$
$$\forall j \in J', -A^T[j,-] \cdot y^T = -c_j$$
$$\forall i \in I', y_i \geq 0$$

Son dual s'écrit alors

$$\texttt{Maximiser } x^T \cdot (-c)^T \texttt{ tel que :}$$

$$\forall i \in I', x^T \cdot -A^T[-,i] \leq -b_i$$
$$\forall i \in I, x^T \cdot -A^T[-,i] = -b_i$$
$$\forall j \in J, x_j \geq 0$$

En prenant l'opposée et la transposée des trois premières lignes, on retrouve **P**.

c.q.f.d. ◊◊◊

La proposition suivante souligne l'intérêt du problème dual.

Proposition 30 *Les problèmes primal (noté* **P***) et dual (noté* **D***) vérifient les relations suivantes :*

- *Si* **P** *n'est pas minoré alors* **D** *n'admet pas de solution.*
- *Si* **D** *n'est pas majoré alors* **P** *n'admet pas de solution.*
- **P** *admet une solution optimale si et seulement si* **D** *admet une solution optimale. Dans ce cas, les valeurs optimales coïncident.*

Preuve

Les deux premières affirmations de la proposition découlent du lemme 15.
Sans perte de généralité, nous supposons le problème **P** exprimé sous forme standard. Supposons que **P** ait une solution optimale et soit B la base trouvée par l'algorithme du simplexe. Rappelons les différentes (in)équations vérifiées par cette base :

$$f(B) = c_B \cdot A_B^{-1} \cdot b \tag{6.3}$$

(cette équation est vérifiée pour toute base)

$$c_N - c_B \cdot A_B^{-1} \cdot A_N \geq 0 \tag{6.4}$$

(cette équation est vérifiée pour toute base optimale)

Posons $y = c_B \cdot A_B^{-1}$. Alors d'une part $c_B = y \cdot A_B$ et d'autre part, l'inéquation 6.4 se réécrit $c_N \geq y \cdot A_N$. D'où $c \geq y \cdot A$. Ceci montre que y est une solution de **D**. L'équation 6.3 se réécrit $f(B) = y \cdot b$. Puisque la valeur associée à y est la valeur optimale de **P**, un majorant des valeurs possibles de **D**, y est une solution optimale et les valeurs optimales des deux problèmes coïncident.

Pour établir la réciproque, il suffit d'observer que le problème dual du problème dual est le problème primal d'après le lemme 16 et appliquer le même raisonnement.

c.q.f.d. ◊◊◊

Autre preuve de la dualité. Cette preuve alternative repose sur un résultat dont nous présentons ici l'une des nombreuse variantes.

Lemme 17 (Farkas) *Soit C une matrice $K \times L$ et $k \in K$. Alors les deux propositions suivantes sont équivalentes.*

1. $\exists v \in \mathbb{R}^K \; v \cdot C = 0 \wedge v \geq 0 \wedge v[k] > 0$
2. $\nexists w \in \mathbb{R}^L \; C \cdot w \leq 0 \wedge (C \cdot w)[k] < 0$

Preuve

Supposons l'existence simultanée de v et w tels que décrits dans le lemme. Alors :

$$\forall k' \in K, v[k'](C \cdot w)[k'] \leq 0 \text{ et } v[k](C \cdot w)[k] < 0$$

Par conséquent $v \cdot C \cdot w < 0$, mais $v \cdot C \cdot w = (v \cdot C) \cdot w = 0$, d'où une contradiction.

Il reste à établir que l'un de ces deux vecteurs existe toujours. Nous le prouvons par récurrence sur $|K|$.

$|K| = 1$ **(i.e. $K = \{k\}$).** Alors soit C est la matrice nulle et v défini par $v[k] = 1$ convient. Si C n'est pas nulle alors $\exists l \in L, C[k,l] \neq 0$. Si $C[k,l] < 0$ alors w défini par $w[l] = 1$ et $w[l'] = 0$ pour tout $l' \neq l$ convient. Sinon le vecteur opposé convient.

$|K| = n + 1$ **et le lemme est vérifié pour** $|K| = n$**.** Nous essayons d'obtenir v ou w de deux façons différentes.

Première tentative. Soit k_1 un indice de ligne différent de k. Posons $K_1 = K \setminus \{k_1\}$ et C_1 la matrice obtenue à partir de C, en supprimant la ligne indicée par k_1. En utilisant l'hypothèse de récurrence :

- soit $\exists v_1 \in \mathbb{R}^{K_1}, v_1 \cdot C_1 = 0 \wedge v_1 \geq 0 \wedge v_1[k] > 0$. Dans ce cas, v défini par $\forall k' \in K_1, v(k') = v_1(k') \wedge v(k_1) = 0$ convient.
- soit $\exists w \in \mathbb{R}^L, C_1 \cdot w \leq 0 \wedge (C_1 \cdot w)[k] < 0$.
  - Si $(C \cdot w)[k_1] \leq 0$ alors w convient.
  - Il reste un cas défavorable : $(C \cdot w)[k_1] > 0$.

Deuxième tentative. Soit W le sous-espace vectoriel de $\mathbb{R}^K$ engendré par l'ensemble $\{C[-,l]\}_{l \in L}$. W peut être décrit par une équation linéaire $v \cdot D = 0$ où les colonnes de D forment une base du sous-espace orthogonal de W. On applique alors l'hypothèse de récurrence à D_1, la matrice obtenue à partir de D en supprimant la ligne $D[k_1,-]$. Alors :

- Soit $\exists v_1 \in \mathbb{R}^{K_1}, v_1 \cdot D_1 = 0 \wedge v_1 \geq 0 \wedge v_1[k] > 0$. Observons que v défini par $\forall k' \in K_1, v[k'] = -v_1[k'] \wedge v[k_1] = 0$ satisfait $v \cdot D = 0$. Par conséquent v est engendré par $\{C[-,l]\}_{l \in L}$, i.e. $v = \sum_{l \in L} w[l] C[-,l] = C \cdot w$ avec $w \in \mathbb{R}^L$. Ce vecteur w convient.

— Soit $\exists w, D_1 \cdot w \geq 0 \wedge (D_1 \cdot w)[k] > 0$. Définissons v par $v = D \cdot w$. Par construction $v \cdot C = 0$.
  — Si $v[k_1] \geq 0$ alors v convient.
  — Il reste un cas défavorable : $v[k_1] < 0$.

Supposons maintenant que les deux cas défavorables sont simultanément rencontrés. Ceci signifie que :

— $\exists w \in \mathbb{R}^L, \forall k' \notin \{k_1,k\}, (C \cdot w)[k'] \leq 0 \wedge (C \cdot w)[k] < 0 \wedge (C \cdot w)[k_1] > 0$ et
— $\exists v \in \mathbb{R}^K, v \cdot C = 0 \wedge \forall k' \notin \{k_1,k\}, v[k'] \geq 0 \wedge v[k] > 0 \wedge v[k_1] < 0$.

Calculons de deux façons $v \cdot C \cdot w$. D'une part, $v \cdot C \cdot w = (v \cdot C) \cdot w = 0$. D'autre part, $v \cdot (C \cdot w) = \sum_{k \notin \{k_1,k\}} v[k'](C \cdot w)[k'] + v[k_1](C \cdot w)[k_1] + v[k](C \cdot w)[k]$. Cette somme est constituée de termes négatifs dont les deux derniers sont strictement négatifs, d'où $v \cdot C \cdot w < 0$. Cette contradiction complète la preuve.

c.q.f.d. ◊◊◊

Deuxième preuve de la proposition 30

Sans perte de généralité, nous démontrons la dualité dans le cas du problème primal suivant :

$$\min(c \cdot x \mid A \cdot x \leq b \wedge x \in \mathbb{R}^J)$$

conduisant au problème dual :

$$\max(-b^T \cdot y \mid y \cdot A = -c^T \wedge y \in \mathbb{R}^I \wedge y \geq 0)$$

Soit x une solution optimale du problème primal.
Posons $I' = \{i \in I \mid A[i,-] \cdot x = b[i]\}$. Alors le système d'équations :

$$\forall i \in I' \; A[i,-] \cdot w \leq 0 \wedge c \cdot w < 0$$

n'admet pas de solutions. En effet, si un tel w existait alors pour un $\varepsilon > 0$ suffisamment petit, $x + \varepsilon w$ serait aussi une solution du problème primal avec $c \cdot x > c \cdot x + \varepsilon w$.

On applique alors le lemme de Farkas qui implique l'existence d'un vecteur positif v indicé par I' et d'un réel positif non nul r tel que :

$$\forall j \in J \; \sum_{i \in I'} v[i] \cdot A[i,j] + rc[j] = 0$$

On définit y un vecteur indicé par I ainsi :

$$\forall i \in I' \; y[i] = \frac{v[i]}{r} \wedge \forall i \in I \setminus I' \; y[i] = 0$$

D'après les équations vérifiées par v, y est une solution du problème dual. Calculons son coût :

$$-b^T \cdot y = -\sum_{i \in I'} y[i]b[i] = -\sum_{i \in I'} y[i](\sum_{j \in j} A[i,j]x[j]) = -y \cdot A \cdot x = c \cdot x$$

c.q.f.d. ◊◊◊

On peut donc résoudre le problème dual plutôt que le problème primal ou s'appuyer sur la dualité pour concevoir et valider des algorithmes plus sophistiqués tels que l'algorithme primal-dual qui améliore la solution courante par résolution d'un problème linéaire de plus petite taille.

6.6 Une méthode intérieure

Les méthodes intérieures procèdent différemment et tentent de suivre un chemin dans le polyèdre (appelé chemin central) dont l'extrémité est une solution optimale. Ces méthodes s'appuient sur des techniques mathématiques plus élaborées que celles du simplexe. Elles ont l'avantage théorique d'avoir une complexité en temps polynomial alors que le simplexe peut atteindre dans le pire des cas une complexité exponentielle.

