

TD n°10

\mathcal{NP} -Complétude et Approximation

1 \mathcal{NP} -complétude de 2-Partition

On définit le problème de décision 2-Partition ainsi : soit $S = \{a_1, \dots, a_n\}$ des entiers, existe-t-il $I \subset S$ tel que $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Question 1.1 Montrer que 2-Partition est NP-complet.

Remarque : on pourra utiliser le problème NP-complet SUBSET-SUM vu dans le TD précédent : Instance : un ensemble fini S d'entiers positifs et un entier objectif t .

Question : existe-t-il un sous-ensemble $S' \subseteq S$ tel que $\sum_{x \in S'} x = t$?

Solution : 2-Partition est trivialement dans NP : on vérifie en temps linéaire qu'un certificat nous donne bien $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$.

On fait une réduction à partir de SUBSET-SUM. Soit une instance de SUBSET-SUM : on a n entiers $S = \{a_1, \dots, a_n\}$, et un objectif t . On construit une instance de 2-Partition de la façon suivante :

- si $t \leq \sum_{i=1}^n a_i < 2t$: on pose $a_{n+1} = 2t - \sum_{i=1}^n a_i$, et on construit un ensemble $S' = S \cup \{a_{n+1}\}$ pour 2-Partition. On a donc $\sum_{i=1}^{n+1} a_i = 2t$. Si on a une solution I_1 à SUBSET-SUM telle que $\sum_{i \in I_1} a_i = t$, alors on a une solution pour 2-Partition en prenant $\sum_{i \in I_1} a_i = \sum_{i \in S' \setminus I_1} a_i = t$. Inversement si 2-Partition a une solution, alors nécessairement $a_{n+1} \in I$ ou $a_{n+1} \in S' \setminus I$, et il suffit de prendre le sous ensemble de S' qui ne contient pas a_{n+1} .
- si $2t \leq \sum_{i=1}^n a_i$: on pose $a_{n+1} = \sum_{i=1}^n a_i - 2t$, et on construit l'ensemble S' comme précédemment. On a donc $\sum_{i=1}^{n+1} a_i = 2 \sum_{i=1}^n a_i - 2t$. Comme précédemment on a l'équivalence des solutions avec $\sum_{i \in I_1} a_i = \sum_{i \in S' \setminus I_1} a_i = \sum_{i=1}^n a_i - t$, et donc nécessairement on a soit dans I_1 soit dans $S' \setminus I_1$: $\sum_{i=1}^n a_i - 2t + \sum_{i \in J} a_i$, et donc l'ensemble J permet d'avoir $\sum_{i \in J} a_i = t$ la solution à SUBSET-SUM.

Donc 2-Partition est NP-complet. □

2 3-Partition à la 2-Partition

Étant donnés n entiers $S\{a_1, a_2, \dots, a_n\}$, peut-on trouver trois sous-ensembles I_1, I_2 et I_3 partitionnant $[1..n]$ et tels que $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i = \sum_{i \in I_3} a_i$?

Question 2.1 Montrer que 3-Partition à la 2-Partition est NP-complet.

Solution : 3-partition appartient à NP : étant donné un certificat on vérifie en temps linéaire que $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i = \sum_{i \in I_3} a_i$.

On va faire une réduction à partir de... 2-Partition (étonnant non ?). Pour construire une instance de 3-Partition on prend donc une instance de 2-Partition avec n entiers $S = \{a_1, \dots, a_n\}$, avec $P = \sum_{i=1}^n a_i$ et tels que $\forall i, a_i \leq P/2$ (sinon il n'y a pas de solution), et on

rajoute un nouvel entier $a_{n+1} = P/2$. Si 2-Partition a une solution I avec l'ensemble initial S , alors il suffit de prendre comme solution pour 3-Partition : $\sum_{i \in I} a_i = \sum_{i \notin I} a_i = a_{n+1}$, soit $I_1 = I$, $I_2 = S \setminus I$ et $I_3 = a_{n+1}$.

Réciproquement, si 3-Partition a une solution, alors nécessairement on a I_1 ou I_2 ou I_3 qui est égal à $I' \cup a_{n+1}$, avec $\sum_{i \in I'} a_i = 0$, et il suffit de prendre les sous-ensembles restant comme solution de 2-Partition.

Donc 3-Partition est NP-complet. □

Question 2.2 S'agit-il de NP-complétude au sens faible ou au sens fort ?

