Chapitre 7

Tentative de réponse à la **NP**-*complétude*

Hugo Salou MPI*

Dernière mise à jour le 13 mai 2023

Table des matières

0 Motivation

On considère le problème **NP**-complet du *voyageur de commerce* : étant donné un graphe pondéré G quel est le tour de longueur [1] minimale *i.e.* quelle est la permutation de sommets telle que la longueur totale est minimale.

On se ramène à un problème de décision : étant donné une constante $K \in \mathbb{R}$, existe-t-il un chemin de longueur inférieure à K.

Un algorithme glouton, allant d'un sommet à son voisin le plus proche, ne permet pas de résoudre ce problème en complexité polynômiale.

On ne cherche plus le « tour optimal » mais on cherche une solution proche : on veut trouver une constante ρ telle que, quelque soit l'entrée, le chemin obtenu est de longueur inférieure à ρ fois la longueur optimale.

1 Problèmes d'optimisation

Dans un premier temps, on s'intéresse à un problème où l'on cherche à minimiser quelque chose. On réalise la transformation réalisée dans la partie précédente : étant donné un seuil K, on est ce que la valeur est inférieure à K. On se ramène donc à un problème de décision.

Définition : Soit $Q \subseteq \mathcal{E} \times \mathcal{S}$ un problème. Soit $\mathrm{opt} \in \{\min, \max\}$. On dit que Q est un *problème d'*opt*imisation* (*i.e.* problème de minimisation, maximisation), si pour toute entrée $e \in \mathcal{E}$, il existe

- un ensemble $\mathrm{sol}(e)$ de solutions,
- une fonction $c_e : \mathrm{sol}(e) \to \mathbb{R}^+$,

tels que $c_e^\star = \mathrm{opt}\{c_e(s) \mid s \in \mathrm{sol}(e)\}$ est bien défini, et

$$\forall s \in \mathrm{sol}(e), \qquad (e, s) \in Q \implies c_e(s) = c_e^\star.$$

On nomme :

- $\mathrm{sol}(e)$ l'*ensemble des solutions pour l'entrée* e,
- c_e *la fonction objectif*,
- $c_e^\star$ la *valeur* opt*imale* (minimale ou maximale),
- pour une solution $s \in \mathrm{sol}(e)$, $c_e(s)$ est appelée la *valeur de la solution*.

On appelle *solution* opt*imale* <u>une</u> solution de valeur optimale.

Exemple :
On considère le problème

$$\begin{cases} \textbf{Entrée} & : G = (S, A) \text{ un graphe orienté fortement connexe, } s \in S, \text{ et } p \in S \\ \textbf{Sortie} & : \text{un plus court chemin (en nombre d'arcs) de } s \text{ à } p \text{ dans } G. \end{cases}$$

Soit l'entrée ci-dessous.

1. poids des arrêtes total

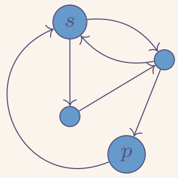

Figure 1 – Entrée du problème du plus court chemin

L'ensemble $\mathrm{sol}(e)$ est l'ensemble (infini) des chemins de s à p, et $c_e(\gamma) = |\gamma|$ (la longueur du chemin γ). On vérifie qu'il existe un chemin de s à p, donc $\{c_e(s) \mid s \in \mathrm{sol}(e)\}$ est une partie de $\mathbb{N}$ non vide, elle admet donc un minimum.

Définition : Le *problème de décision associé* à un problème d'optimisation est le problème obtenu en ajoutant une constante aux entrées et en demandant en sortie s'il est possible de dépasser cette constante.

