Schémas algorithmiques

Dans ce chapitre, on présente différents paradigmes algorithmiques qui permettent de développer des algorithmes résolvant des problèmes (souvent d'optimisation) mettant en jeu des textes, des nombres, des listes, des graphes, des expressions arithmétiques, des points du plan, des ordonnancements...

0.1 Problèmes d'optimisation

Rappel (*problème d'optimisation*) :

Un problème d'optimisation consiste à maximiser ou minimiser une fonction à valeurs réelles donnée sur un ensemble de solutions donné.

Définition (*fonction-objectif, valeur, solutions optimales*) :

Tout problème d'optimisation $\mathcal{P}$ est donc défini par une fonction f appelée fonction-objectif et l'ensemble des solutions $\mathcal{S}$. S'il s'agit de maximiser f, on notera :

$$\mathcal{P} : \max_{X \in \mathcal{S}} f(X)$$

On appelle alors valeur du problème $\mathcal{P}$ le réel : $\text{val}(\mathcal{P}) = \max\{f(X) \mid X \in \mathcal{S}\}$.
De plus, on note : $\underset{X \in \mathcal{S}}{\text{argmax}}\, f(X) = \{X \in \mathcal{S} \mid f(X) = \text{val}(\mathcal{P})\}$ l'ensemble des solutions optimales.

Remarque : Bien sûr, si l'on cherche à minimiser f, les définitions précédentes sont valables en remplaçant max par min et argmax par argmin.

Remarque : Il y a unicité de la valeur optimale mais pas toujours des solutions optimales.

Définition (*relaâchés*) :

Si $\mathcal{S} \subseteq \widetilde{\mathcal{S}} \subseteq \mathcal{D}(f)$, on dit que $\widetilde{\mathcal{P}} = \max_{X \in \widetilde{\mathcal{S}}} f(X)$ est un relâché de $\mathcal{P}$. On a alors : $\text{val}(\widetilde{\mathcal{P}}) \geq \text{val}(\mathcal{P})$.

0.2 Notions sur les produits scalaires dans l'espace $\mathbb{R}^n$

Définition (*indicatrices*) :

Soit $n \in \mathbb{N}^*$. Pour $i \in [1..n]$, on note $\mathbb{1}_i$ ou $\mathbb{1}_{\{i\}}$ et on appelle indicatrice de i, le vecteur de $\mathbb{R}^n$ où toutes les composantes valent 0 sauf en position i où elle vaut 1 :

$$\mathbb{1}_i = (0,...,0,1,0,...,0) = (\delta_{i,j})_{j \in [1..n]}$$

De plus, pour $E \subseteq [1..n]$, on note :

$$\mathbb{1}_E = \sum_{i \in E} \mathbb{1}_i$$

ce qui justifie la notation $\mathbb{1}_{\{i\}}$ pour $\mathbb{1}_i$. Enfin, on note $\mathbb{1}$ le vecteur :

$$\mathbb{1} = \mathbb{1}_{[1..n]} = \sum_{i=1}^{n} \mathbb{1}_i = (1,1,...,1)$$

Rappel (*vecteur colonne*) :

Alors, pour tout $x = (x_1,..., x_n) \in \mathbb{R}^n$, on a :

$$x = \sum_{i=1}^{n} x_i \mathbb{1}_i$$

On associe à ce vecteur sa représentation en vecteur coordonnées ou vecteur colonne, donnée par la matrice colonne suivante de $\mathscr{M}_{n,1}(\mathbb{R})$:

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Remarque : Pour désigner la j-ième composante de l'indicatrice de i, on notera indifféremment $(\mathbb{1}_i)_j$ ou $\mathbb{1}_{\{i\}}(j)$, la deuxième notation faisant apparaître l'indicatrice plus comme une fonction, appelée fonction indicatrice.

Définition (*produit scalaire*) :

Soit $(u, v) \in (\mathbb{R}^n)^2$. Le produit scalaire de u et de v, que l'on note $u \cdot v$, est :

$$u \cdot v = {}^tU \times V = \sum_{i=1}^{n} u_i v_i$$

Remarque : Il s'agit du produit scalaire canonique dans $\mathbb{R}^n$.

1 Algorithmes gloutons

1.1 L'exemple du problème du sac à dos

1.1.1 Présentation

Le principe du problème du sac à dos est le suivant : on dispose d'objets possédant chacun une valeur et un poids, ainsi que d'un sac qui peut les transporter avec une limitation de poids. L'objectif est alors de choisir des objets de façon à maximiser la valeur cumulée contenue dans le sac :

SAC À DOS

entrée : $P \in \mathbb{R}$
$v = (v_i)_{i\in[1..n]} \in (\mathbb{R}^{+*})^n$
$p = (p_i)_{i\in[1..n]} \in (\mathbb{R}^{+*})^n$

sortie : $\max\limits_{\substack{\delta\in\{0,1\}^n \\ \sum_{i=1}^n \delta_i p_i \leq P}} \sum\limits_{i=1}^{n} \delta_i v_i = \max\limits_{\substack{\delta\in\{0,1\}^n \\ \delta\cdot p \leq P}} \delta \cdot v$

On en introduit une variante (dont on verra qu'elle est plus facile), appelée problème du sac à dos fractionnaire, dans laquelle on s'autorise à ne prendre que des fractions de certains objets :

SAC À DOS FRACTIONNAIRE

entrée : $P \in \mathbb{R}$
$v = (v_i)_{i\in[1..n]} \in (\mathbb{R}^{+*})^n$
$p = (p_i)_{i\in[1..n]} \in (\mathbb{R}^{+*})^n$

sortie : $\max\limits_{\substack{\delta\in[0,1]^n \\ \delta\cdot p \leq P}} \delta \cdot v$

Remarque : Sac à dos fractionnaire est un relâché de Sac à dos.