Solution : Au sens faible. Prendre l'algo du cours pour 2-Partition et l'adapter pour 3-Partition. □

3 Approximabilité de SUBSET-SUM

Dans TD le précédent, on s'est intéressé au problème de décision SUBSET-SUM consistant à savoir s'il existe un sous-ensemble d'entiers de S dont la somme vaut exactement t . Le problème d'optimisation qui lui est associé prend aussi en entrée un ensemble d'entiers strictement positifs S et un entier t , il consiste à trouver un sous-ensemble de S dont la somme est la plus grande possible sans dépasser t (cette somme qui doit donc approcher le plus possible t sera appelée *somme optimale*).

On suppose que $S = \{x_1, x_2, \dots, x_n\}$ et que les ensembles sont manipulés sous forme de listes triées par ordre croissant. Pour une liste d'entiers L et un entier x , on note $L+x$ la liste d'entiers obtenue en ajoutant x à chaque entier de L . Pour les listes L et L' , on note **Fusion**(L, L') la liste contenant l'union des éléments des deux listes.

Premier algorithme

début

$n \leftarrow |S|;$

$L_0 \leftarrow \{0\};$

pour i de 1 à n faire

$L_i \leftarrow \mathbf{Fusion}(L_{i-1}, L_{i-1} + x_i);$

Supprimer de L_i tout élément supérieur à t ;

fin

retourner le plus grand élément de L_n ;

fin

Algorithme 1 : Somme(S, t)

Question 3.1 Quelle est la distance entre la valeur retournée par cet algorithme et la somme optimale ?

Solution : On a un ensemble $S = \{x_1, \dots, x_n\}$ d'entiers strictement positifs ainsi qu'un entier t . On cherche une somme partielle d'éléments de S maximale et inférieure à t . On appellera *somme optimale* ce nombre.

Dans le premier algorithme, L_i représente l'ensemble des sommes partielles des éléments de $\{x_1, \dots, x_i\}$ inférieures à t . L_n représente donc l'ensemble des sommes partielles de S inférieures à t . Par conséquent, $\max(L_n)$ renvoie exactement la somme optimale de S . De plus, cet algorithme a testé toutes les sommes partielles inférieures à t .

La différence entre le résultat trouvé et la somme optimale est 0. □

Question 3.2 Quelle est la complexité de cet algorithme dans le cas général? Et si pour un entier $k \geq 1$ fixé, on ne considère que les entrées telles que $t = \mathcal{O}(|S|^k)$, quelle est la complexité de cet algorithme?

Solution :

- Dans le pire des cas, on teste toutes les sommes partielles de S soit une complexité en $O(2^n)$: il existe des entrées telles que $|L_i| = 2^i$, prendre par exemple $S = \{1, 2, 2^2, \dots, 2^i, \dots, 2^{n-1}\}$ et t grand ($\geq 2^n$). Or comme la fusion à chaque étape est en $O(|L_{i-1}|)$, la complexité totale est exponentielle en n .
- Si $t = O(|S|^k) = O(n^k)$ pour k un entier fixé, alors $|L_i| = O(|S|^k)$. Au pas i de la boucle, la fusion et la suppression se font en $O(|L_i|)$. Comme il y a n pas dans la boucle, la complexité totale est $O(|S|^{k+1})$. □

Deuxième algorithme Cet algorithme prend en entrée un paramètre ϵ en plus, où ϵ est un réel vérifiant $0 < \epsilon < 1$.

début

$n \leftarrow |S|;$

$L_0 \leftarrow \{0\};$

pour i **de** 1 **à** n **faire**

$L_i \leftarrow \text{Fusion}(L_{i-1}, L_{i-1} + x_i);$

$L_i \leftarrow \text{Seuiller}(L_i, \epsilon/2n);$

Supprimer de L_i tout élément supérieur à t ;

fin

retourner le plus grand élément de L_n ;

fin

Algorithme 2 : Somme-avec-seuillage(S, t, ϵ)

L'opération **Seuiller** décrite ci-dessous réduit une liste $L = \langle y_0, y_1, \dots, y_m \rangle$ (supposée triée par ordre croissant) avec le seuil δ :

```

début
   $m \leftarrow |L|;$ 
   $L' \leftarrow \langle y_0 \rangle;$ 
   $dernier \leftarrow y_0;$ 
  pour  $i$  de 1 à  $m$  faire
    si  $y_i > (1 + \delta)dernier$  alors
      Insérer  $y_i$  à la fin de  $L'$ ;
       $dernier \leftarrow y_i;$ 
    fin
  fin
  retourner  $L'$ ;
fin

```

Algorithme 3 : $Seuiller(L, \delta)$

Question 3.3 Évaluer le nombre d'éléments dans L_i à la fin de la boucle. En déduire la complexité totale de l'algorithme. Pour donner la qualité de l'approximation fournie par cet algorithme, borner le ratio valeur retournée/somme optimale.