Exemple :
Étant donné le problème d'optimisation

$$Q_{\mathrm{O}} : \begin{cases} \textbf{Entrée} & : e \in \mathscr{E}_{Q_{\mathrm{O}}} \\ \textbf{Sortie} & : \arg\mathrm{opt}_{s \in \mathrm{sol}(e)}\, c_e(s), \end{cases}$$

on définit le problème de décision associé

$$Q : \begin{cases} \textbf{Entrée} & : e \in \mathscr{E}_{Q_{\mathrm{O}}},\ K \in \mathbb{R}^+ \\ \textbf{Sortie} & : \text{existe-t-il } s \in \mathrm{sol}(e) \text{ tel que } c_e(s) \bowtie K \end{cases}$$

avec $\bowtie = \leqslant$ si $\mathrm{opt} = \min$ et $\bowtie = \geqslant$ si $\mathrm{opt} = \max$.

Exemple :
Avec l'exemple précédent (plus court chemin), le problème de décision associé est

$$\begin{cases} \textbf{Entrée} & : G = (S, A) \text{ un graphe orienté fortement connexe, } (p, s) \in S^2, \text{ et } K \in \mathbb{R}^+ \\ \textbf{Sortie} & : \text{Existe-t-il un chemin de } s \text{ à } p \text{ dans } G \text{ de longueur inférieure ou égale à } K\,? \end{cases}$$

Exemple :
On considère le problème Knapsack de décision défini comme

$$\begin{cases} \textbf{Entrée} & : \text{Un entier } n \in \mathbb{N},\ (p_1, \ldots, p_n) \in (\mathbb{N}^\star)^n,\ (v_1, \ldots, v_n) \in (\mathbb{N}^\star)^n,\ P \in \mathbb{N} \text{ et } K \in \mathbb{N} \\ \textbf{Sortie} & : \text{Existe-t-il } I \subseteq [\![1, n]\!] \text{ telle que } \sum_{i \in I} p_i \leqslant P \text{ et } \sum_{i \in I} v_i \geqslant K\,? \end{cases}$$

Le problème d'optimisation associé est $\textsc{Knapsack}_{\mathrm{O}}$ défini comme

$$\begin{cases} \textbf{Entrée} & : \text{Un entier } n \in \mathbb{N},\ (p_1, \ldots, p_n) \in (\mathbb{N}^\star)^n,\ (v_1, \ldots, v_n) \in (\mathbb{N}^\star)^n, \text{ et } P \in \mathbb{N} \\ \textbf{Sortie} & : I \subseteq [\![1, n]\!] \text{ tel que } \sum_{i \in I} p_i \leqslant P \text{ et maximisant } \sum_{i \in I} v_i. \end{cases}$$

Remarque :
Soit Q_O un problème d'optimisation et Q le problème de décision associé. Étant donné un algorithme $\mathscr{A}_\mathrm{O}$ pour Q_O, on fabrique l'algorithme $\mathscr{A}$ suivant résolvant Q.

Algorithme 1 Solution à un problème de seuil

Entrée e une entrée de Q_O et K un seuil

retourner $c_e(\mathscr{A}_\mathrm{O}) \overset{?}{\bowtie} K$ ▷ *où* $\bowtie$ *est* $\geqslant$ *pour si* opt *est* max, *et* $\leqslant$ *si* opt *est* min

Ainsi, le problème Q_O est plus difficile à résoudre que le problème Q de décision associé. Alors, lorsque le problème de décision Q associé à un problème d'optimisation S_O est **NP**-difficile, c'est mal engagé.

2 Algorithmes d'approximations

Remarque (Vocabulaire) :
On fixe dans la suite un problème d'optimisation Q, on note $\mathrm{OPT}(e)$ la valeur optimale pour une entrée e.

Définition (Algorithme d'approximation pour un problème de maximisation) : On dit d'un algorithme $\mathscr{A} : \mathscr{E}_Q \to \mathbb{R}^+$ qu'il approxime un problème Q de maximisation avec un ratio d'approximation $\rho < 1$ dès lors que

$$\forall e \in \mathscr{E}_Q, \quad \mathscr{A}(e) \geqslant \rho \cdot \mathrm{OPT}(e).$$

On dit alors que l'algorithme $\mathscr{A}$ est une *ρ-approximation* (standard).