Remarque : La contrainte $\delta \in [0,1]^n$ dans Sac à dos fractionnaire peut aussi s'écrire :

$$\begin{cases} \forall\, i \in [1..n],\ \delta \cdot \mathbb{1}_{\{i\}} \geq 0 \\ \forall\, i \in [1..n],\ \delta \cdot \mathbb{1}_{\{i\}} \leq 1. \end{cases}$$

puisque $\forall\, i \in [1..n],\ \delta \cdot \mathbb{1}_{\{i\}} = \sum_{j=1}^{n} \delta_j \mathbb{1}_{\{i\}}(j) = \delta_i \times \mathbb{1}_{\{i\}}(i) = \delta_i$.
L'ensemble des $\delta \in \mathbb{R}^n$ vérifiant cette contrainte forme alors un hypercube de dimension n.

1.1.2 La difficulté de Sac à dos

Il s'avère que Sac à dos est un problème difficile, et qu'on ne peut donc pas le résoudre par une méthode "simple" (c'est-à-dire gloutonne, *cf.* plus loin) telle que celles consistant à prendre les objets par v_i décroissants, p_i croissants ou v_i/p_i décroissants.

On illustre cela dans les exemples suivants.

Exemple : Prenons $P = 20$ et les objets du tableau suivant :

i	p_i	v_i
1	20	10
2	10	9
3	10	9

Si on les choisit *par v_i décroissants*, on trouve : $\delta^* = (1,0,0)$, de valeur $\delta^* \cdot v = 10$.

Pourtant, $\delta = (0,1,1)$ vérifie bien $\delta \cdot p = 20 \leq P$ et on a $\delta \cdot v = 18 > 10$: cette façon de choisir les objets ne permet donc pas d'avoir un sac à dos optimal.

Exemple : Toujours pour $P = 20$, on considère ensuite :

i	p_i	v_i
1	18	10
2	10	1
3	10	1

Choisir cette fois *par p_i croissants* donne : $\delta^* = (0,1,1)$, de valeur $\delta^* \cdot v = 2$.

Or, $\delta = (1,0,0)$ est une autre solution car $\delta \cdot p = 18 \leq P$, et elle est de valeur $\delta \cdot v = 10 > 2$, ce qui montre que cette stratégie n'est pas non plus optimale.

Exemple : En gardant $P = 20$, considérons enfin :

i	p_i	v_i	v_i/p_i
1	11	22	2
2	10	15	1,5
3	10	15	1,5

Si on sélectionne les objets *par (v_i/p_i) décroissants*, on a : $\delta^* = (1,0,0)$, de valeur $\delta^* \cdot v = 22$.

Encore une fois, on peut trouver une solution meilleure invalidant cette façon de choisir les objets : par exemple, $\delta = (0,1,1)$ dont la valeur est $\delta \cdot v = 30 > 22$.

1.1.3 Résolution de Sac à dos fractionnaire

Sac à dos fractionnaire est en revanche un problème facile, et la dernière des trois stratégies vues précédemment va nous permettre de le résoudre.

Exemple : Sur le dernier exemple, le vecteur :

$$\delta^* = \begin{pmatrix} 1 \\ 9/10 \\ 0 \end{pmatrix}$$

obtenu en choisissant par rapports v_i/p_i décroissants (même pour le dernier objet, dont on n'a pris qu'une fraction), est bien une solution optimale, avec pour valeur $\delta^* \cdot v = 22 + 13{,}5 = 35{,}5$.

Voici donc un pseudo-code de l'algorithme correspondant :

Algorithme – Rempli-sac

entrée : P un réel
$(p_i, v_i)_{i\in[1..n]} \in \left((\mathbb{R}^{+*})^2\right)^n$ une suite finie de réels strictement positifs

Trier les objets par ratio v_i/p_i décroissant, *i.e.* trouver $\sigma \in \mathscr{B}ij([1..n],[1..n])$ tel que $(v_{\sigma(k)}/p_{\sigma(k)})_{k\in[1..n]}$ soit décroissante.
$s = 0$
$\delta =$ tableau de réels indicé par $[1..n]$ initialisé à 0
$k = 1$
Tant que $k \leq n$ et $s + p_{\sigma(k)} \leq P$ ***// invariant :*** $\boldsymbol{s = \sum_{i=1}^{n} \delta_i p_i}$
    $s \leftarrow s + p_{\sigma(k)}$
    $\delta_{\sigma(k)} \leftarrow 1$
    $k \leftarrow k + 1$
Si $k \leq n$ alors
    $\delta_{\sigma(k)} \leftarrow (P - s)/p_{\sigma(k)}$
    $s \leftarrow s + \delta_{\sigma(k)} p_{\sigma(k)}$
Renvoyer δ

On montre l'optimalité de cette stratégie par un argument d'échange.

Propriété :

Soit $n \in \mathbb{N}^*$ et $(p_i, v_i)_{i\in[1..n]} \in (\mathbb{R}^{+*} \times \mathbb{R}^{+*})^n$ telle que $(v_i/p_i)_{i\in[1..n]}$ soit strictement décroissante.
Soit de plus $P \in \mathbb{R}^{+*}$.
On note $\mathcal{S} = \{\delta \in [0,1]^n \mid \delta \cdot p \leq P\}$ et on suppose que $p \cdot \mathbb{1} > P$ (donc $\mathbb{1} \notin \mathcal{S}$).

