# Complexité TD4 : NP et réductions

Université d'Aix-Marseille, M1 Informatique

Année universitaire 2020–2021

## 1 Transversal et Zone-Vide

Dans ce problème, nous revenons sur le problème de décision appelé TRANSVERSAL dont la définition est donnée ci-dessous :

> **Problème :** TRANSVERSAL
> **Données :** Un graphe non-orienté sans boucle $G = (S, A)$, et un entier positif $k \leq |S|$.
> **Question :** Existe-t-il un transversal de $G$ de taille $k$ ou moins ?

où un *transversal* de $G = (S, A)$ un graphe non-orienté, est un sous-ensemble $S'$ de $S$ tel que pour toute arête $\{x, y\} \in A$, alors $x \in S'$ ou $y \in S'$. En d'autres termes, un transversal est un sous-ensemble de sommets partageant au moins un sommet avec chaque arête du graphe.

**Question 1.1.** Démontrez que le problème TRANSVERSAL appartient à la classe **NP** en exhibant un algorithme pour modèle de calcul non-déterministe (pseudo-C non-déterministe par exemple) qui le résout et dont la complexité est polynomiale. Démontrez que le problème TRANSVERSAL appartient à la classe **NP** en exhibant un certificat polynomial. (Vous pouvez utiliser la représentation de graphes en matrice d'adjacence.)

**Solution 1.1.** Cet algorithme devine de forme non déterministe un sous-ensemble $X$ des sommets sous forme de tableau de booléens. À chaque itération il reste à choisir $c$ cases (initialement $c = k$) pour y mettre un 1. Si $c = 0$, cela signifie qu'on a déjà choisi $k$ éléments, donc le reste des cases contiendront 0. Si $0 < c < n - x$, on choisit de façon non déterministe de prendre ou ne pas prendre l'élément courant (en décrémentant $c$ si c'est le cas). Sinon, c'est-à-dire si $c = n - x$, cela signifie que pour choisir un ensemble de $k$ éléments il faudra choisir tous les éléments qui restent, en mettant des 1 dans la case correspondante. Finalement, on vérifie si l'ensemble $X$ deviné est tout à fait un transversal, comme on a vu dans la planche de TD2.

```
1     algorithme transversal(G, k):
2         X := tableau de booléens de longueur G.n
3         c := k
4         pour x := 1 à G.n faire
5             si c = 0 alors
6                 X[x] := 0
7             sinon si 0 < c < n-x alors
8                 X[x] := devine(0, 1)
9             sinon
10                X[x] := 1
11            if X[x] = 1 alors
12                c := c - 1
```

```
13        pour x := 1 à G.n faire
14            pour y := x + 1 à G.n faire
15                si G.A[x][y] et non X[x] et non X[y] alors
16                    retourner faux
17        retourner vrai
```

Deviner l'ensemble $X$ prends du temps $\Theta(n)$ (lignes 2–12), et vérifier s'il s'agit d'un transversal $\Theta(n^2)$ dans le pire des cas (lignes 13–17), pour un total de $\Theta(n^2)$, ce qui est du temps linéaire pour la représentation en matrices d'adjacence. Cela montre que le problème appartient à **NP**.

Un certificat pour ce problème est, tout simplement, l'ensemble des sommet d'un transversal de taille $k$ ou moins, qui a taille polynomial, par exemple $\Theta(n)$ si on le représente par un tableau de booléens. Voici un algorithme qui prend en entrée le graphe $G$, la taille du transversal $k$ et le certificat $X$ et vérifie en temps polynomial si $X$ est tout à fait un transversal de la taille correcte :

```
algorithme test-transversal(G, k, X):
    c := k
    pour x := 1 à G.n faire
        if X[x] = 1 alors
            c := c - 1
    si c != 0 alors
        retourner faux
    pour x := 1 à G.n faire
        pour y := x + 1 à G.n faire
            si G.A[x][y] et non X[x] et non X[y] alors
                retourner faux
    retourner vrai
```

Cet algorithme vérifie d'abord la taille de $X$ en temps $\Theta(n)$, puis s'il est un transversal, toujours selon le même principe et en temps $\Theta(n^2)$. Le temps total est, encore une fois, du temps linéaire $\Theta(n^2)$.

**Question 1.2.** En supposant que vous disposez d'un algorithme appelé Algo-Trans, de résolution pour ce problème, on cherche à exploiter cet algorithme pour résoudre un autre problème de décision, le problème Zone-Vide (une zone vide est un sous-graphe sans arête).

> **Problème :** Zone-Vide
> **Données :** Un graphe non-orienté sans boucle $G = (S, A)$, et un entier positif $k \leq |S|$.
> **Question :** Existe-t-il dans $G$ une zone vide de taille $k$ ou plus ?