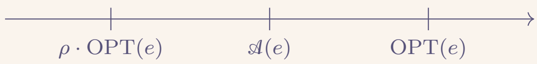

Figure 2 – Algorithme d'approximation pour un problème de maximisation

Définition (Algorithme d'approximation pour un problème de minimisation) : On dit d'un algorithme $\mathscr{A} : \mathscr{E}_Q \to \mathbb{R}^+$ qu'il approxime un problème Q de minimisation avec un ratio d'approximation $\rho > 1$ dès lors que

$$\forall e \in \mathscr{E}_Q, \quad \mathscr{A}(e) \leqslant \rho \cdot \mathrm{OPT}(e).$$

On dit alors que l'algorithme $\mathscr{A}$ est une *ρ-approximation* (standard).

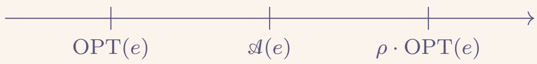

Figure 3 – Algorithme d'approximation pour un problème de minimisation

Dans la définition suivante, on suppose connu une fonction $\mathrm{Pire}(e)$ donnant la pire valeur de solution pour une entrée e.

Définition : On dit qu'un algorithme $\mathscr{A} : \mathscr{E}_Q \to \mathbb{R}$ est une *ρ-approximation différentielle* dès lors que

$$\frac{\left|\mathscr{A}(e) - \mathrm{Pire}(e)\right|}{\left|\mathrm{Pire}(e) - \mathrm{OPT}(e)\right|} \geqslant \rho.$$

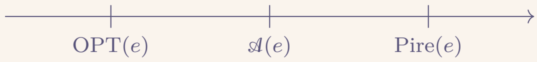

FIGURE 4 – ρ-approximation différentielle

REMARQUE :
Dans le cadre d'un problème de minimisation, on ne calcule pas $\mathrm{OPT}(e)$ en général. On minore $\mathrm{OPT}(e)$, et alors

$$\frac{\mathscr{A}(e)}{\mathrm{OPT}(e)} \leqslant \underbrace{\frac{\mathscr{A}(e)}{B}}_{\rho}.$$

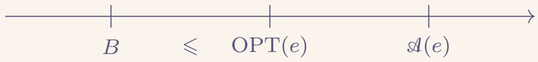

FIGURE 5 – Calcul de $\mathrm{OPT}(e)$

EXEMPLE :
Soit $C \in \mathbb{N}^\star$. On rappelle le problème STABLE, restreint à un graphe de degré maximal C :

$$\textsc{Stable} : \begin{cases} \textbf{Entrée} & : G = (S, A) \text{ une graphe tel que } 0 \neq \varDelta(G) \leqslant C \\ \textbf{Sortie} & : \text{ un stable de } G \text{ de cardinal maximal} \end{cases}$$

où $X \subseteq S$ est un stable si pour tout $(u, v) \in X^2$, $\{u, v\} \notin A$, et $\varDelta(G) = \max_{v \in S} \deg_G(v)$ est le degré du graphe.

Algorithme 2 Algorithme glouton de recherche de stables

Entrée $G = (S, A)$ un graphe
- $S' \leftarrow \varnothing$
- **tant que** $S \neq \varnothing$ **faire**
  - $v^\star = \arg\min_{v \in S} \deg_G(v)$ ▷ *les degrés sont modifiés à chaque itération*
  - $S' \leftarrow S' \cup \{v^\star\}$
  - $S \leftarrow S \setminus \big(\{v^\star\} \cup \mathrm{voisins}(v^\star)\big)$
  - $A \leftarrow$ restriction de A à S
- **retourner** S'

Cet algorithme n'est pas correct, la figure ci-après en est un contre-exemple.

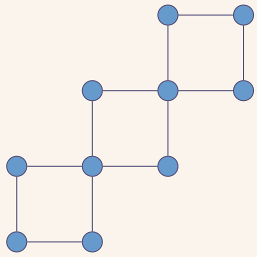

Figure 6 – Contre-exemple à l'algorithme 2

Propriété : L'algorithme 2 est une $\frac{1}{\Delta(G)}$-approximation.