Alors, si $\delta^* \in \underset{\delta\in\mathcal{S}}{\operatorname{argmax}}\, v \cdot \delta$, il existe $m \in [1..n]$ tel que : ***i.*** $\forall\, i \in [1..m[,\ \delta_i^* = 1$
ii. $\forall\, i \in [m+1..n], \delta_i^* = 0$
iii. $\delta_m^* = \dfrac{P - \sum_{k=1}^{m-1} p_k}{p_m}$

Preuve :
Soit $\delta^* \in \operatorname{argmax}_{\delta\in\mathcal{S}} v \cdot \delta$. Comme $\mathbb{1} \notin \mathcal{S}$, $\delta^* \neq \mathbb{1}$ donc $\{i \in [1..n] \mid \delta_i^* < 1\} \neq \emptyset$. Notons alors :

$$m = \min\{i \in [1..n] \mid \delta_i^* < 1\}$$

et montrons que cet entier m vérifie les trois propriétés ci-dessus.

i. Par définition de m, on a bien $\forall\, i \in [1..m[,\ \delta_i^* = 1$.

ii. Par l'absurde, on suppose qu'il existe $i_0 \in [m+1..n]$ tel que $\delta^*_{i_0} \neq 0$.
Posons $\varepsilon = \min\left(p_{i_0}\delta^*_{i_0}, p_m(1-\delta^*_m)\right)$: puisque $p_{i_0}\delta^*_{i_0} > 0$ par hypothèse et que $\delta^*_m < 1$, $\varepsilon > 0$ en tant que minimum de deux valeurs strictement positives. On considère alors :

$$\widehat{\delta} = \delta^* - \frac{\varepsilon}{p_{i_0}}\mathbb{1}_{i_0} + \frac{\varepsilon}{p_m}\mathbb{1}_m$$

Alors, $\forall\, i \in [1..n]$, $\widehat{\delta}_i \in [0,1]$ et donc $\widehat{\delta} \in [0,1]^n$. En effet :

- pour tout $i \notin \{i_0, m\}$, on a $\widehat{\delta}_i = \delta^*_i \in [0,1]$.
- pour i_0, on a $\widehat{\delta}_{i_0} = \delta^*_{i_0} - \varepsilon/p_{i_0} \leq \delta^*_{i_0} \leq 1$ et de plus, $\varepsilon/p_{i_0} \leq 1/p_{i_0} \times p_{i_0}\delta^*_{i_0} = \delta^*_{i_0}$ par définition de ε donc $\widehat{\delta}_{i_0} = \delta^*_{i_0} - \varepsilon/p_{i_0} \geq 0$.
- pour m, $\widehat{\delta}_m = \delta^*_m + \varepsilon/p_m \geq \delta^*_m \geq 0$ et $\varepsilon/p_m \leq 1 - \delta^*_m$, ce qui donne $\widehat{\delta}_m = \delta^*_m + \varepsilon/p_m \leq 1$.

De plus, comme $\delta^* \in \mathcal{S}$:

$$\widehat{\delta}\cdot p = (\delta^*\cdot p) + \frac{\varepsilon}{p_m}(\mathbb{1}_m\cdot p) - \frac{\varepsilon}{p_{i_0}}(\mathbb{1}_{i_0}\cdot p) = \delta^*\cdot p + \frac{\varepsilon}{p_m}p_m - \frac{\varepsilon}{p_{i_0}}p_{i_0} = \delta^*\cdot p \leq P$$

On en déduit donc que $\widehat{\delta} \in \mathcal{S}$. Par ailleurs :

$$\widehat{\delta}\cdot v = \delta^*\cdot v + \frac{\varepsilon}{p_m}(\mathbb{1}_m\cdot v) - \frac{\varepsilon}{p_{i_0}}(\mathbb{1}_{i_0}\cdot v) = \delta^*\cdot v + \varepsilon\left(\frac{v_m}{p_m} - \frac{v_{i_0}}{p_{i_0}}\right) > \delta^*\cdot v$$

puisque la suite $(v_i/p_i)_{i\in[1..n]}$ est strictement décroissante et que $i_0 > m$. Or, cela est absurde puisqu'on a choisi $\delta^* \in \operatorname{argmax}_{\delta\in\mathcal{S}}\, v\cdot\delta$.

Donc on a forcément $\forall\, i \in [m+1..n]$, $\delta^*_i = 0$.

iii. Enfin, notons :

$$w := \frac{P - \sum_{k=1}^{m-1} p_k}{p_m} = \frac{P - (\sum_{k=1}^{m-1} p_k\delta^*_k + \sum_{k=m+1}^{n} p_k\delta^*_k)}{p_m} = \frac{P - (p\cdot\delta^* - p_m\delta^*_m)}{p_m}$$

et supposons par l'absurde que $\delta^*_m \neq w$.

· Si $\delta^*_m < w$: On a $\delta^*_m p_m < P - (p\cdot\delta^* - p_m\delta^*_m)$ soit $0 < P - p\cdot\delta^*$, soit encore $p\cdot\delta^* < P$. On note alors $\varepsilon = P - \delta^*\cdot p > 0$ et on pose :

$$\widehat{\delta} = \delta^* + \frac{\varepsilon}{p_m}\mathbb{1}_m$$

On a :
$$\widehat{\delta}\cdot p = \delta^*\cdot p + \frac{\varepsilon}{p_m}(\mathbb{1}_m\cdot p) = \delta^*\cdot p + \varepsilon = P$$