6.6.1 La forme de Goldman-Tucker

Afin de présenter l'une des méthodes intérieures, nous introduisons une nouvelle expression d'un programme linéaire. Supposons que le problème soit fourni sous forme canonique. Il s'énonce ainsi :

$$(P) \quad \texttt{Calculer } \min\{c \cdot u \mid A \cdot u \geq b, u \geq 0\}$$

Hypothèses. Dans cette section, nous faisons les hypothèses suivantes : A n'a ni colonne nulle ni ligne nulle, c n'est pas le vecteur nul et A,b,c sont à coefficients entiers. Dans le cas d'une colonne j nulle, posons u_j la variable associée. Si $c_j \geq 0$ alors le problème peut être réduit par suppression de la variable sans rien changer au problème initial. Sinon soit le problème réduit admet au moins une solution et le problème initial est non minoré, soit les deux problèmes n'ont pas de solutions. Dans le cas d'une ligne i nulle, soit $b_i \leq 0$ et la contrainte $0 \geq b_i$ peut être supprimée, soit $b_i > 0$ et le problème n'a pas de solution. Si c est le vecteur nul alors on peut lui substituer la fonction de coût u_1 sans changer le problème excepté qu'avec le nouveau problème, si l'espace des solutions est non vide on trouvera une solution qui minimise sa première composante. La contrainte d'intégralité s'obtient en multipliant A et b (resp. c) par le ppcm des dénominateurs qui y apparaissent. Cette transformation implique une augmentation au plus quadratique de la taille du problème et ne change donc pas le caractère polynomial des méthodes intérieures.

Son problème dual s'énonce ainsi :

$$(D) \quad \texttt{Calculer } \max\{v \cdot b \mid v \cdot A \leq c, v \geq 0\}$$

D'après le théorème de dualité, ceci revient à trouver une solution au problème primal et une solution au problème dual telle que la valeur de la solution duale soit supérieure ou égale à la valeur de la solution primale (donc en réalité identique). En introduisant des variables auxiliaires, on obtient le système d'équations :

$$\begin{aligned} A \cdot u - z &= b, \quad & u \geq 0, z \geq 0 \\ A^T \cdot v + w &= c, \quad & v \geq 0, w \geq 0 \\ b^T \cdot v - c^T \cdot u - \rho &= 0, \quad & \rho \geq 0 \end{aligned}$$

Dans ce système d'équations, v et c sont considérés maintenant comme des vecteurs colonnes.

Observation. Ceci démontre que le problème de l'existence d'une solution a la même complexité de calcul que le problème de la recherche d'une solution optimale.

En « homogénéisant », on obtient le système sous la forme *Goldman-Tucker* :

$$\begin{aligned} A \cdot u - \tau b - z = 0, &\quad u \geq 0, z \geq 0 \\ -A^T \cdot v + \tau c - w = 0, &\quad v \geq 0, w \geq 0 \\ b^T \cdot v - c^T \cdot u - \rho = 0, &\quad \rho \geq 0, \tau \geq 0 \end{aligned}$$

Observations.

- Une solution (v,u,τ,z,w,ρ) de ce système avec $\tau > 0$ fournit une solution optimale au primal $(\frac{1}{\tau}u)$ et au dual $(\frac{1}{\tau}v)$ et vice versa en prenant $\tau = 1, \rho = 0, z = A \cdot u - b, w = -A^T \cdot v + c$.
- Il ne peut y avoir une solution (v,u,τ,z,w,ρ) de ce système avec $\tau > 0$ et $\rho > 0$ car cela contradirait le théorème de dualité.
- Une solution (v,u,τ,z,w,ρ) de ce système avec $\rho > 0$ (et $\tau = 0$) garantit qu'il n'y a pas de solution optimale. En effet, la première ligne devient $A \cdot u \geq 0$, la deuxième ligne devient $A^T \cdot v \leq 0$ et la troisième ligne entraine que $b^T \cdot v > 0$ ou $c^T \cdot u < 0$.
  Dans le premier cas, notons u' une solution de (P), on a $0 < b^T \cdot v \leq u'^T \cdot A^T \cdot v \leq 0$. Ce qui est absurde.
  Dans le deuxième cas, notons v' une solution de (D), on a $0 > c^T \cdot u \geq v'^T \cdot A \cdot u \geq 0$. Ce qui est absurde.

En résumant, il nous suffit de trouver une solution avec $\tau > 0$ ou $\rho > 0$ pour résoudre notre problème.

Afin d'alléger les notations, on pose :

$$x = \begin{pmatrix} v \\ u \\ \tau \end{pmatrix} s(x) = \begin{pmatrix} z \\ w \\ \rho \end{pmatrix} M = \begin{pmatrix} 0 & A & -b \\ -A^T & 0 & c \\ b^T & -c^T & 0 \end{pmatrix}$$

Le système s'écrit alors :

$$(GT) \quad M \cdot x = s(x),\ x \geq 0,\ s(x) \geq 0$$

Notons que $M^T = -M$ et que M n'a ni colonne ni ligne nulle. Notre problème consiste alors à trouver une solution de (GT) telle que. $x_n + s(x)_n > 0$ où n est le nombre de composantes de x.

6.6.2 La forme auto-duale

Les méthodes intérieures maintiennent une solution dont *tous* les coefficients sont strictement positifs. Ceci n'est pas possible lorsque la forme de Goldman-Tucker a été obtenue par la transformation indiquée précédemment (car pour toute solution, $\rho\tau = 0$). Aussi nous introduisons une nouvelle formulation de notre problème.

Soit un problème (P)

$$M \cdot x = s(x),\ x \geq 0,\ s(x) \geq 0$$

avec M antisymétrique $n \times n$ et x est le vecteur colonne $(x_1, \ldots, x_n)$.

Notations. 0 désigne selon le contexte le vecteur ou l'entier nul. e dénote le vecteur dont toutes les composantes sont égales à 1, le vecteur r est défini par $r \equiv e - M \cdot e$.

Remarquons que $r^T \cdot e = e^T \cdot e - e^T \cdot M \cdot e = n$ (par antisymétrie de M). Par conséquent, $r \neq 0$. Nous définissons le problème étendu (PE) par :

$$\texttt{Minimiser } q^T \cdot x' \texttt{ telque}$$

$$M' \cdot x' + q = s(x'),\ x' \geq 0,\ s(x') \geq 0$$

avec

$$x' = \begin{pmatrix} x \\ x_{n+1} \end{pmatrix}, M' = \begin{pmatrix} M & r \\ -r^T & 0 \end{pmatrix}, q = \begin{pmatrix} 0 \\ n+1 \end{pmatrix}$$

Remarquons que le vecteur $x' \equiv 0$,$s(x') \equiv (0,n+1)$ est solution de (PE) et par conséquent le minimum recherché est 0.

La définition suivante introduit les notions clés associées aux méthodes intérieures.

Notations. Soit v et w deux vecteurs de même dimension, alors vw désigne le vecteur tel que $\forall i, (vw)_i = v_i w_i$. Dans la suite de ce chapitre, $v > 0 \equiv \forall i, v_i > 0$.

Définition 21 *Soit un programme linéaire dont les variables sont* $(x,s(x))$.
Une solution $(x,s(x))$ *est une solution* complémentaire *ssi* $xs(x) = 0$.
Une solution $(x,s(x))$ strictement complémentaire *est une solution complémentaire telle que* $x + s(x) > 0$.

Les dernières composantes d'une solution strictement complémentaire de (GT) garantissent que $\rho + \tau > 0$. Aussi nous chercherons à obtenir une solution strictement complémentaire de (GT).

Lemme 18 *Il existe une solution strictement complémentaire du problème* (GT) *ssi il existe une solution strictement complémentaire* optimale *du problème* (PE).

Preuve
Supposons qu'il existe une solution $(x,s(x))$ strictement complémentaire du problème (GT) alors la solution $((\alpha x,0),(\alpha s(x),n+1-\alpha r^t \cdot x))$, pour $\alpha > 0$ suffisamment petit, est une solution strictement complémentaire optimale du problème (PE).

Supposons qu'il existe une solution $(x',s(x'))$ strictement complémentaire optimale du problème (PE). Puisque la valeur minimale est 0, on a $x'_{n+1} = 0$. Par conséquent, la solution $(x, s(x))$ où $x \equiv (x'_1, \ldots, x'_n)$ est une solution strictement complémentaire du problème (GT).

c.q.f.d. ◊◊◊

De plus, cette transformation garantit que le problème (PE) a la propriété suivante.

Lemme 19 *Il existe une solution strictement positive du problème* (PE).

Preuve
Soit $e' \equiv (e,1)$. L'équation suivante établit que (e',e') est une solution strictement positive de (PE).

$$\begin{pmatrix} M & r \\ -r^T & 0 \end{pmatrix} \begin{pmatrix} e \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ n+1 \end{pmatrix} = \begin{pmatrix} M \cdot e + r \\ -r^T \cdot e + n + 1 \end{pmatrix} = \begin{pmatrix} e \\ 1 \end{pmatrix}$$

c.q.f.d. ◊◊◊

Nous arrivons à la forme du problème traité par les méthodes intérieures.

Définition 22 *Un problème auto-dual (SP) est défini par :*

$$(SP) \qquad \min\{q^T \cdot x \mid s(x) = M \cdot x + q, x \geq 0, s(x) \geq 0\}$$

avec M matrice $n \times n$ anti-symétrique, q vecteur colonne de dimension n à coefficients positifs. De plus M n'a pas de colonne nulle et il existe des indices i et j tels que $q_i > 0$ et $q_j = 0$.
Un problème auto-dual est dit initialisé *s'il admet une solution strictement positive.*

Nous noterons indifférement le vecteur des n dernières variables $s(x)$ pour insister sur le fait que ces variables sont déterminées par x ou simplement s par souci de lisibilité. Nous laissons le soin au lecteur de vérifier que ce problème est son propre dual (d'où la dénomination) et nous décrivons quelques propriétés de ce problème.