Preuve :
Soit S' la réponse de l'algorithme. Soit $S^\star$ la solution optimale. On a, par terminaison de l'algorithme

$$\forall v \in S \setminus S, \exists v' \in S',\ \{v, v'\} \in A$$

car l'algorithme s'arrête. En particulier, $\forall v^\star \in S^\star \setminus S',\ \exists v' \in S',\ \{v, v'\} \in A$. Or, $S^\star$ est stable donc, si $v^\star \in S^\star$ et $v' \in S'$ tels que $\{v^\star, v' \in A\}$, alors $v' \notin S^\star$. D'où,

$$\forall v^\star \in S^\star \setminus S',\ \exists v' \in S' \setminus S^\star,\ \{v^\star, v'\} \in A.$$

Par définition de degré d'un graphe, on a $|S^\star \setminus S'| \leqslant \Delta(G)\,|S' \setminus S^\star|$ donc

$$\begin{aligned}
|S^\star| &= |S^\star \cap S'| + |S^\star \setminus S'| \\
&\leqslant |S^\star \cap S'| + \Delta(G)\,|S' \setminus S^\star| \\
&\leqslant \Delta(G)\,|S^\star \cap S'| + \Delta(G)\,|S' \setminus S^\star| \\
&\leqslant \Delta(G)\,|S'|.
\end{aligned}$$

□

Remarque :
Cette preuve ne fait pas d'hypothèses sur le résultat de l'algorithme. Tout algorithme répondant au problème est une $\frac{1}{\Delta(G)}$-approximation.

Exemple :
On appelle *couverture par sommets* d'un graphe $G = (S, A)$ la donnée d'un ensemble $X \subseteq S$ tel que

$$\forall \{u, v\} \in A, \quad u \in X \text{ ou } v \in X.$$

Exemple :
L'ensemble ● est une couverture par sommets du graphe ci-dessous.

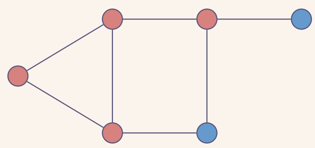

Figure 7 – Exemple de couverture par sommets

On considère le problème

$$\begin{cases} \textbf{Entrée} & : \ G = (S, A) \text{ un graphe} \\ \textbf{Sortie} & : \ \text{une couverture de cardinal maximal.} \end{cases}$$

Ce problème peut-être résolu à l'aide du calcul de couplages maximal.

Algorithme 3 Calcul d'un couplage maximal (CouplageMaximal)

Entrée $G = (S, A)$ un graphe
Sortie C un couplage maximal, *non nécessairement maximum*

$C \leftarrow \varnothing$
tant que $\exists\{u, v\} \in A$, u libre dans C et v libre dans C **faire**
  Soit $\{u, v\}$ une telle arrête.
  $C \leftarrow C \cup \{\{u, v\}\}$
retourner C

L'algorithme retourne un couplage maximal d'après la négation de la condition de boucle. On répond donc au problème avec l'algorithme ci-dessous.

Algorithme 4 Approximation de couverture par sommets

Entrée $G = (S, A)$ un graphe
Sortie Une couverture par sommets

$C \leftarrow \text{CouplaxeMaximal}(G)$
retourner $\{u \in S \mid \exists v \in S,\ \{u, v\} \in C\}$.