Ensuite, $\forall\, k \in [1..n]\backslash\{m\}$, $\widehat{\delta}_k = \delta^*_k \in [0,1]$. De plus, $\widehat{\delta}_m = \delta^*_m + \varepsilon/p_m > \delta^*_m \geq 0$ et on a aussi :

$$\widehat{\delta}_m = \delta^*_m + \frac{P - p\cdot\delta^*}{p_m} = \frac{\delta^*_m p_m + P - p\cdot\delta^*}{p_m} = w$$

Il faut montrer que $w = \left(P - \sum_{k=1}^{m-1} p_k\right)/p_m < 1$, soit : $\sum_{k=1}^{m} p_k = \sum_{k=1}^{m-1} p_k + p_m > P$ $(*)$.
Par l'absurde, supposons que $\sum_{k=1}^{m} p_k \leq P$. Alors, on a $\mathbb{1}_{[1..m]} \in \mathcal{S}$ puisque $\mathbb{1}_{[1..m]} \in [0,1]^n$ et $\mathbb{1}_{[1..m]}\cdot p \leq P$. De plus :

$$1_{[1..m]}\cdot v = \sum_{k=1}^{m} v_k = \sum_{k=1}^{m-1} v_k\delta^*_k + \sum_{k=m+1}^{n} v_k\delta^*_k + v_m = (\delta^*\cdot v - \delta^*_m v_m) + v_m = \delta^*\cdot v + v_m(1-\delta^*_m)$$

ce qui donne, puisque $1 - \delta^*_m > 0$ par définition de m, $\mathbb{1}_{[1..m]}\cdot v > \delta^*\cdot v$. Cela contredit que δ^* est optimale puisque $\mathbb{1}_{[1..m]}$ est alors solution : on a donc bien $w < 1$, et par conséquent $\widehat{\delta} \in \mathcal{S}$.

Enfin, on a :

$$\widehat{\delta}\cdot v = \delta^*\cdot v + \frac{\varepsilon}{p_m}(\mathbb{1}_m\cdot v) = \delta^*\cdot v + \frac{v_m}{p_m}\varepsilon > \delta^*\cdot v$$

On a ainsi trouvé une solution $\widehat{\delta}$ meilleure que δ^*, ce qui est absurde car δ^* est optimale. Finalement, $\delta^*_m \geq w$.

· Si $\delta^*_m > w$: Alors, $\delta^*_m p_m > P - (p \cdot \delta^* - p_m \delta^*_m)$ soit $0 > P - p \cdot \delta^*$ et donc $p \cdot \delta^* > P$, ce qui est en contradiction avec le fait que $\delta^* \in \mathcal{S}$ et donc que $\delta^* \cdot p \leq P$. Ainsi, $\delta^*_m \leq w$

Conclusion : On a bien $\delta^*_m = w$.

Corollaire :

Sous les hypothèses et notations de la propriété précédente, $\operatorname{argmax}_{\delta \in \mathcal{S}} v \cdot \delta$ est réduite à la solution donnée par :

$$M = \min \left\{ i \in [1..n] \,\middle|\, \sum_{k=1}^{i} p_k > P \right\} \text{ c'est-à-dire } {}^t\Big(\underbrace{1\ 1\ \ldots\ 1}_{M-1}\ w\ \underbrace{0\ \ldots\ 0}_{n-M}\Big)$$

Preuve :
Soit $\delta^* \in \operatorname{argmax}_{\delta \in \mathcal{S}} \delta \cdot v$ et $m = \min \{ i \in [1..n] \,|\, \delta^*_i < 1 \}$. On veut montrer que $m = M$.
Puisque $\delta^* \in \mathcal{S}$, $\delta^* \cdot p \leq P$, ce qui donne :

$$\sum_{k=1}^{n} \delta^*_k p_k = \sum_{k=1}^{m} \delta^*_k p_k = \underbrace{\delta^*_m p_m}_{\geq 0} + \sum_{k=1}^{m-1} \delta^*_k p_k \leq P$$

et donc comme $\forall\, k \in [1..m-1]$, $\delta^*_k = 1$, on a $\sum_{k=1}^{m-1} \delta^*_k p_k \leq P$, d'où $m - 1 < M$ soit $m \leq M$.

De plus, on a montré dans la preuve précédente (*cf.* $(*)$) que $\sum_{k=1}^{m} p_k > P$. Par conséquent, $m \in \left\{ i \in [1..n] \,\middle|\, \sum_{k=1}^{i} p_k > P \right\}$, donc $m \geq M$ qui est le minimum de cet ensemble.

1.2 Le problème du tri

TRI ‖ **entrée :** $(x_i)_{i \in [1..n]} \in X^n$, où $(X, \leq)$ est un ensemble totalement ordonné
‖ **sortie :** $\sigma \in \mathcal{S}_n$ telle que $\left(x_{\sigma(i)}\right)_{i \in [1..n]}$ soit croissante pour $\leq$

Remarque : Tri est un problème d'optimisation, puisqu'il consiste en la recherche de :

$$\min_{\sigma \in \mathcal{S}_n} \sum_{i=1}^{n-1} \max(x_{\sigma(i)} - x_{\sigma(i+1)}, 0),$$

Ce minimum vaut 0 et est atteint par n'importe quel $\sigma \in \mathcal{S}_n$ solution du problème.

On propose ci-dessous un algorithme dit de tri, appelé tri par sélection, qui résout le problème Tri.