Notons tout d'abord qu'en théorie des graphes, on emploie le terme *stable* plutôt que "zone vide". Montrez comment on peut résoudre le problème Zone-Vide en exploitant l'algorithme Algo-Trans. Pour cela, vous essayerez de transformer les données de Zone-Vide en données de Transversal. Evaluez la complexité de résolution de Zone-Vide par cette approche, en supposant que la complexité de Algo-Trans est $\Theta(f(n))$.

**Solution 1.2.** On commence par démontrer que

> un ensemble $X \subseteq S$ est une zone vide du graphe $G = (S, A)$ si et seulement si $S \setminus X$ est un transversal.

Si $X$ est une zone vide, alors pour toute arête $\{x, y\} \in A$, soit $x \notin X$, soit $y \notin A$ (soit les deux) ou, autrement dit, soit $x \in S \setminus X$, soit $y \in S \setminus X$ (soit les deux). Cela implique que $S \setminus X$ est un transversal.

Vice-versa, supposons que $S \setminus X$ soit un transversal. Alors, pour tout $x, y \in X$, ce n'est jamais le cas que $\{x, y\} \in A$, puisque $S \setminus X$ contient une extrémité de chaque arête. Donc $X$ est une zone vide.

Par conséquent :

> le graphe $G = (S, A)$ a une zone vide ($X$) de taille $k$ si et seulement il a un transversal ($S \setminus X$) de taille $n - k$.

On peu donc résoudre le problème Zone-Vide de la manière suivante : d'abord on calcule la réduction suivante de Zone-Vide à Transversal :

```
algorithme réduction-zv-à-t(G, k):
    retourner (G, G.n - k)
```

puis à appliquer l'algorithme Algo-Trans au résultat :

```
algorithme algo-zone-vide'(G, k):
    retourner algo-trans(réduction-zv-à-t(G, k))
```

ce qui prend le même temps de calcul d'Algo-Trans, c'est à dire $\Theta(f(n))$, plus un temps constant pour la réduction.

**Question 1.3.** En supposant que vous disposez d'un algorithme de résolution de Zone-Vide appelé Algo-Zone-Vide dont la complexité serait $\Theta(g(n))$, montrez comment on peut résoudre le problème Transversal en l'exploitant. Evaluez la complexité de résolution de Transversal par cette approche.

**Solution 1.3.** Le raisonnement de la question précédente est complètement symétrique ; voici donc une réduction de Trasnversal à Zone-Vide :

```
algorithme réduction-t-à-zv(G, k):
    retourner (G, G.n - k)
```

qui nous permet de résoudre le problème Transversal comme suit :

```
algorithme algo-trans'(G, k):
    retourner algo-zone-vide(réduction-t-à-zv(G, k))
```

en temps de calcul $\Theta(g(n))$ plus une constante.

**Question 1.4.** Quelle serait la complexité de résolution de Zone-Vide en utilisant le procédé développé dans la question 1.2 en supposant que la complexité de l'algorithme Algo-Trans soit maintenant polynomiale ?

**Solution 1.4.** La complexité serait également polynomiale, avec le même polynôme plus une constante. En effet, le raisonnement étant symétrique (question 1.3), on peut en déduire que soit les problèmes possèdent tous les deux des algorithmes polynomiaux, soit aucun des deux n'en possède (mais on n'est pas encore en mesure de l'établir...).

## 2 Partition et Somme

**Question 2.1.** On considère le problème suivant :

> **Nom :** Partition
> **Donnée :** Un ensemble fini $A$, une fonction de taille $t \colon A \to \mathbb{N}$
> **Question :** $A$ possède-t-il un sous-ensemble $B$ dont la somme de la taille de ses éléments est exactement égale à la somme de la taille des éléments de $A$ qui ne sont pas dans $B$ ?

Démontrez que le problème PARTITION appartient à la classe **NP** en exhibant un algorithme pour modèle de calcul non-déterministe (pseudo-C non-déterministe par exemple) qui le résout et dont la complexité est polynomiale. Démontrez que le problème PARTITION appartient à la classe **NP** en exhibant un certificat polynomial.

**Solution 2.1.** On peut représenter la fonction $t$ par un tableau d'entiers naturels `t` de longueur $n = |A|$, où `t[i]` $= t(a_i)$ et $A = \{n_1, \ldots, a_n\}$. Dans ce cas, on n'a pas besoin de représenter explicitement l'ensemble $A$.

L'algorithme non déterministe construit l'ensemble $B$ (ici représenté, comme d'habitude, par un tableau de booléens) en devinant pour chaque élément de $A$ s'il fait partie de $B$ ou pas. Enfin, on calcule la somme des éléments de $B$ (`somme1`) et la somme des éléments de $B \setminus A$ (`somme2`) et on les compare.

```
algorithme partition(t):
    n := longueur(t)
    B := tableau de booléens de longueur n
    pour i := 1 à n faire
        B[i] := devine(0, 1)
    somme1 := 0
    somme2 := 0
    pour i := 1 à n faire
        si B[i] alors
            somme1 := somme1 + t[i]
        sinon
            somme2 := somme2 + t[i]
    retourner somme1 = somme2
```

Le temps de calcul de cet algorithme est dominé par les deux parcours du tableau `B`, ce qui donne un temps de calcul polynomial $\Theta(n)$ par rapport à la taille de $A$, ce qui montre que le problème appartient à **NP**.