L'algorithme retourne une couverture : soit X la valeur retournée pour une entrée $G = (S, A)$. Soit $\{u, v\} \in A$. Si $u \notin X$ et $v \notin X$, alors le couplage C calculé pour l'algorithme n'est pas maximal, on peut y ajouter $\{u, v\}$. Montrons que l'algorithme $\mathscr{A}$ est une 2-approximation du problème « couverture par sommets. »

$$\forall G \in \mathscr{C}, \quad \mathscr{A}(G) \leqslant 2\,\mathrm{OPT}(G).$$

Soit $G \in \mathscr{C}$. Soit X la couverture par sommets calculé par $\mathscr{A}$ sur G, et C le couplage calculé par cet algorithme. Soit $X^\star$ la couverture par sommets optimale. Soit donc

$$\begin{aligned} \varphi : \quad & C \longrightarrow X^\star \\ & \{u, v\} \longmapsto \begin{cases} u & \text{si } u \in X^\star \\ v & \text{si } v \in X^\star \end{cases} \end{aligned}$$

Soit $(c_1, c_2) \in C^2$, tels que $\varphi(c_1) = \varphi(c_2)$, alors c_1 et c_2 partagent un sommet, ce qui est absurde (*c.f.* définition de couplage). Donc φ est injective, d'où $|C| \leqslant |X^\star|$. Or, $\mathscr{A}(X) = |X| = 2|C| \leqslant 2\,|X^\star| \leqslant 2\,\mathrm{OPT}(X)$.

3 *Branch and Bound* — Séparation et évaluation

Branch and Bound n'est pas un algorithme, mais une famille d'algorithmes, similairement au algorithmes diviser pour régner. Ces algorithmes répondent à des problèmes de maximisation. Les algorithmes *Branch and Bound* sont des algorithmes enrichit de trois fonctions :

— une fonction `branch` de branchement, *i.e.* découpage en sous-problèmes,
— une fonction `valeur` donnant un résultat, pas forcément optimal, *i.e.* elle associe une solution partielle à une solution,
— une fonction `bound` donnant un majorant de la solution optimale, complétant cette solution partielle.

Avec les deux dernières fonctions, on borne la valeur de la solution optimale.

Exemple (pl et plne) :
c.f. dm4.

Exemple :
On considère le problème Knapsack :

$$\begin{cases}\textbf{Entrée} & : n \in \mathbb{N},\ (v_i)_{i \in [\![1,n]\!]} \in (\mathbb{N}^\star)^n,\ (w_i)_{i \in [\![1,n]\!]} \in (\mathbb{N}^\star)^n,\ \text{et } P \in \mathbb{N}^\star \\ \textbf{Sortie} & : \text{une allocation } I \text{ maximale d'objets de somme de poids } (w_i)_{i \in I} \text{ inférieure ou égale à } P.\end{cases}$$

Dans toute la suite de l'exemple, les $(v_i)_{i \in [\![1,n]\!]}$ et $(w_i)_{i \in [\![1,n]\!]}$ sont triés par v_i/w_i décroissants.

Tentative 1. Algorithme glouton.

Algorithme 5 Algorithme glouton $\mathscr{G}^{\mathbb{N}}$ répondant au problème Knapsack

```
I ← ∅
S ← 0
pour i ∈ ⟦1, n⟧ faire
    si S + w_i ⩽ P alors
        I ← I ∪ {i}
        S ← S + w_i
retourner I
```

Cet algorithme ne donne pas toujours une solution optimale, voici un contre-exemple : $P = 3$, $(v_i) = (1, 2)$ et $(w_i) = (1, 3)$. L'algorithme renvoie $\mathscr{G}^{\mathbb{N}}(E_1) = 1$ mais la solution optimale $\mathrm{OPT}(E_1) = 2$, pour l'entrée E_1. On peut compléter une solution partielle à l'aide de cet algorithme, on a donc défini la fonction `valeur`.

Tentative 2. On résout le problème associé dans $\mathbb{R}$, définit ci-dessous :

$$\textsc{Knapsack}_{\mathbb{R}} \begin{cases}\textbf{Entrée} & : n \in \mathbb{N},\ (v_i)_{i \in [\![1,n]\!]} \in (\mathbb{N}^\star)^n,\ (w_i)_{i \in [\![1,n]\!]} \in (\mathbb{N}^\star)^n,\ \text{et } P \in \mathbb{N}^\star \\ \textbf{Sortie} & : \arg\max_{x \in S} \sum_{i=1}^n x_i v_i\end{cases}$$

où $S = \left\{ (x_i)_{i \in [\![1,n]\!]} \in [0,1]^n \;\middle|\; \sum_{i=1}^n x_i w_i \leqslant P \right\}$. On résout ce problème à l'aide d'un algorithme glouton.