Algorithme – Tri-sélection

‖ **entrée :** $(x_i)_{i \in [1..n]}$ une famille d'éléments comparables avec $\leq$ en $\Theta(1)$, rangés dans un tableau

T = copie du tableau (*i.e.* $\forall\, i \in [1..n], T[\,i\,] = x_i$)
I = tableau identité de $[1..n]$
σ = tableau d'entiers indicé par $[1..n]$ initialisé à 0
Pour k allant de 1 à n
// invariants : $\forall\, \boldsymbol{i \in [1..n]}, \boldsymbol{T[\,i\,] = x_{I[\,i\,]}}$
$\boldsymbol{T[1..k-1]}$ ***est trié***
$\forall\, \boldsymbol{i \in [k..n]}, \boldsymbol{T[\,i\,] \geq \max\{T[\,j\,] \,|\, j \in [1..k-1]\}}$
$\forall\, \boldsymbol{i \in [1..k-1]}, \boldsymbol{T[\,i\,] = x_{\sigma[\,i\,]}}$

Trouver $i_0 \in \operatorname{argmin}_{i\in[k..n]} T[\,i\,]$
$\sigma[\,k\,] \leftarrow I[\,i_0\,]$
Échanger $T[\,k\,]$ et $T[\,i_0\,]$
Échanger $I[\,k\,]$ et $I[\,i_0\,]$
Renvoyer σ

Remarque : Cet algorithme peut être optimisé en initialisant directement σ au tableau identité, ce qui permet de ne pas avoir à créer le tableau supplémentaire I.

Propriété :

Soit I un ensemble fini non vide de cardinal n (typiquement $[1..n]$ ou $[0..n-1]$).
Soit $(x_i)_{i\in I} \in X^I$ où $(X, \leq)$ est un ensemble totalement ordonné et $i_1 \in \operatorname{argmin}_{i\in I} x_i$.
On note $\widetilde{I} = I \backslash \{i_1\}$.

Si $\widetilde{\sigma} \in \mathscr{B}ij\Big([1..n-1], \widetilde{I}\Big)$ est telle que $\Big(x_{\widetilde{\sigma}(i)}\Big)_{i\in[1..n-1]}$ est croissante, alors le prolongement σ de $\widetilde{\sigma}$ que l'on définit ci-dessous est une bijection telle que $\Big(x_{\sigma(i)}\Big)_{i\in[1..n]}$ est croissante :

$$\sigma = \begin{pmatrix} [1..n] \to I \\ 1 \mapsto i_1 \\ i \geq 2 \mapsto \widetilde{\sigma}(i-1) \end{pmatrix}$$

Preuve :
cf. annexe "tri par sélection".

1.3 À retenir sur les algorithmes gloutons

Définition (*algorithme glouton*) :

On dit d'un algorithme qu'il est glouton lorsqu'il construit une solution à un problème (d'optimisation) en prenant des décisions localement pertinentes (c'est-à-dire optimales) à chaque étape et qu'il ne revient par sur ces décisions.

Dans le cadre d'un problème d'optimisation, on dit qu'un algorithme glouton est optimal s'il renvoie toujours une solution optimale.

Exemples : L'algorithme de Huffman (*cf.* DM n°3, partie 2) et les deux algorithmes vus dans cette partie (Sac à dos fractionnaire et Tri-sélection) sont des algorithmes gloutons.

Remarque : Attention, selon les problèmes, un même algorithme peut être optimal ou non. En général, les algorithmes gloutons sont efficaces (faible complexité, a fortiori polynomiale), mais ils ne sont donc pas exacts pour les problèmes difficiles/NP-complets comme Sac à dos.

Néanmoins, ils peuvent être utiles pour obtenir rapidement la borne supérieure ou inférieure, c'est-à-dire un majorant ou un miniorant de la valeur optimale.

À retenir :

Pour savoir si un problème d'optimisation peut être résolu par un algorithme glouton, on se pose deux questions :
- La fonction-objectif est-elle décomposable comme somme de fonctions sur les sous-parties de la solution ?
- L'ensemble des solutions est-il décomposable comme produit cartésien ou au contraire défini par des contraintes liantes ?

Remarque : En pratique, cela s'articule comme suit : on cherche à exprimer $\max_{X\in\mathcal{S}} f(X)$ comme $\max_{x\in\mathcal{S}_x} f(x) + \max_{y\in\mathcal{S}_y} f(y)$.

2 Programmation dynamique

2.1 Le problème du sac à dos entier

cf. TP n°13 - "introduction à la programmation dynamique".

2.2 Plus long sous-mot commun

Dans cette section, on fixe Σ un alphabet (i.e. un ensemble fini et non vide de symboles).

Rappel (*sous-mot*) :

Soit $(u,v)\in(\Sigma^*)^2$. Notons $n=|u|$ et $m=|v|$. On dit que u est un sous-mot de v et on note $u \preccurlyeq v$ ssi il existe $\varphi\in\mathscr{F}([1..n],[1..m])$ strictement croissante, telle que :

$$u = v_{\varphi(1)}v_{\varphi(2)}...v_{\varphi(n)} \quad \text{c'est-à-dire} \quad \forall\, i\in[1..n],\ u_i = v_{\varphi(i)}$$

Le problème du Plus long sous-mot commun, que nous abrègerons en PLSMC, consiste donc à trouver la longueur maximale d'un sous-mot commun à deux mots donnés :

PLSMC ‖ **entrée :** $(u,v)\in(\Sigma^*)^2$
‖ **sortie :** $\max\{|w|\,|\,w\preccurlyeq u \text{ et } w\preccurlyeq v\}$ ou $w^*\in\operatorname{argmax}\{|w|\,|\,w\preccurlyeq u \text{ et } w\preccurlyeq v\}$

Soit $(u,v)\in(\Sigma^*)^2$, on note $n=|u|$ et $m=|v|$. Pour $(i,j)\in[0..n]\times[0..m]$, on pose :

$$\mathcal{L}_{i,j} = \max\left\{|w| \;\middle|\; \underbrace{w\in\Sigma^*,\ \begin{matrix} w\preccurlyeq u_1...u_i \\ w\preccurlyeq v_1...v_j\end{matrix}}_{:=\mathcal{S}_{i,j}}\right\} = \max_{w\in\mathcal{S}_{i,j}}|w|$$

On obtient alors via la propriété suivante une manière de résoudre PLSMC.