Proposition 31
- *Soient $(x,s(x))$ et $(y,s(y))$ deux solutions de (SP) alors $(x-y)^T \cdot (s(x)-s(y)) = 0$.*
- *Une solution de (SP) est optimale ssi elle est complémentaire.*
- *Soient $(x,s(x))$ une solution quelconque et $(y,s(y))$ une solution complémentaire, alors $\max(x^T \cdot s(y), y^T \cdot s(x)) \leq x^T \cdot s(x)$.*
- *Soient $(x,s(x))$ et $(y,s(y))$ deux solutions complémentaires, alors $xs(y) = 0$.*
- *Si (SP) est initialisé, alors l'ensemble des solutions optimales de (SP) est borné.*

Preuve
Soient $(x,s(x))$ et $(y,s(y))$ deux solutions de (SP).
$(x-y)^T \cdot (s(x)-s(y)) = (x-y)^T \cdot (M \cdot x + q - M \cdot y - q) = (x-y)^T \cdot M \cdot (x-y) = 0$
Soit $(x,s(x))$ une solution de (SP).
$q^T \cdot x = (s(x) - M \cdot x)^T \cdot x = s(x)^T \cdot x - x^T \cdot M^T \cdot x = s(x)^T \cdot x$
(par anti-symétrie)
Ce qui démontre la deuxième affirmation.

Soient $(x,s(x))$ une solution quelconque et $(y,s(y))$ une solution complémentaire.
$0 = (x-y)^T \cdot M \cdot (x-y) = (x-y)^T \cdot (s(x)-s(y))$
$= x^T \cdot s(x) + y^T \cdot s(y) - x^T \cdot s(y) - y^T \cdot s(x) = x^T \cdot s(x) - x^T \cdot s(y) - y^T \cdot s(x)$
D'où : $x^T \cdot s(y) + y^T \cdot s(x) = x^T \cdot s(x)$. Puisque chaque terme de cette somme est positive, on en déduit que $\max(x^T \cdot s(y), y^T \cdot s(x)) \leq x^T \cdot s(x)$.

Soient $(x,s(x))$ et $(y,s(y))$ deux solutions complémentaires.
$0 = (x-y)^T \cdot M \cdot (x-y) = (x-y)^T \cdot (s(x)-s(y))$
$= x^T \cdot s(x) + y^T \cdot s(y) - x^T \cdot s(y) - y^T \cdot s(x) = -x^T \cdot s(y) - y^T \cdot s(x)$
Puisque cette somme est positive, on en déduit que $x^T \cdot s(y) = y^T \cdot s(x) = 0$ et par conséquent $xs(y) = ys(x) = 0$.

Soit $(x0,s0)$ une solution strictement positive de (SP). et soit (x,s) une solution optimale. Alors (d'après le troisième point) :
$x^T \cdot s0 \leq x0^T \cdot s0$ et $x0^T \cdot s \leq x0^T \cdot s0$
Notons $xmin$ (resp. $smin$) la plus petite composante de $x0$ (resp. de $s0$).
On a : $\forall i, x_i \leq (1/smin)x0^T \cdot s0$ et $\forall i, s_i \leq (1/xmin)x0^T \cdot s0$.

c.q.f.d. ◊◊◊

Notons $B = \{i \mid \exists(x,s(x))$ `solution complémentaire` $, x_i > 0\}$ et $N = \{i \mid \exists(x,s(x))$ `solution complémentaire` $, s(x)_i > 0\}$. La proposition précédente permet d'affirmer que B et N sont des sous-ensembles disjoints de $\{1,\ldots,n\}$. D'autre part, N est non vide car $\{i \mid q_i > 0\} \subseteq N$.

6.6.3 Le chemin central

Notations. On notera SP l'ensemble des solutions de (SP) et SP^*, le sous-ensemble des solutions optimales. Soit w un vecteur, on note $D(w)$ la matrice diagonale dont la diagonale est le vecteur w. Si toutes les composantes de w sont non nulles, on note w^{-1}, le vecteur défini par $\forall i, w_i^{-1} = \frac{1}{w_i}$.

Nous voulons démontrer qu'il existe une solution strictement complémentaire à un problème initialisé et qu'il existe un « chemin » dans l'espace des solutions qui y conduit. A cette fin, nous introduisons quelques résultats techniques.

Lemme 20 *Soient M une matrice antisymétrique et D une matrice diagonale dont les coefficients diagonaux sont strictement positifs. Alors $D+M$ est inversible.*

Preuve
Soit $x \neq 0$, $x^T \cdot (D+M) \cdot x = x^T \cdot D \cdot x > 0$. Par conséquent, $(D+M) \cdot x \neq 0$.

c.q.f.d. ◊◊◊

Lemme 21 *Pour tout $0 < \underline{w} < \overline{w} \in \mathbb{R}^n$, les ensembles suivants sont compacts.*

$$L_{\overline{w}} = \{(x,s) \in SP \mid xs \leq \overline{w}\}$$

$$U(\underline{w},\overline{w}) = \{\underline{w} \leq w \leq \overline{w} \mid \exists(x,s) \in SP, xs = w\}$$

Preuve
$L_{\overline{w}}$ est un fermé. Montrons qu'il est borné. Comme précédemment $(x0,s0)$ désigne une solution strictement positive. Soit $(x,s) \in L_{\overline{w}}$, alors :
$0 = (x-x0)^T \cdot (s-s0) = x^T \cdot s + x0^T \cdot s0 - x^T \cdot s0 - x0^T \cdot s$
D'où : $x^T \cdot s0 \leq e^T \cdot \overline{w} + x0^T \cdot s0$ et $x0^T \cdot s \leq e^T \cdot \overline{w} + x0^T \cdot s0$
Notons $xmin$ (resp. $smin$) la plus petite composante de $x0$ (resp. de $s0$).
On a : $\forall i, x_i \leq (1/smin)(e^T \cdot \overline{w} + x0^T \cdot s0)$ et $\forall i, s_i \leq (1/xmin)(e^T \cdot \overline{w} + x0^T \cdot s0)$.
$U(\underline{w},\overline{w})$ est borné. Montrons que c'est un fermé. Supposons qu'il existe une séquence $\{w^i\}$ convergeant vers w. On a $\underline{w} \leq w \leq \overline{w}$. Soit la solution (x^i,s^i)

t.q $w^i = x^i s^i$. Puisque $(x^i,s^i) \in L_{\overline{w}}$ (compact) on peut en extraire une sous-suite convergente vers un certain (x,s). Par continuité, $xs = w$ ce qui permet de conclure.

c.q.f.d. $\Diamond\Diamond\Diamond$

Théorème 11 *Soit (SP) un problème auto-dual. Alors les trois propriétés suivantes sont équivalentes.*

(i) (SP) est initialisé.

(ii) Pour tout réel $\mu > 0$, il existe une solution strictement positive (x,s) du système

$$M \cdot x + q = s \wedge xs = \mu e$$

(iii) Pour tout vecteur $w > 0$, il existe une solution strictement positive (x,s) du système

$$M \cdot x + q = s \wedge xs = w$$

De plus, il y a unicité des solutions des systèmes d'équations ci-dessus.

Preuve

Démontrons l'unicité dans le cas (iii) (le cas (ii) en est un cas particulier). Soient (x,s) et (x',s') deux solutions du système d'équation. Puisque $xs = x's' = w$, on a $\forall i, x_i < x'_i \Leftrightarrow s_i > s'_i$ et vice versa. Par conséquent, $\forall i, (x_i - x'_i)(s_i - s'_i) \leq 0$ et $(x_i - x'_i)(s_i - s'_i) = 0 \Rightarrow x_i = x'_i \wedge s_i = s'_i$. Puisque d'après la proposition 31, $\sum_i (x_i - x'_i)(s_i - s'_i) = 0$, on en déduit que $(x,s) = (x',s')$.

De manière évidente $(iii) \Rightarrow (ii) \Rightarrow (i)$. Démontrons que $(i) \Rightarrow (iii)$. Soit $(x0,s0)$ la solution strictement positive, $w0 \equiv x0s0$ et $w > 0$ quelconque et différent de $w0$. Posons $\overline{w} = \max(w,w0)$, $\overline{\eta} = \max(\overline{w}_i) + 1$, $\underline{w} = \min(w,w0)$, $\underline{\eta} = (1/2)\min(\underline{w}_i)$. On a $\underline{\eta} e < w < \overline{\eta} e$ et $\underline{\eta} e < w0 < \overline{\eta} e$. Notons $U \equiv U(\underline{\eta} e, \overline{\eta} e)$. D'après le lemme 21, U est compact. Puisque $w0 \in U$, U est non vide. Soit la fonction $dist(w') \equiv \|w' - w\|_\infty$ qui mesure la distance à w. Cette distance atteint son minimum sur U en un vecteur $\hat{w}$ (U est compact). Montrons par l'absurde que $\hat{w} = w$, i.e. $dist(\hat{w}) = 0$.

Supposons que $dist(\hat{w}) > 0$. Notons $(\hat{x},\hat{s})$ la solution associée à $\hat{w}$. Nous allons nous approcher de w tout en restant dans U ce qui établira la contradiction. Posons $\Delta x \equiv (M + D(\hat{x})^{-1} \cdot D(\hat{s}))^{-1} \cdot (\hat{x}^{-1} w - \hat{s})$ et $\Delta s \equiv M \cdot \Delta x$. Soit α un réel positif, on a $\hat{s} + \alpha \Delta s = M \cdot (\hat{x} + \alpha \Delta x) + q$. Notons $w(\alpha) \equiv (\hat{x} + \Delta x)(\hat{s} + \Delta s)$. Pour α suffisamment petit, on a à la fois $\hat{x} + \Delta x > 0$ et $\hat{s} + \Delta s > 0$.