Algorithme 6 Algorithme glouton $\mathscr{G}^{\mathbb{R}}$ répondant au problème $\textsc{Knapsack}_{\mathbb{R}}$

```
S ← 0
i ← 1
x ← (0)_{i ∈ ⟦1,n⟧}
tant que i ⩽ n et S + w_i < P faire
    S ← S + w_i
    x_i ← 1
    i ← i + 1
si i ⩽ n alors
    x_i ← (P − S)/2
    S ← P
retourner x
```

En notant $\mathrm{OPT}(e)$ une solution optimale à Knapsack, et $\mathrm{OPT}^{\mathbb{R}}(e)$ une solution optimale à $\textsc{Knapsack}_{\mathbb{R}}$, on a $\forall e \in \mathscr{E}, \quad \mathrm{OPT}(e) \leqslant \mathrm{OPT}^{\mathbb{R}}(e)$. De plus, à faire à la maison, le glouton $\mathscr{G}^{\mathbb{R}}$ donne la solution optimale : on a $\mathrm{OPT}^{\mathbb{R}}(e) = \mathscr{G}^{\mathbb{R}}(e)$.

On branche sur la partie fractionnaire. On considère l'entrée définie dans la table ci-après, avec $P = 20$.

i	1	2	3	4	5	6
v_i	13	16	19	24	3	5
w_i	6	8	10	14	2	5
$\approx v_i/w_i$	2,2	2	1,9	1,7	1,5	1

Table 1 – Entrée du problème Knapsack

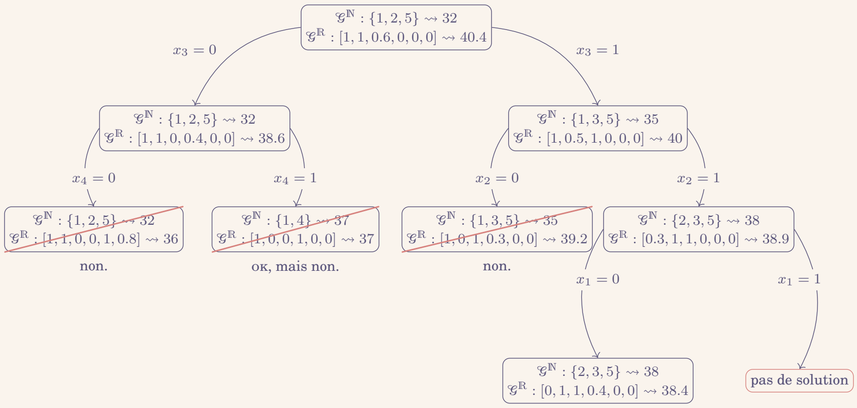

Figure 8 – Stratégie *branch and bound* appliquée au problème Knapsack

Annexe A. Programmation dynamique

On rappelle le problème Knapsack :

$$\begin{cases}\textbf{Entrée} & : n \in \mathbb{N},\ w \in (\mathbb{N}^\star)^n,\ v \in (\mathbb{N}^\star)^n,\ P \in \mathbb{N}\\ \textbf{Sortie} & : \max_{x\in\{0,1\}^n} \{\ \langle x, v\rangle \ \mid\ \langle x, w\rangle \leqslant P\}.\end{cases}$$

On pose

$$\textsc{sad}(n, w, v, P) = \max_{x\in\{0,1\}^n} \{\ \langle x, v\rangle \ \mid\ \langle x, w\rangle \leqslant P\},$$

et

$$\mathrm{sol}(n, w, v, P) = \{\langle x, v\rangle \mid \langle x, w\rangle \leqslant P,\ x \in \{0, 1\}^n\}.$$