Propriétés :

On a alors : ***i.*** $\forall\, i\in[0..n],\ \mathcal{L}_{i,0}=|\varepsilon|=0$
ii. $\forall\, j\in[0..m],\ \mathcal{L}_{0,j}=0$
iii. $\forall(i,j)\in[0..n]\times[0..m],\ \mathcal{L}_{i,j} = \begin{cases}\mathcal{L}_{i-1,j-1}+1 \text{ si } u_i=v_j\\ \max(\mathcal{L}_{i-1,j},\mathcal{L}_{i,j-1}) \text{ sinon}\end{cases}$

De plus, $\text{PLSMC}(u,v)=\mathcal{L}_{n,m}$.

Avant de commencer la preuve de ces propriétés, il nous faut le lemme qui suit.

Lemme : Soit $(u,v)\in(\Sigma^*)^2$, on note $n=|u|$ et $m=|v|$. Si $u\preccurlyeq v$ et $u_m\neq v_m$, alors $u\preccurlyeq v_1...v_{m-1}$.

Preuve de la propriété :
i. Soit $i \in [0..n]$.
On a : $\mathcal{L}_{i,0} = \max\{|w| \,|\, w \in \Sigma^*,\, w \preccurlyeq u_1...u_i,\, w \preccurlyeq \varepsilon\} \leq \max\{|w| \,|\, w \in \Sigma^*,\, w \preccurlyeq \varepsilon\}$ par relaxation, ce qui donne $\mathcal{L}_{i,0} \leq \max\{|\varepsilon|\} = |\varepsilon| = 0$ puisque le seul sous-mot de ε est lui-même. Or, on a aussi $\varepsilon \preccurlyeq u_1...u_i$ et $\varepsilon \preccurlyeq \varepsilon$ donc $|\varepsilon| \leq \mathcal{L}_{i,0}$ soit $0 \leq \mathcal{L}_{i,0}$.

D'où $\mathcal{L}_{i,0} = 0$ par double inégalité.

ii. La démonstration se fait sur le même principe que pour la première propriété.

iii. Soit $(i,j) \in [1..n] \times [1..m]$.
· Supposons que $u_i = v_j$. Alors, on a :

$$\begin{aligned}
\mathcal{L}_{i,j} &= \max\left\{ |w| \,\middle|\, w \in \Sigma^*,\, \begin{array}{l} w \preccurlyeq u_1...u_{i-1}u_i \\ w \preccurlyeq v_1...v_{j-1}v_j \end{array} \right\} \\
&\geq \max\left\{ |w| \,\middle|\, w \in \Sigma^*,\, w_{|w|} = u_i,\, \begin{array}{l} w \preccurlyeq u_1...u_{i-1}u_i \\ w \preccurlyeq v_1...v_{j-1}v_j \end{array} \right\} \\
&= \max\left\{ |x| + 1 \,\middle|\, x \in \Sigma^*,\, \begin{array}{l} x \preccurlyeq u_1...u_{i-1} \\ x \preccurlyeq v_1...v_{j-1} \end{array} \right\} \\
&= \mathcal{L}_{i-1,j-1} + 1
\end{aligned}$$

puisque l'ensemble de la deuxième ligne est inclus dans celui de la première.

Ensuite, montrons que $\mathcal{L}_{i,j} - 1 \leq \mathcal{L}_{i-1,j-1}$: il existe $w \in \mathcal{S}_{i,j}$ tel que $|w| = \mathcal{L}_{i,j}$.
· Si $w = \varepsilon$, alors $\mathcal{L}_{i,j} = |\varepsilon| = 0$, or $u_i = v_j \in \mathcal{S}_{i,j}$ donc $\mathcal{L}_{i,j} \geq |u_i| = 1$, absurde.
· Si $w \neq \varepsilon$, alors $p = |w| > 0$. On pose alors $x = w_1w_2...w_{p-1}$ et on a $|x| = p-1 = |w|-1$.
Montrons que $x \in \mathcal{S}_{i-1,j-1}$:
· si $w_p = u_i = v_j$, alors $x \preccurlyeq u_1...u_{i-1}$ et $x \preccurlyeq v_1...v_{j-1}$ (par identification).
· si $w_p \neq u_i$ (*i.e.* $w_p \neq v_j$), alors d'après le lemme $w \preccurlyeq u_1...u_{i-1}$ et $w \preccurlyeq v_1...v_{j-1}$, donc comme $x \preccurlyeq w$, par transitivité on en déduit $x \preccurlyeq u_1...u_{i-1}$ et $x \preccurlyeq v_1...v_{j-1}$.
Comme dans les deux cas $x \in \mathcal{S}_{i-1,j-1}$, on a $\mathcal{L}_{i,j} - 1 = |w| - 1 = |x| \leq \mathcal{L}_{i-1,j-1}$.

· Supposons à présent que $u_i \neq v_j$.
Montrons que $\mathcal{L}_{i,j} = \max(\mathcal{L}_{i-1,j}, \mathcal{L}_{i,j-1})$ en montrant que $\mathcal{S}_{i,j} = \mathcal{S}_{i-1,j} \cup \mathcal{S}_{i,j-1}$.

