

CHAPITRE 7

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

Hugo SALOU MPI*

Dernière mise à jour le 31 mars 2023

Table des matières

0	Motivation	2
1	Problèmes d'optimisation	2
2	Algorithmes d'approximations	3
3	<i>Branch and Bound</i> — Séparation et évaluation	6
Annexe A. Programmation dynamique		7

0 Motivation

On considère le problème **NP**-complet du *voyageur de commerce* : étant donné un graphe pondéré G quel est le tour de longueur¹ 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 polynomiale.

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 $\text{opt} \in \{\min, \max\}$. On dit que Q est un *problème d'optimisation* (*i.e.* problème de minimisation, maximisation), si pour toute entrée $e \in \mathcal{E}$, il existe

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

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

$$\forall s \in \text{sol}(e), \quad (e, s) \in Q \implies c_e(s) = c_e^*.$$

On nomme :

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

On appelle *solution optimale* une solution de valeur optimale.

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.

REMARQUE :

Soit Q_O un problème d'optimisation et Q le problème de décision associé. Étant donné

1. poids des arrêtes total

un algorithme \mathcal{A}_O pour Q_O , on fabrique l'algorithme \mathcal{A} suivant résolvant Q .

Algorithme 1 Solution à un problème de seuil

Entrée e une entrée de Q_O et K un seuil

1 : **retourner** $c_e(\mathcal{A}_O) \stackrel{?}{\bowtie} K$ ▷ où \bowtie est \geq pour si opt est max, et \leq 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 $\text{OPT}(e)$ la valeur optimale pour une entrée e .

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

$$\forall e \in \mathcal{E}_Q, \quad \mathcal{A}(e) \geq \rho \cdot \text{OPT}(e).$$

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

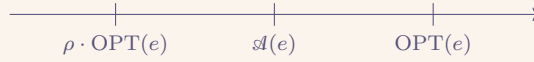


FIGURE 1 – Algorithme d'approximation pour un problème de maximisation

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

$$\forall e \in \mathcal{E}_Q, \quad \mathcal{A}(e) \leq \rho \cdot \text{OPT}(e).$$

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

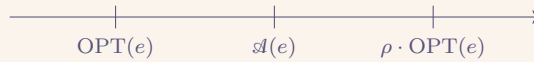


FIGURE 2 – Algorithme d'approximation pour un problème de minimisation

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

Définition : On dit qu'un algorithme $\mathcal{A} : \mathcal{E}_Q \rightarrow \mathbb{R}$ est une ρ -approximation différen-

tielle dès lors que

$$\frac{|A(e) - \text{Pire}(e)|}{|\text{Pire}(e) - \text{OPT}(e)|} \geq \rho.$$

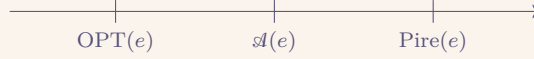


FIGURE 3 – ρ -approximation différentielle

REMARQUE :

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

$$\frac{A(e)}{\text{OPT}(e)} \leq \underbrace{\frac{A(e)}{B}}_{\rho}.$$

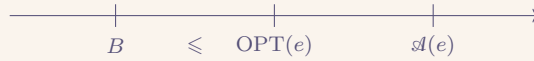


FIGURE 4 – Calcul de $\text{OPT}(e)$

Algorithme 2 Algorithme glouton de recherche de stables

Entrée $G = (S, A)$ un graphe

```

1:  $S' \leftarrow \emptyset$ 
2: tant que  $S \neq \emptyset$  faire
3:    $v^* = \arg \min_{v \in S} \deg_G(v)$   $\triangleright$  les degrés sont modifiés à chaque itération
4:    $S' \leftarrow S' \cup \{v^*\}$ 
5:    $S \leftarrow S \setminus (\{v^*\} \cup \text{voisins}(v^*))$ 
6:    $A \leftarrow$  restriction de  $A$  à  $S$ 
7: retourner  $S'$ 

```

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

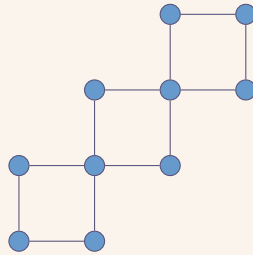


FIGURE 5 – Contre-exemple à l'algorithme 2

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

■

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

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

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*

1 : $C \leftarrow \emptyset$

2 : **tant que** $\exists \{u, v\} \in A$, u libre dans C et v libre dans C **faire**

3 : **Soit** $\{u, v\}$ une telle arête.

4 : $C \leftarrow C \cup \{\{u, v\}\}$

5 : **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

1 : $C \leftarrow \text{COUPLAGEMAXIMAL}(G)$

2 : **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 \mathcal{A} est une 2-approximation du problème « couverture par sommets. »

$$\forall G \in \mathcal{G}, \quad \mathcal{A}(G) \leq 2 \text{OPT}(G).$$

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

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

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 $|C| \leq |X^*|$. Or, $\mathcal{A}(X) = |X| = 2|C| \leq 2|X^*| \leq 2 \text{OPT}(X)$.

3 *Branch and Bound* — Séparation et évaluation

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

Annexe A. Programmation dynamique

On rappelle le problème KNAPSACK :

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

On pose

$$\text{SAD}(n, w, v, P) = \max_{x \in \{0,1\}^n} \{ \langle x, v \rangle \mid \langle x, w \rangle \leq P \},$$

et

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

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 $n > 0$,

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

D'où, par passage au max, si $n > 0$,

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

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

REMARQUE :

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

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

$$s_{i,j} = \text{SAD}(n-i, v|_{\llbracket i+1, n \rrbracket}, w|_{\llbracket i+1, n \rrbracket}, j).$$

On a alors $\text{SAD}(n, v, w, P) = s_{0,P}$. Ainsi, pour $j \in \llbracket 0, P \rrbracket$, $s_{n,j} = 0$; pour $i \in \llbracket 0, n \rrbracket$, $s_{i,0} = 0$;

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

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

La complexité de remplissage de la matrice est en $\mathcal{O}(nP)$ 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**.