Explicitons $w(\alpha)$:
$w(\alpha) = \hat{w} + \alpha^2 \Delta x \Delta s + \alpha(\hat{x}(M \cdot \Delta x) + \hat{s}\Delta x)$
$= \hat{w} + \alpha^2 \Delta x \Delta s + \alpha(\hat{x}(M \cdot \Delta x) + \hat{x}(D(\hat{x})^{-1} \cdot D(\hat{s}) \cdot \Delta x))$
$= \hat{w} + \alpha^2 \Delta x \Delta s + \alpha(\hat{x}((M + D(\hat{x})^{-1} \cdot D(\hat{s})) \cdot \Delta x))$
$= \hat{w} + \alpha^2 \Delta x \Delta s + \alpha(\hat{x}(\hat{x}^{-1} w - \hat{s}))$
$= \hat{w} + \alpha^2 \Delta x \Delta s + \alpha(w - \hat{w})$
Par conséquent : $w(\alpha) - w = (1 - \alpha)(\hat{w} - w) + \alpha^2 \Delta x \Delta s$

On en déduit que pour α suffisamment petit,

— $dist(w(\alpha)) < dist(\hat{w})$
— Soit i tel que $|w_i - \hat{w}_i| > 0$ alors $|w_i - w(\alpha)_i| < |w_i - \hat{w}_i|$
— Soit i tel que $|w_i - \hat{w}_i| = 0$ alors $|w_i - w(\alpha)_i| < \min(1,(1/4)\min(w_i))$

Le premier point implique $w(\alpha)$ est plus proche de w que $\hat{w}$ et les deux autres points que $w(\alpha) \in U$, d'où la contradiction.

c.q.f.d. $\Diamond\Diamond\Diamond$

Remarque. Le choix de $\Delta x \equiv (M+D(\hat{x})^{-1} \cdot D(\hat{s}))^{-1} \cdot (\hat{x}^{-1}w - \hat{s})$ peut sembler « magique ». Il n'en est rien. Supposons qu'il existe $(\hat{x} + \Delta x, s + \Delta s)$ solution de (SP) telle que $(\hat{x} + \Delta x)(\hat{s} + \Delta s) = w$, alors :
$w - \hat{w} = \hat{x}\Delta s + \hat{s}\Delta x + \Delta x \Delta s$
On néglige le dernier terme (puisqu'on cherche une direction). D'où :
$\hat{x}^{-1}w - \hat{x}^{-1}\hat{w} = \Delta s + \hat{x}^{-1}\hat{s}\Delta x$
En observant que $\Delta s = M \cdot \Delta x$, on obtient :
$\hat{x}^{-1}w - \hat{s} = (M + D(\hat{x})^{-1} \cdot D(\hat{s})) \cdot \Delta x$
Ce qui fournit la direction de la preuve.

On appelle chemin central, l'ensemble des solutions $(x(\mu),s(\mu))$ pour μ réel strictement positif telles que $x(\mu)s(\mu) = \mu e$. Notre prochaine étape consiste à étudier le chemin central lorsque μ tend vers 0.

Théorème 12 *La limite* $\lim_{\mu \to 0}(x(\mu),s(\mu))$ *existe et c'est une solution strictement complémentaire de* (SP).

Preuve
Soit μ_k une suite décroissante de réels strictement positifs tendant vers 0. Alors $\forall k, (x(\mu_k),s(\mu_k)) \in L_{\mu_1 e}$, un compact. Cette suite admet donc un point d'accumulation (x^*,s^*).

Par continuité (x^*,s^*) est une solution de (SP) et même une solution optimale puisque $x^*s^* = \lim_{k\to\infty} \mu_k e = 0$. D'après la proposition 31,
$0 = (x^* - x(\mu_k))^T \cdot (s^* - s(\mu_k)) = n\mu_k - x^{*T} \cdot s(\mu_k) - x(\mu_k)^T \cdot s^*$
Notons $B^* = \{i \mid x_i^* > 0\}$ et $N^* = \{i \mid s_i^* > 0\}$, l'égalité précédente se réécrit :
$\sum_{i\in B^*} x_i^* s(\mu_k)_i + \sum_{i\in N^*} s_i^* x(\mu_k)_i = n\mu_k$
ou encore

$$\sum_{i\in B^*} \frac{x_i^*}{x(\mu_k)_i} + \sum_{i\in N^*} \frac{s_i^*}{s(\mu_k)_i} = n \tag{6.5}$$

En prenant la limite de cette égalité (restreinte à la sous-suite) quand k tend vers∞, on obtient que $|B^*| + |N^*| = n$, autrement dit (x^*,s^*) est une solution strictement complémentaire. Par conséquent, $B^* = B$, $N^* = N$ et $\{1,\ldots,n\} = B \uplus N$.

Soit (x^+,s^+) un autre point d'accumulation de la séquence. En prenant la limite de l'équation 6.5 restreinte à la sous-suite associée à (x^+,s^+), on obtient :
$(1/n)(\sum_{i\in B} \frac{x_i^*}{x_i^+} + \sum_{i\in N} \frac{s_i^*}{s_i^+}) = 1$
En inversant le raisonnement on en déduit que la moyenne arithmétique des nombres $\{\frac{x_i^*}{x_i^+}\}_{i\in B} \cup \{\frac{s_i^*}{s_i^+}\}_{i\in N}$ est égale à leur moyenne harmonique ce qui n'est possible que si tous ces nombres sont identiques, donc égaux à 1. Par conséquent, il n'y a qu'un seul point d'accumulation ce qui signifie que la suite converge.

c.q.f.d. $\Diamond\Diamond\Diamond$

En utilisant un raisonnement similaire à celui qui nous a permis de démontrer que le chemin central n'avait qu'un unique point d'accumulation, il est possible

de démontrer que la solution strictement complémentaire limite est l'unique solution complémentaire (donc optimale) qui maximise $\prod_{i\in B} x_i \prod_{i\in N} s_i$. Pour cette raison, on appelle $\lim_{\mu\to 0}(x(\mu),s(\mu))$ le *centre analytique* de (SP).

Observation. Nous affirmons que $B \neq \emptyset$. En effet, si $B = \emptyset$ alors (SP) admet une unique solution optimale $(0,q)$. D'après le résultat précédent, c'est une solution strictement complémentaire donc $q > 0$ contrairement à l'hypothèse d'existence d'un q_i nul.

6.6.4 Au voisinage du centre analytique

La distance d'une solution (x,s) strictement positive de (SP) au centre analytique peut être caractérisé par deux paramètres :

— sa « distance au bord » $x^T \cdot s$ à valeurs dans $[0,\infty[$;

— sa « distance au chemin central » $\frac{\max_{i\leq n} x_i s_i}{\min_{i\leq n} x_i s_i}$ que l'on note $\delta(x)$ à valeurs dans $[1,\infty[$.

Nous introduisons aussi une quantité fondamentale liée au problème (SP).

$$\sigma_{SP} \equiv \min_{i\leq n} \max_{x\in SP^*} x_i + s_i$$

Autrement dit, σ_{SP} représente le minimum sur l'ensemble des coordonnées de la valeur maximale atteinte sur une coordonnée par une solution optimale. En vertu de la proposition 31, σ_{SP} est fini et, en vertu du théorème 12, σ_{SP} est non nul.

Lemme 22 *Soit (x,s) une solution strictement positive de (SP), alors :*

$$\forall i \in B,\ x_i \geq \frac{\sigma_{SP}}{n\delta(x)} \ \wedge\ s_i \leq \frac{x^T \cdot s}{\sigma_{SP}}$$

$$\forall i \in N,\ s_i \geq \frac{\sigma_{SP}}{n\delta(x)} \ \wedge\ x_i \leq \frac{x^T \cdot s}{\sigma_{SP}}$$

Preuve
Appelons $\tau_1 \equiv \min_{i\leq n} x_i s_i$ et $\tau_2 \equiv \max_{i\leq n} x_i s_i$. On a : $\tau_2 = \tau_1 \delta(x)$. Soit $i \in B$, et $(\overline{x},\overline{s})$ la solution complémentaire qui maximise la variable x_i. On a :

$$\overline{x}_i s_i \leq \overline{x}^T \cdot s \leq x^T \cdot s$$

(la dernière inégalité est une des affirmations de la proposition 31).
D'où, puisque $\overline{x}_i \geq \sigma_{SP}$, $s_i \leq \frac{x^T\cdot s}{\sigma_{SP}}$.

Puisque $x_i s_i \geq \tau_1$, $x_i \geq \frac{\tau_1 \sigma_{SP}}{x^T\cdot s}$. D'après la définition de τ_2, $x^T \cdot s \leq n\tau_2$. Par conséquent :

$$x_i \geq \frac{\tau_1 \sigma_{SP}}{n\tau_2} = \frac{\sigma_{SP}}{n\delta(x)}$$

La preuve des deux autres inéquations est similaire et laissée en exercice.

c.q.f.d. ◊◊◊

Bien entendu, sous cette forme, ce résultat n'est guère utile car il repose sur σ_{SP} dont le calcul est au moins aussi difficile que le problème (SP) lui-même. Nous allons donc établir un minorant de σ_{SP} plus facile à calculer.

Lemme 23 *Soit A une matrice $n \times n$, alors $|det(A)| \leq \prod_{i \leq n} \|A[-,i]\|$ où $det(A)$ désigne le déterminant de A, et $\|v\|$ désigne la norme euclidienne du vecteur v.*

Preuve
Soit A une matrice orthogonale, alors $A^{-1} = A^T$. D'où $1 = det(Id) = det(A \cdot A^{-1}) = det(A)det(A^T) = det(A)^2$. D'où $|det(A)| = 1 = \prod_{i \leq n} \|A[-,i]\|$.