Soit $w \in \mathcal{S}_{i,j}$, on pose $p = |w|$.
· si $w_p = u_i$, alors $w_p \neq v_j$, or $w \preccurlyeq v_1...v_j$ donc d'après le lemme $w \preccurlyeq v_1...v_{j-1}$, et comme on a de plus $w \preccurlyeq u_1...u_i$, on en déduit que $w \in \mathcal{S}_{i,j-1}$
· si $w_p \neq u_i$, alors comme $w \preccurlyeq u_1....u_i$, le lemme donne que $w \preccurlyeq u_1...u_{i-1}$ donc comme on a aussi $w \preccurlyeq v_1...v_j$, on en déduit $w \in \mathcal{S}_{i-1,j}$.
D'où l'inclusion $\mathcal{S}_{i,j} \subseteq \mathcal{S}_{i-1,j} \cup \mathcal{S}_{i,j-1}$.

Ensuite, soit $w \in \mathcal{S}_{i-1,j}$. Par définition, $w \preccurlyeq u_1...u_{i-1} \preccurlyeq u_1...u_i$ (car un préfixe d'un mot en est un sous-mot) et $w \preccurlyeq v_1...v_j$ donc $w \in \mathcal{S}_{i,j}$. On a donc $\mathcal{S}_{i-1,j} \subseteq \mathcal{S}_{i,j}$, ainsi que $\mathcal{S}_{i,j-1} \subseteq \mathcal{S}_{i,j}$ (par la même démonstration), donc $\mathcal{S}_{i-1,j} \cup \mathcal{S}_{i,j-1} \subseteq \mathcal{S}_{i,j}$.

On conclut finalement par double inclusion. Ainsi :

$$\begin{aligned}
\mathcal{L}_{i,j} &= \max\left\{ |w| \,\middle|\, w \in \mathcal{S}_{i,j} \right\} \\
&= \max\left\{ |w| \,\middle|\, w \in \mathcal{S}_{i-1,j} \cup \mathcal{S}_{i,j-1} \right\} \\
&= \max\left(\max\left\{ |w| \,\middle|\, w \in \mathcal{S}_{i-1,j} \right\}, \max\left\{ |w| \,\middle|\, w \in \mathcal{S}_{i,j-1} \right\} \right) \\
&= \max(\mathcal{L}_{i-1,j}, \mathcal{L}_{i,j-1})
\end{aligned}$$

Exemple : Pour $u =$ GIRAFE et $v =$ GRAFITI, on a $n = 6$, $m = 7$.
On peut appliquer la méthode de résolution présentée précédemment en rassemblant l'ensemble des sous-problèmes dans un tableau, comme suit, où pour chaque case correspondant à un couple (i, j) donné :

- une flèche ascendante indique qu'on a pris $\mathcal{L}_{i,j} = \mathcal{L}_{i-1,j}$
- une flèche vers la gauche indique qu'on a pris $\mathcal{L}_{i,j} = \mathcal{L}_{i,j-1}$
- une flèche diagonale indique qu'on a pris $\mathcal{L}_{i,j} = \mathcal{L}_{i-1,j-1} + 1$
- cette case est grisée si $u_i = v_j$ (ce qui correspond à avoir une flèche diagonale).

$\mathcal{L}_{i,j}$			G	R	A	F	I	T	I
	i \ j	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$
I	2	0	$\uparrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\nwarrow 2$
R	3	0	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\leftarrow 2$
A	4	0	$\uparrow 1$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$	$\leftarrow 3$
F	5	0	$\uparrow 1$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\leftarrow 4$	$\leftarrow 4$	$\leftarrow 4$
E	6	0	$\uparrow 1$	$\uparrow 2$	$\uparrow 3$	$\uparrow 4$	$\leftarrow 4$	$\leftarrow 4$	$\leftarrow 4$

2.3 À retenir sur la programmation dynamique et la mémoïsation

Lorsqu'un algorithme récursif résout un problème en faisant appel à la solution de plusieurs sous-problèmes, une implémentation naïve risque de calculer de nombreuses fois les mêmes sous-problèmes. Dans certains cas, cela conduit à un algorithme de complexité inutilement exponentielle.

Définition (*mémoïsation*) :

La mémoïsation permet de pallier l'inefficacité ci-dessus : il s'agit de garder en mémoire à chaque appel la valeur cherchée. Plus précisément, on stocke les associations entre des paramètres caractérisant une instance ou sous-instance du problème et la valeur de la solution associée.

On utilise pour cela une structure de dictionnaire, qui peut dans de nombreux cas être implémentée par un tableau, un ARN, ou encore un tas de hachage.

Remarque : Il faut prendre soin d'utiliser les valeurs déjà calculées disponibles avant de lancer les appels récursifs.

Définition (*programmation dynamique*) :

La programmation dynamique repose sur la même idée de stockage des valeurs de sous-problèmes pour résoudre un problème mais cette approche n'est pas récursive : on calcule toutes les valeurs d'une famille de sous-problèmes des plus petits aux plus grands (selon un ordre à définir).

On effectue ci-dessous une comparaison entre ces deux paradigmes algorithmiques :

Mémoïsation
Avantage : *on calcule seulement les valeurs nécessaires*
Inconvénients : – *la complexité spatiale d'une implémentation par tableau ne peut être réduite.*
– *possiblement plus difficile à programmer*

Programmation dynamique
Avantages : – *permet parfois une réduction de la complexité spatiale*
– *facile à coder*
Inconvénient : *potentiellement plus d'appels*

2.4 Autres remarques et exemples

3 Diviser pour régner (divide-and-conquer)

3.1 L'exemple du tri-fusion

On s'intéresse une nouvelle fois au problème du Tri : on propose ici une autre approche à l'aide de l'algorithme de tri suivant, appelé tri-fusion.