Solution : *Considérons une liste $l = \{y_0, \dots, y_m\}$ d'entiers triés par ordre croissant et $\delta > 0$. L'opération de seuillage va consister à dire : soit y' le dernier élément gardé de l (on garde y_0 au départ) si $y_i > y' * (1 + \delta)$ alors on va garder y_i . En gros, on enlève les éléments qui sont trop proches d'autres éléments.*

L'algorithme proposé va fonctionner de la même façon que le précédent sauf qu'il va seuiller à chaque pas de la boucle L_i par rapport à $\epsilon/2n$. On a alors $L_n = \{y_0, \dots, y_m\}$. On va essayer de majorer m . On sait que :

$$t \geq y_m \geq (1 + \epsilon/2n)^{m-1} * y_0 \geq (1 + \epsilon/2n)^{m-1}$$

D'où : $m \leq \ln t / \ln(1 + \epsilon/2n) + 1 \leq \frac{2n \log(t)}{\epsilon \log(2)} + 1$, car $(m-1) \log(1 + \epsilon/2n) \leq \log(t)$ et donc $(m-1)\epsilon/2n \log(2) \leq \log(t)$ (car on a l'inégalité $x \log(2) \leq \log(1+x)$ quand $0 \leq x \leq 1$).

m est donc polynomial en $\ln t$, en $1/\epsilon$ et en n , donc polynomial en la taille des données. L'algorithme est en $O(mn)$, donc polynomial en la taille des données. Complexité totale : $O(n \times (\frac{2n \log(t)}{\epsilon \log(2)} + 1)) = O(\frac{n^2 \log(t)}{\epsilon})$.

*Il faut maintenant vérifier qu'on a bien trouvé une $(1 + \epsilon)$ Approximation en montrant que : $\max\{L_n\} \geq (1 + \epsilon) * Opt(\text{somme optimale})$*

On note P_i l'ensemble des sommes partielles de $\{x_1, \dots, x_i\}$ inférieures à t .

Invariant : $\forall x \in P_i, \exists y \in L_i, x/(1 + \delta)^i \leq y \leq x$

Par récurrence :

- $i = 0$: OK

- On suppose que c'est vrai pour $(i-1)$ et on le montre pour i : Soit $x \in P_i$

$$P_i = P_{i-1} \cup (P_{i-1} + x_i)$$

$x = x' + e$ avec $x' \in P_{i-1}$, $e = 0$ ou $e = x_i$

$x' \in P_{i-1}$ donc $\exists y', x'/(1 + \delta)^i \leq y' \leq x'$

Si $y' + e$ est conservé par le seuillage :

$$(x' + e)/(1 + \delta)^i \leq x'/(1 + \delta)^{i-1} + e \leq y' + e \leq x' + e = x$$

Si $y' + e$ n'est pas conservé par le seuillage :

$$\exists y'' \in L_i, y'' \leq y' + x_i \leq y'' * (1 + \delta)$$

$$(y' + e)/(1 + \delta) \leq y'' \leq y' + e \leq x' + e \leq x$$

$$(x'/(1 + \delta)^{i-1} + e)/(1 + \delta) \leq y''$$

$$(x' + e)/(1 + \delta)^i \leq y''$$

C'est l'encadrement recherché.

Grâce à l'invariant, on obtient : $Algo \geq Opt/(1 + \epsilon/2n)^n \geq Opt(1 + \epsilon)$

On a donc une $(1 + \epsilon)$ approximation de Subset-sum polynomiale en la taille des données et polynomiale en $1/\epsilon$.

□

4 Encore des problèmes \mathcal{NP} -complets

4.1 2-Partition-Approx

Question 4.1 Etant donnés n entiers a_1, a_2, \dots, a_n , peut-on trouver un sous-ensemble $I \subset [1..n]$ tel que $|\sum_{i \in I} a_i - \sum_{i \notin I} a_i| \leq 1$ **Solution** : $\in NP$: *trivial*.

Instance I1 de départ : a_1, \dots, a_n de 2-partition.

Instance I2 créée : $(1664a_1, \dots, 1664a_n)$.