Soit A une matrice dont les vecteurs sont orthogonaux 2 à 2, et A' la matrice orthogonale obtenue en normalisant les vecteurs colonnes de A. Par multilinéarité du déterminant, $|det(A)| = \prod_{i \leq n} \|A[-,i]\| |det(A')| = \prod_{i \leq n} \|A[-,i]\|$

Soit A une matrice quelconque, alors on définit par récurrence une matrice A' dont les vecteurs sont orthogonaux 2 à 2, telle que $det(A') = det(A)$ et $\forall i, \|A'[-,i]\| \leq \|A[-,i]\|$. Soit v_i la vecteur projection de $A[-,i]$ sur l'espace linéaire engendré par $\{A[-,j]\}_{j<i}$. On pose $A'[-,i] = A[-,i] - v_i$. $A'[-,i]$ est orthogonal à $\{A[-,j]\}_{j<i}$, $\|A'[-,i]\| \leq \|A[-,i]\|$ et le déterminant de la matrice intermédiaire est inchangée (par multilinéarité du déterminant et le fait que v_i appartient à l'espace linéaire engendré par $\{A[-,j]\}_{j<i}$).

c.q.f.d. $\Diamond\Diamond\Diamond$

Lemme 24 *Soit A une matrice $m \times n$ à coefficients entiers, b un vecteur colonne $n \times 1$ à coefficients entiers et soit $\mathcal{S} \equiv \{x | A \cdot x = b \wedge x \geq 0\}$. Supposons que $\mathcal{S}$ soit borné et contienne un vecteur strictement positif. Alors :*

$$\forall i \leq n, \max(x_i \mid x \in \mathcal{S}) \geq \frac{1}{\prod_{i \leq n} \|A[-,i]\|}$$

Preuve
Supposons que $rang(A) < m$ alors en supprimant des lignes redondantes (comme dans l'algorithme du simplexe) on peut se ramener à une matrice A' (et un vecteur b') avec le même espace de solutions dont le rang est égal à son nombre de lignes. De plus puisque $\forall i \leq n, \|A'[-,i]\| \leq \|A[-,i]\|$, la démonstration du résultat pour A' entraîne le résultat pour A. Aussi on suppose dans la suite de la preuve que $rang(A) = m$.

Soit i un indice de colonne de A et $x \in \mathcal{S}$ une solution qui maximise x_i et parmi celles-ci une solution dont le support J ($J \equiv \{j \mid x_j > 0\}$) est minimal. Nous affirmons que les vecteurs colonnes de J sont indépendants. Si ce n'est pas le cas, il existe un vecteur y non nul dont le support est contenu dans J tel que $A \cdot y = 0$. Pour un $\varepsilon \in \mathbb{R}$ suffisamment petit en valeur absolue, le vecteur $x + \varepsilon y \in \mathcal{S}$. Par conséquent par maximalité de x_i, on a $y_i = 0$. Puisque $\mathcal{S}$ est borné, on ne peut augmenter indéfiniment ε. Soit ε la plus grande valeur telle que $x' \equiv x + \varepsilon y \geq 0$. Alors $x'_i = x_i$ et le support de x' est strictement inclus dans celui de x. Ce qui est contradictoire.

Puisque $rang(A) = m$, on peut compléter J pour obtenir un sous-ensemble d'indices K de cardinalité m t.q A_K, la matrice réduite aux colonnes de K, soit inversible. Par la règle de Cramer, on a :

$$x_i = \frac{det(A'_K)}{det(A_K)}$$

où A'_K est obtenue à partir de A_K en remplaçant $A[-,i]$ par b. Par conséquent, $x_i \geq \frac{1}{|det(A_K)|}$. En appliquant le lemme 23, on obtient :

$$x_i \geq \frac{1}{\prod_{i \in K} \|A[-,i]\|} \geq \frac{1}{\prod_{i \leq n} \|A[-,i]\|}$$

La dernière inégalité est justifiée par le fait que si A possédait une colonne i nulle alors $\mathcal{S}$ ne serait pas borné car on pourrait augmenter indéfiniment la composante x_i d'une solution quelconque.

c.q.f.d. ◊◊◊

Notations. Soit A une matrice, on note A_{IJ} la sous-matrice extraite dont les indices de ligne sont ceux de I (resp. de J). v_I est le vecteur colonne extrait du vecteur v dont les indices sont ceux de I.

Avec ces notations, il est immédiat que l'ensemble des solutions optimales projeté sur les composantes significatives est défini par le système linéaire :

$$\begin{pmatrix} M_{BB} & 0_{BN} \\ M_{NB} & -Id_{NN} \end{pmatrix} \begin{pmatrix} x_B \\ s_N \end{pmatrix} = \begin{pmatrix} 0_B \\ -q_N \end{pmatrix}$$

Par voie de conséquence (et en utilisant que le fait que M n'a pas de colonne nulle), le lemme 24 établit la proposition suivante.

Proposition 32 *Soit un problème (SP), alors :*

$$\sigma_{SP} \geq \frac{1}{\prod_{i \in B} \|M[-,i]\|} \geq \frac{1}{\prod_{i \leq n} \|M[-,i]\|}$$

On introduit la notation $\varepsilon(M) \equiv \frac{1}{\prod_{i \leq n} \|M[-,i]\|}$.

En combinant les résultats précedents, on obtient la détermination de la partition $B \uplus N$ lorsque la solution est suffisamment proche du centre analytique.

Théorème 13 *Soit (x,s) une solution strictement positive de (SP) telle que $(x^T \cdot s)\delta(x) < \frac{\varepsilon(M)^2}{n}$. Alors $B = \{i \mid x_i > s_i\}$ et $N = \{i \mid x_i < s_i\}$*

Preuve
Soit $i \in B$, d'après le lemme 22,

$$x_i \geq \frac{\sigma_{SP}}{n\delta(x)} \geq \frac{\varepsilon(M)}{n\delta(x)} > \frac{x^T \cdot s}{\varepsilon(M)} \geq \frac{x^T \cdot s}{\sigma_{SP}} \geq s_i$$

Un raisonnement similaire pour un indice $i \in N$ permet de conclure.

c.q.f.d. ◊◊◊

6.6.5 L'algorithme

L'algorithme consiste à se rapprocher itérativement de la solution strictement complémentaire du théorème 12 en suivant d'assez près que possible le chemin central. Arrivé en un point suffisamment proche, cette solution est alors obtenue par une résolution d'un système d'équations linéaire. C'est ce dernier point que nous étudions maintenant.

Supposons que nous ayons une solution strictement positive (x,s) que nous essayons de modifier pour obtenir une solution strictement complémentaire (x',s'). Notons $\Delta \equiv x_B - x'_B$. Puisque $s'_B = 0$, on doit vérifier l'équation suivante :

$$0 = M_{BB} \cdot x'_b = M_{BB} \cdot (x_B - \Delta) = s_B - M_{BN} \cdot x_N - M_{BB} \cdot \Delta$$

Autrement dit,

$$M_{BB} \cdot \Delta = s_B - M_{BN} \cdot x_N \tag{6.6}$$

Ce qui conduit à la définition suivante.

Définition 23 *Soit (x,s) une solution strictement positive de (SP). Alors $(\overline{x},\overline{s})$, un arrondi de (x,s) est défini par $\overline{x}_N = \overline{s}_B = 0$ et par :*

Cas $M_{BB} = 0$**.** $\overline{x}_B = x_B$ *et* $\overline{s}_N = s_N - M_{NN} \cdot x_N$

Cas $M_{BB} \neq 0$**.** *Soit $M_{B_1B_2}$ une matrice carrée inversible extraite de M de dimension maximale, soit $v \equiv M^{-1}_{B_1B_2} \cdot (s_{B_1} - M_{B_1N} \cdot x_N)$ un vecteur indicé par B_2, et Δ un vecteur indicé par B défini par $\Delta_{B_2} = v$ et $\Delta_{B \setminus B_2} = 0$. $\overline{x}_B = x_B - \Delta$ et $\overline{s}_N = s_N - M_{NN} \cdot x_N - M_{NB} \cdot \Delta$*

On parle d'un arrondi puisqu'il peut y avoir plusieurs choix de paires (B_1,B_2). La recherche de B_1 et B_2 se fait implicitement en résolvant par une élimination de Gauss (donc en temps polynômial) l'équation 6.6.