Algorithme – Tri-fusion

entrée : T un tableau
d, f deux indices (valides pour ce tableau)

$n = \max(f - d + 1, 0)$ (*i.e.* Card $[d..f]$)) *// ou mettre en hypothèse que* $d \leq f$
Si $n = 0$ ou 1 alors
retourner $T[\,d..f\,]$
Sinon
$m \leftarrow \lfloor n/2 \rfloor$
$T_1 \leftarrow$ Tri-fusion $(T, d, m-1)$
$T_2 \leftarrow$ Tri-fusion (T, m, f)
retourner Interclassement (T_1, T_2)

où la fonction Interclassement a été définie au préalable comme suit :

Algorithme – Interclassement

entrée : T_1, T_2 deux tableaux triés
hypothèses : les tableaux sont indexés à partir de 0

$n, m =$ taille (T_1), taille (T_2)
$i, j = 0, 0$
$T =$ un tableau de taille $n + m$
Tant que $i < n$ et $j < m$
// invariant : $T[\,0..i+j\,[$ *est trié*
$\{\!\{T_1[\,k\,] \,|\, k \in [0..i[\}\!\} \cup \{\!\{T_2[\,k\,] \,|\, k \in [0..j[\}\!\} = \{\!\{T[\,k\,] \,|\, k \in [0..i+j[\}\!\}$
Si $T_1[\,i\,] \leq T_2[\,j\,]$ alors
$T[\,i+j\,] \leftarrow T_1[\,i\,]$
$i \leftarrow i + 1$
Sinon
$T[\,i+j\,] \leftarrow T_2[\,j\,]$
$j \leftarrow j + 1$
Tant que $i < n$
$T[\,i+j\,] \leftarrow T_1[\,i\,]$
$i \leftarrow i + 1$
Tant que $j < m$
$T[\,i+j\,] \leftarrow T_2[\,j\,]$
$j \leftarrow j + 1$
Retourner T

Propriété :

Interclassement fait au plus $|T_1| + |T_2| - 1$ comparaisons.

Preuve :
Il y a moins de comparaisons que de tours de la première boucle "Tant que" ; or, ce nombre de tours de boucle est inférieur à $n+m$ car initialement, $i+j=0$, à chaque tour $i+j$ augmente de 1, et par la condition de boucle, $i+j \leq n+m-2$).

Exercice : Exhiber une famille de pire cas qui atteint cette borne.

On s'intéresse à présent à la complexité de Tri-fusion : pour $n \in \mathbb{N}$, on note C_n le nombre maximal de comparaisons effectuées pour trier un sous-tableau de taille n. Alors, on a:

- $C_0 = 0$
- $C_1 = 0$
- $\forall\, n \in \mathbb{N}^*$, $C_n = \overbrace{0}^{\text{diviser}} + \overbrace{C_{\lfloor n/2 \rfloor} + C_{\lceil n/2 \rceil}}^{\text{régner}} + \overbrace{\lfloor n/2 \rfloor + \lceil n/2 \rceil - 1}^{\text{combiner}} = C_{\lfloor n/2 \rfloor} + C_{\lceil n/2 \rceil} + n - 1$

Propriété :

La complexité de Tri-fusion est en $\Theta(n \log n)$.

Preuve :
Montrons par récurrence forte sur $n \in \mathbb{N}^*$ la propriété

$$\mathcal{P}_n : \text{“}C_n \leq n \log_2 n\text{”}$$

- Pour $n = 1$, on a $C_1 = 0$ et $1 \times \log_2(1) = 0$ donc $\mathcal{P}_1$ est vraie.
- Soit $n \in \mathbb{N}^*$. On suppose $\forall\, k \in [1..n]$, $\mathcal{P}_k$ vraie.

· Si $n+1$ est pair, disons $n+1 = 2p$, alors :

$$\begin{aligned} C_{n+1} &= 2C_p + (n+1-1) \\ &\leq 2\,(p \log_2(p)) + n \text{ d'après } \mathcal{P}_p \\ &= (n+1)(\log_2(n+1) - \log_2 2) + n \\ &= (n+1)\log_2(n+1) - (n+1) - n \\ &\leq (n+1)\log_2(n+1) \end{aligned}$$

· Si $n+1$ est impair, alors $\left\lfloor \frac{n+1}{2} \right\rfloor = \frac{n}{2}$ et $\left\lceil \frac{n+1}{2} \right\rceil = \frac{n}{2} + 1$. Ainsi :

$$\begin{aligned} C_{n+1} &= C_{n/2} + C_{n/2+1} + ((n+1)-1) \\ &\leq \frac{n}{2}(\log_2(n) - 1) + \left(\frac{n}{2} + 1\right)(\log_2(n+2) - 1) + n \\ &= \frac{n}{2}(\log_2(n) + \log_2(n+2)) - \frac{n}{2} - \frac{n}{2} + n + \log_2(n+2) - 1 \\ &\leq n \log_2\left(\frac{n+(n+2)}{2}\right) + \log_2(n+2) - 1 \text{ par concavité de } t \mapsto \log_2 t \\ &= n \log_2(n+1) + \log_2\left(\frac{n+2}{2}\right) \end{aligned}$$

Or, pour $n > 0$, $\log_2\left(\frac{n+2}{2}\right) \leq \log_2(n+1) \iff \frac{n+2}{2} \leq n+1 \iff n+2 \iff 1 \leq 2$.

Ainsi, on a bien $C_{n+1} \leq (n+1)\log_2(n+1)$ dans tous les cas, d'où $\mathcal{P}_{n+1}$.