Equivalence : S'il existe I avec $\sum_{i \in I} a_i = \sum_{i \in [1,n] \setminus I} a_i$, alors $|\sum_{i \in I} 1664a_i - \sum_{i \in [1,n] \setminus I} 1664a_i| = 0 \leq 1$

Réciproquement, s'il existe I avec $|\sum_{i \in I} 1664a_i - \sum_{i \in [1,n] \setminus I} 1664a_i| \leq 1$, alors on a $|\sum_{i \in I} a_i - \sum_{i \in [1,n] \setminus I} a_i| \leq 1/1664$. Or on travaille dans les entiers, ainsi on aboutit à $\sum_{i \in I} a_i = \sum_{i \in [1,n] \setminus I} a_i$.
□

4.2 2-Partition avec même cardinal

Question 4.2 Soient $n = 2p$ un entier pair, et n entiers strictement positifs a_1, a_2, \dots, a_n . Existe-t-il une partition de $\{1, 2, \dots, n\}$ en deux ensembles I et I' de même cardinal p et tels que $\sum_{i \in I} a_i = \sum_{i \in I'} a_i$? **Solution** : $\in NP$: *trivial*.

Instance I1 de départ : a_1, \dots, a_n de 2-partition.

Instance I2 créée : $(a_1 + 1, \dots, a_n + 1, 1, 1, \dots, 1)$ de cardinal $2n$.

Equivalence : Si $\sum_{i \in I} a_i = \sum_{i \in [1,n] \setminus I} a_i$ alors $\sum_{i \in I} (a_i + 1) + (n - |I|) = \sum_{i \in [1,n] \setminus I} (a_i + 1) + |I|$.

Réciproquement, une solution est de la forme $\sum_{i \in I} (a_i + 1) + (n - |I|) = \sum_{i \in [1,n] \setminus I} (a_i + 1) + |I|$. En effet il faut avoir n termes à droite et à gauche donc il faut rajouter un $(n - |I|)$ à gauche, et $|I| \leq n$ sinon au n'aurait jamais l'égalité, car tous les a_i sont positifs. Ainsi on a bien $\sum_{i \in I} a_i = \sum_{i \in [1,n] \setminus I} a_i$
□

4.3 Chevaliers de la table ronde

Question 4.3 Chevaliers de la table ronde

Étant donnés n chevaliers, et connaissant toutes les paires de féroces ennemis parmi eux, est-il possible de les placer autour d'une table circulaire de telle sorte qu'aucune paire de féroces ennemis ne soit côte à côte ?

Solution : $\in NP$: *trivial*.

Instance I1 de départ : (V,E) de *Circuit Hamiltonien*.

Instance I2 créée : On crée un chevalier pour chaque sommet de V . Deux chevaliers sont féroces ennemis ssi il n'y a pas d'arête de E entre les deux sommets de V desquels ils proviennent.

Équivalence : Si on a un Circuit réalisable dans I1, alors on place les chevaliers autour de la table de la façon suivante : un chevalier est voisin d'un autre ssi le sommet représentant du premier est relié au sommet représentant du second dans le circuit.

Réciproquement, si on peut réunir les chevaliers autour de la table, alors on peut faire un Circuit en prenant pour arêtes les arêtes entre les représentants des chevaliers de toutes les paires de chevaliers voisins. \square

4.4 Vertex Cover avec degré pair

Question 4.4 Soient $G = (V,E)$ un graphe dont tous les sommets sont de degré pair, et $k \geq 1$ un entier. Existe-t-il un ensemble de sommets de G couvrant toutes les arêtes et de taille inférieure ou égale à k ?

Solution : $\in NP$: *trivial*.

Instance I1 de départ : (G,k) de *Vertex Cover*.

Instance I2 créée : On peut déjà remarquer qu'il y a un nombre pair de sommets de degré impair dans G car $\sum deg(v) = 2|E|$. On construit une instance $(G2, k+2)$ en créant trois nouveaux sommets x, y et z . Ensuite on relie x à chaque sommet de G de degré impair, ainsi qu'à y et z , et on relie y et z (on a ainsi formé un triangle xyz)

Équivalence : S'il existe une couverture C dans V de cardinal plus petit que k pour G , alors il existe une couverture de cardinal plus petit que $k+2$ (par exemple $C \cup \{x,y\}$ pour $G2$).

Réciproquement, s'il existe une couverture $C2$ de cardinal plus petit que $k+2$ pour $G2$, alors $C2$ contient forcément au moins deux sommets dans $\{x,y,z\}$ car c'est un triangle qui n'est relié au reste du graphe que grâce à x . Donc $C = C2 \setminus \{x,y,z\}$ est une couverture de G de cardinal plus petit que k . \square