D'après la maximalité de (B_1,B_2), il existe une matrice C telle que
$M_{(B \setminus B_1)B} = C \cdot M_{B_1B}$
Ecrivons les égalités vérifiées par s_{B_1} et $s_{B \setminus B_1}$.
$s_{B_1} = M_{B_1B} x_B + M_{B_1N} x_N$
$s_{B \setminus B_1} = M_{(B \setminus B_1)B} x_B + M_{(B \setminus B_1)N} x_N$
D'où :

$$s_{B \setminus B_1} - M_{(B \setminus B_1)N} x_N = C \cdot (s_{B_1} - M_{B_1N} x_N) \tag{6.7}$$

Lemme 25 *Soit (x,s) une solution strictement positive de (SP) et $(\overline{x},\overline{s})$, un arrondi de (x,s). Alors $\overline{s} = M \cdot \overline{x} + q$.*

Preuve
Supposons d'abord $M_{BB} = 0$. Alors :
$\overline{s}_B = 0 = M_{BN} \cdot \overline{x}_N + q_B$
(car l'optimalité implique $q_B = 0$)
D'autre part :
$\overline{s}_N = s_N - M_{NN} \cdot x_N = M_{NB} \cdot x_B + M_{NN} \cdot x_N + q_N - M_{NN} \cdot x_N$
$= M_{NB} \cdot \overline{x}_B + M_{NN} \cdot \overline{x}_N + q_N$
Supposons maintenant que $M_{BB} \neq 0$. Alors :
$\overline{s}_N = s_N - M_{NN} \cdot x_N - M_{NB} \cdot \Delta$
$= M_{NB} \cdot x_B + M_{NN} \cdot x_N + q_N - M_{NN} \cdot x_N - M_{NB} \cdot (x_B - \overline{x}_B)$
$= M_{NB} \cdot \overline{x}_B + M_{NN} \cdot \overline{x}_N + q_N$

Nous décomposons la démonstration pour $\overline{s}_B$ $(= 0)$ en deux parties une pour les indices de B_1 et une pour les autres indices.
$M_{B_1B} \cdot \overline{x}_B = M_{B_1B} \cdot (x_B - \Delta) = M_{B_1B_2} \cdot (x_{B_2} - v) + M_{B_1(B\setminus B_2)} \cdot x_{B\setminus B_2}$
$= -M_{B_1B_2} \cdot M_{B_1B_2}^{-1} \cdot (s_{B_1} - M_{B_1N} \cdot x_N) + M_{B_1B} \cdot x_B$
$= -s_{B_1} + M_{B_1N} \cdot x_N + M_{B_1B} \cdot x_B = 0$

$M_{(B\setminus B_1)B} \cdot \overline{x}_B = M_{(B\setminus B_1)B} \cdot (x_B - \Delta)$
$= M_{(B\setminus B_1)B_2} \cdot (x_{B_2} - v) + M_{(B\setminus B_1)(B\setminus B_2)} \cdot x_{B\setminus B_2}$
$= -M_{(B\setminus B_1)B_2} \cdot M_{B_1B_2}^{-1} \cdot (s_{B_1} - M_{B_1N} \cdot x_N) + M_{(B\setminus B_1)B} \cdot x_B$
$= -C \cdot (s_{B_1} - M_{B_1N} \cdot x_N) + M_{(B\setminus B_1)B} \cdot x_B$
$= -s_{(B\setminus B_1)} + M_{(B\setminus B_1)N} \cdot x_N + M_{(B\setminus B_1)B} \cdot x_B$
(en utilisant l'équation 6.7)
$= 0$

c.q.f.d. ◊◊◊

Afin qu'un arrondi soit une solution strictement complémentaire, il nous faut garantir que $\overline{x}_B > 0$ et $\overline{s}_N > 0$. C'est l'objectif du lemme suivant.

Notations. Soit v un vecteur alors $\|v\|_\infty \equiv \max_i |v_i|$. On a : $\|v\| \leq \sqrt{n}\|v\|_\infty$. Soit M une matrice alors $\|M\|_\infty \equiv \max_i(\sum_j |A[i,j]|)$. On a : $\|M \cdot v\|_\infty \leq \|M\|_\infty \|v\|_\infty$.

Lemme 26 *Soit* (x,s) *une solution strictement positive de* (SP) *qui vérifie l'inégalité suivante.*

$$(x^T \cdot s)\delta(x) < \frac{\varepsilon(M)^3}{4n^{\frac{3}{2}}(\|M\|_\infty)^2} \tag{6.8}$$

Alors un arrondi $(\overline{x},\overline{s})$ *est une solution strictement complémentaire.*

Preuve
Remarquons que

$$\frac{\varepsilon(M)^3}{4n^{\frac{3}{2}}\|M\|_\infty)^2} < \frac{\varepsilon(M)^3}{2n^{\frac{3}{2}}\|M\|_\infty} < \frac{\varepsilon(M)^2}{2n\|M\|_\infty} < \frac{\varepsilon(M)^2}{n\|M\|_\infty} \leq \frac{\varepsilon(M)^2}{n}$$

Aussi l'inéquation 6.8 garantit que si on dispose de (x,s), alors B et N sont connus (théorème 13).
Supposons que $M_{BB} \neq 0$. Soit $i \in B_2$,

$$v_i = \frac{det(M_{B_1B_2}^{(i)})}{det(M_{B_1B_2})}$$

où $M_{B_1B_2}^{(i)}$ est obtenue en substituant à la ième colonne de $M_{B_1B_2}$ le vecteur $s_{B_1} - M_{B_1N} \cdot x_N$.
Par conséquent,
$|v_i| \leq |det(M_{B_1B_2}^{(i)})| \leq \|s_{B_1} - M_{B_1N} \cdot x_N\| \prod_{j\in B_2\setminus\{i\}} \|(M_{B_1B_2})_j\|$
$\leq \varepsilon(M)^{-1}\|s_{B_1} - M_{B_1N} \cdot x_N\| \leq \varepsilon(M)^{-1}(\|s_{B_1}\| + \|M_{B_1N} \cdot x_N\|)$
$\leq \varepsilon(M)^{-1}\sqrt{n}(\|s_{B_1}\|_\infty + \|M_{B_1N}\|_\infty \cdot \|x_N\|_\infty)$
$\leq \frac{\sqrt{n}(x^T\cdot s)(1+\|M\|_\infty)}{\varepsilon(M)^2} \leq \frac{2\sqrt{n}(x^T\cdot s)\|M\|_\infty}{\varepsilon(M)^2} < \frac{\varepsilon(M)}{n\delta(x)}$

Les dernières inégalités sont dues au lemme 22, à la proposition 32 et à l'inéquation 6.8.
Par conséquent en vertu des mêmes résultats,
$\forall i \in B, \overline{x}_i = x_i - \Delta_i \geq \frac{\varepsilon(M)}{n\delta(x)} - \max_j(|v_j|) > 0.$
Il reste à vérifier que $\overline{s}_N$ est strictement positif.
$\|M_{NB} \cdot \Delta\|_\infty \leq \|M_{NB}\|_\infty \|\Delta\|_\infty \leq \frac{2\sqrt{n}(x^T \cdot s)(\|M\|_\infty)^2}{\varepsilon(M)^2} < \frac{\varepsilon(M)}{2n\delta(x)}$
La dernière égalité étant due à l'inéquation 6.8.
D'autre part,
$\|M_{NN} \cdot x_N\|_\infty \leq \|M_{NN}\|_\infty \|x_N\|_\infty \leq \|M\|_\infty \|x_N\|_\infty \leq \frac{\|M\|_\infty x^T \cdot s}{\varepsilon(M)} < \frac{\varepsilon(M)}{2n\delta(x)}$

Par conséquent,
$\forall i \in N, \overline{s}_i = s_i - (M_{NN} \cdot x_N)_i - (M_{NB} \cdot \Delta)_i$
$\geq s_i - \|M_{NN} \cdot x_N\|_\infty - \|M_{NB} \cdot \Delta\|_\infty > \frac{\varepsilon(M)}{n\delta(x)} - \frac{\varepsilon(M)}{2n\delta(x)} - \frac{\varepsilon(M)}{2n\delta(x)} = 0$
Supposons que $M_{BB} = 0$. alors $\overline{x}_B = x_B > 0$. D'autre part, soit $i \in N$:
$\|M_{NN} \cdot x_N\|_\infty \leq \|M_{NN}\|_\infty \|x_N\|_\infty \leq \|M\|_\infty \|x_N\|_\infty \leq \frac{\|M\|_\infty x^T \cdot s}{\varepsilon(M)} < \frac{\varepsilon(M)}{n\delta(x)} \leq s_i$
Les dernières inégalités sont dues au lemme 22, à la proposition 32 et à l'inéquation 6.8. Par conséquent,
$\forall i \in N, \overline{s}_i = s_i - (M_{NN} \cdot x_N)_i \geq s_i - \|M_{NN} \cdot x_N\|_\infty > 0.$

c.q.f.d. ◊◊◊

Notations. Nous généralisons les opérations composantes par composantes des vecteurs. Soient u et v des vecteurs, $\frac{u}{v}$ dénote le vecteur obtenu par division composante par composante et u^k dénote le vecteur dont chaque composante est la composante correspondante de u mise à la puissance k.

Nous décrivons maintenant le coeur de l'algorithme. Soit $\tau > 1$ une distance au chemin central et (x,s) une solution courante strictement positive telle que $\delta(x) \leq \tau$, l'algorithme « améliore » la solution courante en lui substituant $(x+\Delta x, s+\Delta s)$ qui vérifie aussi $\delta(x+\Delta x) \leq \tau$ et telle que $(x+\Delta x)^T \cdot (s+\Delta s) \leq \rho(x^T \cdot s)$ avec $\rho < 1$ ne dépendant que de τ. En itérant cette substitution, on obtient un décroissance géométrique de $x^T \cdot s$ qui nous conduira en temps polynômial à une solution vérifiant l'inéquation 6.8.