Lorsque $y \in \mathbb{R}^n$, avec $y = (y_1, \dots, y_n)$, on note $\mathbb{R}^{n-1} \ni \tilde{y} = (y_2, y_3, \dots, 0)$. Ainsi, si $n > 0$,

$$\begin{aligned}\mathrm{sol}(n, w, v, P) &= \{\langle x, v\rangle \mid \langle x, w\rangle \leqslant P,\ x \in \{0, 1\}^n \text{ et } x_1 = 0\}\\ &\quad \cup \{\langle x, v\rangle \mid \langle x, w\rangle \leqslant P,\ x \in \{0, 1\}^n \text{ et } x_1 = 1\}\\ &= \{\langle \tilde{x}, \tilde{v}\rangle \mid \langle \tilde{x}, \tilde{w}\rangle \leqslant P,\ xu \in \{0, 1\}^n \text{ et } x_1 = 0\}\\ &\quad \cup \{v_1 + \langle \tilde{x}, \tilde{v}\rangle \mid \langle \tilde{x}, \tilde{w}\rangle \leqslant P - w_1,\ x \in \{0, 1\}^n \text{ et } x_1 = 1\}\\ &= \{\langle y, \tilde{v}\rangle \mid \langle y, \tilde{w}\rangle \leqslant P \text{ et } y \in \{0, 1\}^{n-1}\}\\ &\quad \cup \{v_1 + \langle y, \tilde{v}\rangle \mid \langle y, \tilde{w}\rangle \leqslant P - w_1 \text{ et } y \in \{0, 1\}^{n-1}\}\end{aligned}$$

D'où, par passage au $\max$, si $n > 0$,

$$\begin{aligned}\textsc{sad}(n, w, v, P) = \max(&\\ &\max\{\langle y \mid \tilde{v}\rangle \mid \langle y, \tilde{w}\rangle \leqslant P \text{ et } y \in \{0, 1\}^{n-1}\}\\ &v_1 + \max\{\langle y \mid \tilde{v}\rangle \mid \langle y, \tilde{w}\rangle \leqslant P - w_1 \text{ et } y \in \{0, 1\}^{n-1}\}\\ ) = \max\big(&\textsc{sad}(n-1, \tilde{w}, \tilde{v}, P), v_1 + \textsc{sad}(n-1, \tilde{w}, \tilde{v}, P - w_1)\big).\end{aligned}$$

Si $n = 0$, alors $\textsc{sad}(0, v, w, P) = 0$.

> **Remarque :**
> Si on le code *tel quel*, il y aura $\mathcal{O}(2^n)$ appels récursifs. Mais, on a $(n+1)(P+1)$ sous-problèmes.

Notons alors, pour n, v, w, P fixés, $(s_{i,j})_{\substack{i\in[\![1,n]\!]\\ j\in[\![0,P]\!]}}$ tel que

$$s_{i,j} = \textsc{sad}\Big(n - i, v_{|[\![i+1,n]\!]}, w_{|[\![i+1,n]\!]}, j\Big).$$

On a alors $\textsc{sad}(n, v, w, P) = s_{0,P}$. Ainsi, pour $j \in [\![0, P]\!]$, $s_{n,j} = 0$; pour $i \in [\![0, n]\!]$, $s_{i,0} = 0$;

$$s_{i,j} = \max(s_{i+1,j}, v_{i+1} + s_{i+1,j-w_{i+1}});$$

et, si $w_{i+1} > j$, alors $s_{i,j} = s_{i+1,j}$.

La complexité de remplissage de la matrice est en $\mathcal{O}(n\,P)$ en temps et en espace. On n'a pas prouvé $\mathbf{P} = \mathbf{NP}$, la taille de l'entrée est

— pour un entier n : $\log_2(n)$,
— pour un tableau de n entiers : $n \log_2(n)$,
— pour un tableau de n entiers : $n \log_2(n)$,
— pour un entier P : $\log_2(P)$.

Vis à vis de la taille de l'entrée, la complexité de remplissage est exponentielle.