Nous allons d'abord rechercher une bonne direction Δx (il y en a plusieurs). Cette direction est celle proposée par I.I. Dikin en 1967. Nous remplaçons la contrainte de positivité de la nouvelle solution par cette contrainte (non équivalente) :

$$\left\| \frac{\Delta x}{x} + \frac{\Delta s}{s} \right\| \leq 1$$

Cette ellipsoïde est appelée ellipsoïde de Dikin. Elle est introduite avec l'idée de considérer des déplacements relatifs par rapport à xs en négligeant le déplacement du second ordre $\Delta x \Delta s$.
On a $(x + \Delta x)^T \cdot (s + \Delta s) - x^T \cdot s = x^T \cdot \Delta s + s^T \cdot \Delta x$. Nous cherchons donc à résoudre le problème de minimisation suivant (dépendant de (x,s)).

$$\min \left\{ x^T \cdot \Delta s + s^T \cdot \Delta x \mid \Delta s = M \cdot \Delta x \wedge \left\| \frac{\Delta x}{x} + \frac{\Delta s}{s} \right\| \leq 1 \right\}$$

En introduisant des objets intermédiaires, nous allons simplifier l'expression de ce problème. Soient le scalaire $\mu \equiv \frac{x^T \cdot s}{n}$, et les vecteurs $d \equiv \sqrt{\frac{x}{s}}$ et $u \equiv \sqrt{\frac{xs}{\mu}}$. Observons que $d^{-1}x = ds = \sqrt{\mu}u$. D'une certaine manière, d consiste à changer l'échelle des axes. Aussi nous introduisons les déplacements remis à la nouvelle échelle $p_x \equiv \sqrt{\mu}^{-1} d^{-1}\Delta x$ et $p_s \equiv \sqrt{\mu}^{-1} d\Delta s$. Observons que $\frac{\Delta x}{x} = \frac{p_x}{u}$ et $\frac{\Delta s}{s} = \frac{p_s}{u}$, $\mu p_x p_s = \Delta x \Delta s$ et par conséquent $p_x^T \cdot p_s = 0$.
Réécrivons notre problème auxiliaire.
$\Delta s - M \cdot \Delta x = \sqrt{\mu} d^{-1} p_s - M \cdot (\sqrt{\mu} d p_x)$
Aussi $\Delta s - M \cdot \Delta x = 0$ est équivalent à :
$p_s = D(d) \cdot M \cdot D(d) \cdot p_x$
Puisque $s\Delta x + x\Delta s = xs\left(\frac{\Delta x}{x} + \frac{\Delta s}{s}\right) = \mu u(p_x + p_s)$, l'objectif se réécrit :
$s^T \cdot \Delta x + x^T \cdot \Delta s = \mu u^T \cdot (p_x + p_s)$

Nous introduisons finalement $p \equiv p_x + p_s$. Remarquons qu'à partir de p on retrouve p_x (et donc p_s) par la formule $p_x = (Id + D(d) \cdot M \cdot D(d))^{-1} \cdot p$. Le problème auxiliaire s'écrit alors :

$$\min\left\{u^T \cdot p \mid \left\|\frac{p}{u}\right\| \leq 1\right\}$$

Un dernier changement de variable nous conduit à la solution. Soit $\overline{p} = \frac{p}{u}$ alors le problème s'écrit :

$$\min\left\{(u^2)^T \cdot \overline{p} \mid \|\overline{p}\| \leq 1\right\}$$

Autrement dit, $\overline{p} = -\frac{u^2}{\|u^2\|}$. D'où : $p = -\frac{u^3}{\|u^2\|}$.
La direction recherchée est donc :

$$\Delta x = \sqrt{\mu} D(d) \cdot (Id + D(d) \cdot M \cdot D(d))^{-1} \cdot \left(-\frac{u^3}{\|u^2\|}\right) \wedge \Delta s = M \cdot \Delta x$$

Le déplacement effectif sera $\alpha\Delta x$ (resp. $\alpha\Delta x$) avec $\alpha > 0$ et la nouvelle solution sera notée $x(\alpha) \equiv x + \alpha\Delta x$ (resp. $s(\alpha) \equiv s + \alpha\Delta s$). Calculons le vecteur $x(\alpha)s(\alpha)$.

$$x(\alpha)s(\alpha) = \sqrt{\mu} d(u + \alpha p_x)\sqrt{\mu} d^{-1}(u + \alpha p_s) = \mu(u^2 - \alpha\frac{u^4}{\|u^2\|} + \alpha^2 p_x p_s) \quad (6.9)$$

Le lemme suivant « valide » la direction de Dikin.

Lemme 27 *Soit* (x,s) *une solution strictement positive de* (SP) *et* $\alpha > 0$. *Alors :*

$$x(\alpha)^T \cdot s(\alpha) \leq (1 - \frac{\alpha}{\sqrt{n}})x^T \cdot s$$

Preuve
On rappelle d'abord $x^T \cdot s = \mu e^T \cdot u^2 = \mu\|u\|^2$ et que p_x et p_s sont orthogonaux. Par conséquent, en vertu de l'équation 6.9 :
$\mu^{-1} x(\alpha)^T \cdot s(\alpha) = \|u\|^2 - \alpha\frac{e^T \cdot u^4}{\|u^2\|} = \|u\|^2 - \alpha\|u^2\|$
Or (inégalité de Cauchy-Schwartz) : $\|u^2\| = \frac{1}{\sqrt{n}}\|e\|\|u^2\| \geq \frac{e^T \cdot u^2}{\sqrt{n}} = \frac{\|u\|^2}{\sqrt{n}}$
Par conséquent : $x(\alpha)^T \cdot s(\alpha) = \mu(1 - \frac{\alpha}{\sqrt{n}})\|u\|^2 = (1 - \frac{\alpha}{\sqrt{n}})x^T \cdot s$

c.q.f.d. ◊◊◊

Il nous reste à trouver quelles valeurs de α garantissent que $(x(\alpha),s(\alpha))$ une solution strictement positive de (SP) (il en existe nécessairement au voisinage de 0). C'est l'objet des lemmes qui suivent.

Lemme 28 *Soit (x,s) une solution strictement positive de (SP) et $\alpha > 0$ tel que pour tout $0 \leq \alpha' \leq \alpha$, on a $x(\alpha')s(\alpha') > 0$. Alors $(x(\alpha),s(\alpha))$ est une solution strictement positive de (SP).*

Preuve
Pour $\alpha' \in [0,\alpha]$, $x(\alpha')$ et $s(\alpha')$ n'ont jamais une composante nulle. Puisqu'initialement, ces composantes sont strictement positives, par continuité aucune d'elles ne peut devenir négative.

c.q.f.d. $\Diamond\Diamond\Diamond$

A l'aide du lemme précédent, il nous suffit de vérifier que sur un intervalle contenant 0, $x(\alpha)s(\alpha) > 0$. Le lemme suivant nous fournit l'intervalle recherché. Afin de faciliter son énoncé, nous nous donnons τ tel que $\tau \geq \delta(x)$ et nous introduisons deux quantités auxiliaires : $\tau_1 = \min_i(u_i^2)$ et $\tau_2 = \tau\tau_1 \geq \max_i(u_i^2)$.

Lemme 29 *Soit (x,s) une solution strictement positive de (SP) et $\tau > 1$ tels que $\delta(x) \leq \tau$. Alors pour tout α tel que $\alpha \leq \frac{\|u^2\|}{2\tau_2} \wedge \alpha < \frac{4\tau_1}{\|u^2\|}$:*
$(x(\alpha),s(\alpha))$ est une solution strictement positive de (SP) et $\delta(x(\alpha)) < \tau$.

Preuve
La fonction $t \mapsto t - \alpha\frac{t^2}{\|u^2\|}$ est croissante sur $[0,\tau_2]$ d'après la première contrainte sur α. En appliquant cette fonction aux composantes de u^2, on obtient :

$$\left(\tau_1 - \alpha\frac{\tau_1^2}{\|u^2\|}\right)e \leq u^2 - \alpha\frac{u^4}{\|u^2\|} \leq \left(\tau_2 - \alpha\frac{\tau_2^2}{\|u^2\|}\right)e$$

En ajoutant aux termes des inégalités le vecteur $\alpha^2 p_x p_s$ puis en multipliant par μ, on obtient :

$$\mu\left(\left(\tau_1 - \alpha\frac{\tau_1^2}{\|u^2\|}\right)e + \alpha^2 p_x p_s\right) \leq x(\alpha)s(\alpha) \leq \mu\left(\left(\tau_2 - \alpha\frac{\tau_2^2}{\|u^2\|}\right)e + \alpha^2 p_x p_s\right)$$

Il nous suffit alors de démontrer que : $0 < \left(\tau_1 - \alpha\frac{\tau_1^2}{\|u^2\|}\right)e + \alpha^2 p_x p_s$
(d'après le lemme précédent)
et que : $\left(\tau_2 - \alpha\frac{\tau_2^2}{\|u^2\|}\right)e + \alpha^2 p_x p_s < \tau\left(\left(\tau_1 - \alpha\frac{\tau_1^2}{\|u^2\|}\right)e + \alpha^2 p_x p_s\right)$
mais la seconde inégalité implique la première. En effet supposons qu'il existe une composante de $\left(\tau_1 - \alpha\frac{\tau_1^2}{\|u^2\|}\right)e + \alpha^2 p_x p_s$ qui soit nulle alors d'après l'encadrement de $x(\alpha)s(\alpha)$, la composante correspondante de $\left(\tau_2 - \alpha\frac{\tau_2^2}{\|u^2\|}\right)e + \alpha^2 p_x p_s$ est positive ou nulle contredisant la deuxième inégalité qui est stricte. Par conséquent les coordonnées restent strictement positives si la deuxième inégalité est vérifiée.
En rappelant que $\tau_2 = \tau\tau_1$, l'inégalité devient : $\left(\frac{\tau_2^2 - \tau\tau_1^2}{\|u^2\|}\right)e + \alpha(\tau-1)p_x p_s > 0$
et en simplifiant encore : $\frac{\tau_1\tau_2}{\|u^2\|}e + \alpha p_x p_s > 0$

Nous allons substituer à cette condition, une condition plus forte mais aussi plus simple à vérifier.

$$p_x p_s = \frac{1}{4}\left((p_x + p_s)^2 - (p_x - p_s)^2\right)$$

Par conséquent,
$-\frac{1}{4}(p_x - p_y)^2 \leq p_x p_y \leq \frac{1}{4}(p_x + p_y)^2$ et $-\frac{1}{4}\|p_x - p_y\|^2 \leq p_x p_y \leq \frac{1}{4}\|p_x + p_y\|^2$
Puisque p_x et p_y sont orthogonaux $\|p_x - p_y\| = \|p_x + p_y\|$
et par conséquent, $\|p_x p_s\|_\infty \leq \frac{1}{4}\|p_x + p_y\|^2 = \frac{1}{4}\|p\|^2$.
Il nous suffit donc de vérifier que : $\frac{\tau_1\tau_2}{\|u^2\|} - \frac{\alpha\|p\|^2}{4} > 0$
Or :

$$\|p\| = \left\|\frac{u^3}{\|u^2\|}\right\| \leq \|u\|_\infty \left\|\frac{u^2}{\|u^2\|}\right\| = \|u\|_\infty \leq \sqrt{\tau_2}$$

Par conséquent : $\frac{\tau_1\tau_2}{\|u^2\|} - \alpha\frac{\|p\|^2}{4} \geq \frac{\tau_1\tau_2}{\|u^2\|} - \frac{\alpha\tau_2}{4} > 0$
en vertu de la deuxième contrainte sur α.

c.q.f.d. ◊◊◊

Ce dernier lemme nous permet de définir α indépendamment de (x,s).

Lemme 30 *Soit (x,s) une solution strictement positive de (SP) et $\delta(x) \leq \tau$ Soit $\alpha \equiv \frac{1}{\tau\sqrt{n}}$. Alors :*

$$\alpha \leq \frac{\|u^2\|}{2\tau_2} \wedge \alpha < \frac{4\tau_1}{\|u^2\|}$$

Preuve
Rappelons que $n \geq 2$.

$$\frac{1}{\tau\sqrt{n}} = \frac{\tau_1}{\tau_2\sqrt{n}} \leq \frac{\sqrt{n}\tau_1}{2\tau_2} = \frac{\|\tau_1 e\|}{2\tau_2} \leq \frac{\|u^2\|}{2\tau_2}$$

D'autre part,

$$\frac{1}{\tau\sqrt{n}} < \frac{4}{\tau\sqrt{n}} = \frac{4\tau_1}{\tau_2\sqrt{n}} \leq \frac{4\tau_1}{\|u^2\|}$$

c.q.f.d. ◊◊◊

L'algorithme 43 synthéthise les différentes étapes du calcul.

6.6.6 Analyse de complexité

Sachant que la solution strictement positive initiale est sur le chemin central, nous pouvons choisir n'importe quel $\tau > 1$. Nous choisissons $\tau = 2$ pour effectuer notre analyse de complexité.
En notant (x,s) la solution courante et (x',s') la nouvelle solution, à chaque itération $x'^T \cdot s' \leq (1 - \frac{1}{2n})x^T \cdot s$. Initialement $x^T \cdot s = 1$ et nous arrêtons les itérations lorsque $x^T \cdot s \leq \frac{\varepsilon(M)^3}{8n^{\frac{3}{2}}(\|M\|_\infty)^2}$.
Soit t la taille de la représentation du problème,
$\varepsilon(M)^{-1} \leq (\sqrt{n}2^t)^n$, $\|M\|_\infty \leq n2^t$ et $n \leq t$.
Par conséquent, $\varepsilon(M)^{-1} \leq 2^{t^2+(1/2)t\log(t)}$ et $\|M\|_\infty \leq 2^{t+\log(t)}$
$\frac{\varepsilon(M)^3}{8n^{\frac{3}{2}}(\|M\|_\infty)^2} \geq \frac{1}{8\cdot 2^{5\log(t)+2t+3t^2}}$
Pour t assez grand $\frac{1}{8\cdot 2^{5\log(t)+2t+3t^2}} \geq \frac{1}{2^{4t^2}}$

Algorithme 43: Une méthode intérieure

`Optimiser`(M,q) : solution
Input : M, une matrice $n \times n$ antisymétrique
Input : q, un vecteur positif de dimension n tel que $e = M \cdot e + q$
Output : (x,s) une solution strictement complémentaire
Data : B,N des ensembles d'indices
Data : μ,ε des scalaires ; d,u,Δx, Δs des vecteurs de dimension n
Data : Δ un vecteur indicé par B

$x \leftarrow e$; $s \leftarrow e$
$\varepsilon \leftarrow \frac{\varepsilon(M)^3}{4n^{\frac{3}{2}}(\|M\|_\infty)^2}$
while $x^T \cdot s > \varepsilon$ **do**
  $\mu \leftarrow \frac{x^T \cdot s}{n}$; $d \leftarrow \sqrt{\frac{x}{s}}$; $u \leftarrow \sqrt{\frac{xs}{\mu}}$
  $\Delta x \leftarrow \sqrt{\mu} D(d) \cdot (Id + D(d) \cdot M \cdot D(d))^{-1} \cdot \left(-\frac{u^3}{\|u^2\|}\right)$
  $\Delta s \leftarrow M \cdot \Delta x$
  $x \leftarrow x + \frac{1}{2\sqrt{n}} \Delta x$; $s \leftarrow s + \frac{1}{2\sqrt{n}} \Delta s$
end
$B \leftarrow \{i \mid x_i > s_i\}$; $N \leftarrow \{i \mid x_i < s_i\}$
if $M_{BB} = 0$ **then**
  $s_N \leftarrow s_N - M_{NN} \cdot x_N$
else
  `Resoudre` en Δ l'équation $M_{BB} \cdot \Delta = s_B - M_{BN} \cdot x_N$
  $x_B \leftarrow x_B - \Delta$
  $s_N \leftarrow s_N - M_{NN} \cdot x_N - M_{NB} \cdot \Delta$
end
$x_N \leftarrow 0$; $s_B \leftarrow 0$
return (x,s)

Aussi le nombre d'itérations est borné par un k tel que

$$(1 - \frac{1}{2n})^k \leq \frac{1}{2^{4t^2}}$$

En passant au logarithme k doit vérifier

$$k \log(1 - \frac{1}{2n}) \leq -4t^2$$

ou de manière équivalente

$$-k \log(1 - \frac{1}{2n}) \geq 4t^2$$

Puisque $-\log(1 - \frac{1}{2n}) \geq \frac{1}{2n}$, il sufit que k vérifie :

$$\frac{k}{2n} \geq 4t^2$$

Puisque $n \leq t$, il suffit que k vérifie :

$$k \geq 8t^3$$

Le nombre d'itérations est donc polynomial. Le lemme suivant établit un encadrement des composantes de la solution strictement positive durant les itérations.

Lemme 31 *Soit* (x,s) *une solution strictement positive rencontrée lors des itérations de l'algorithme 43. Alors :*

$$\forall i \ \frac{\varepsilon(M)^3}{8n^{\frac{7}{2}}(\|M\|_\infty)^2} \leq s_i,x_i \leq 2n$$

Preuve
On rappelle qu'en vertu du lemme 27, $x^T \cdot s \leq e^T \cdot e = n$. Appliquons le premier point de la proposition 31 avec la solution initiale (e,e) et la solution courante (x,s).

$$e^T \cdot x + e^T \cdot s = x^T \cdot s + e^T \cdot e \leq 2n$$

D'où $\forall i \ s_i,x_i \leq 2n$.

D'après le test de la boucle $\varepsilon \leq x^T \cdot s$ où ε est la variable de l'algorithme. Sachant que $\delta(s) \leq 2$, $\forall i \ 2nx_is_i \geq \varepsilon$. En se servant de la borne supérieure, on obtient $\forall i \ x_i,s_i \geq \frac{\varepsilon}{2n^2}$.

c.q.f.d. ◊◊◊

En choisissant une précision qui permet de représenter des valeurs suffisament petites (e.g. $10^{-2}\frac{\varepsilon(M)^3}{8n^{\frac{7}{2}}(\|M\|_\infty)^2}$), les pertubations d'un calcul approché lors des itérations ne modifient pas la convergence. Il suffit par exemple de prendre α égal à la moitié de la valeur théorique. D'après le lemme précédent, le calcul peut même se faire en représentation fixe. Seul le calcul final (une élimination de Gauss) doit s'effectuer dans les rationnels. Ce qui signifie que chaque opération arithmétique se fait en temps polynomial. Enfin le nombre d'opérations arithmétiques de chaque itération ainsi que celui de l'arrondi final est polynomial.

Bibliographie

[Aho et al] A.V. Aho, J.E. Hopcroft and J.D. Ullman. The Design and Analysis of Computer Algorithms. *Addison-Wesley Publishing Company 1974*

[Beauquier et al] D. Beauquier, J. Berstel and P. Chrétienne. Eléments d'algorithmique. *Masson 1992* `http://www-igm.univ-mlv.fr/~berstel/Elements/Elements.html`

[Cormen et al] T. Cormen, C. Leiserson, R. Rivest and C. Stein. Introduction à l'algorithmique. *Dunod 2002 2ème édition ISBN 2-10-003922-9*

[Crochemore et Rytter] M. Crochemore and W. Rytter. Jewels of Stringology. *World Scientific, 2002, 310 pages. ISBN 981-02-4782-6*

[Mitzenmacher et Upfal] M.Mitzenmacher and E. Upfal Probability and Computing. Randomized Algorithms and Probabilistic Analysis. *Cambridge Univeristy Press 2005 ISBN 0-521-83540-2*

[Nelson et Gailly] M. Nelson and J-L. Gailly The Data Compression Book. *John Wiley & Sons 2ème édition 1995 ISBN 978-1558514348*

[Papadimitriou et Steigliz] C. H. Papadimitriou and K. Steigliz. Combinatorial Optimization. Algorithms and Complexity. *Dover publications 1998 2ème édition ISBN 0-486-40258-4*

[Roos et al] C. Roos, T. Terlaky and J. P. Vial Theory and Algorithms for Linear Optimization : An Interior Point Approach. *John Wiley and Sons 1997 ISBN 0-471-95676-7*

[Vazirani] V. Vazirani Approximation Algorithms. *Springer 2003 ISBN 3540653678*