Un certificat qui montre que une certaine entrée est une instance positive du problème est un tableau de booléens qui représente l'ensemble $B$. Ce certificat è de taille polynomial, $\Theta(n)$ par rapport à la taille de $A$, et on peut le vérifier en temps polynomial déterministe (toujours $\Theta(n)$) avec l'algorithme suivant :

```
algorithme test-partition(t, B):
    n := longueur(t)
    somme1 := 0
    somme2 := 0
    pour i := 1 à n faire
        si B[i] alors
            somme1 := somme1 + t[i]
        sinon
            somme2 := somme2 + t[i]
    retourner somme1 = somme2
```

Cet algorithme correspond tout à fait à l'algorithme non déterministe précédent, sauf qu'il n'est pas nécessaire de deviner `B`, vu que celui-ci est fourni en entrée.

**Question 2.2.** On considère le problème suivant :

**Nom :** Somme
**Donnée :** Un ensemble fini $A$, une fonction de taille $t\colon A \to \mathbb{N}$ et un entier $k$
**Question :** A possède-t-il un sous-ensemble $B$ dont la somme de la taille de ses éléments vaut $k$ ?

Mêmes questions que dans le problème 1 avec les problèmes Partition et Somme et des algorithmes respectivement appelés Algo-Partition et Algo-Somme

**Solution 2.2.** Il est possible d'utiliser un algorithme `algo-somme(t, k)`, qui vérifie s'il existe un sous-ensemble $B$ ayant somme $\sum_{b_i \in B} \texttt{t[i]} = k$, pour résoudre le problème Partition en gardant la même fonction $t$ et en choisissant comme valeur de $k$ la démi-somme des tailles de tous les éléments $\sum_{a_i \in A} \texttt{t[i]}/2$. Cela correspond à calculer la réduction suivante :

```
algorithme réduction-p-à-s(t):
    n := longueur(t)
    s := 0
    pour i := 1 à n faire
        s := s + t[i]
    k := s / 2
    retourner (t, k)
```

ce qui prend tu temps linéaire $\Theta(n)$, et l'exploiter pour résoudre le problème Partition en utilisant `algo-somme` :

```
algorithme algo-partition'(t):
    retourner algo-somme(réduction-p-à-s(t))
```

Si le temps de calcul de `algo-somme` est $\Theta(f(n))$, alors `algo-partition'` a un temps de calcul $\Theta(f(n)+n)$, qui est polynomial si et seulement si $\Theta(f(n))$ l'est.

Vice-versa, si on possède un algorithme `algo-partition(t)` on peut l'exploiter pour résoudre Somme par réduction.

Soit $(A, t, k)$ l'instance de Somme, et soient $A' = A \cup \{b_1, b_2\}$ l'ensemble obtenu en ajoutant deux nouveaux éléments à $A$, soit $s = \sum_{a \in A} t(a)$ et $t'\colon A' \to \mathbb{N}$ la fonction définie par

$$t'(x) = \begin{cases} t(x) & \text{si } x \in A \\ s + k & \text{si } x = b_1 \\ 2s - k & \text{si } x = b_2 \end{cases}$$

On veut prouver qu'il existe un sous-ensemble $B \subseteq A$ tel que $\sum_{a \in B} t(a) = k$ si et seulement s'il existe un sous-ensemble $B' \subseteq A'$ tel que $\sum_{a \in B'} t'(a) = \sum_{a \in A' \setminus B'} t'(a)$.

S'il existe un sous-ensemble $B \subseteq A$ tel que $\sum_{a \in B} t(a) = k$, soit $B' = B \cup \{b_2\}$. Alors la somme des tailles des éléments de $B'$ est

$$\sum_{a \in B'} t'(a) = \sum_{a \in B} t(a) + t'(b_2) = k + 2s - k = 2s$$

et la somme des éléments du complémentaire $A' \setminus B'$ est égalément

$$\begin{aligned} \sum_{a \in A' \setminus B'} t'(a) &= \sum_{a \in A'} t'(a) - \sum_{a \in B'} t'(a) = \sum_{a \in A} t(a) + t'(b_1) + t'(b_2) - 2s \\ &= s + (s + k) + (2s - k) - 2s = 2s. \end{aligned}$$

Vice-versa, supposons qu'il existe $B' \subseteq A'$ tel que $\sum_{a\in B'} t'(a) = \sum_{a\in A'\setminus B'} t'(a)$. La valeur de ces sommes est la moitié de la somme de toutes les valeurs :

$$\frac{1}{2}\sum_{a\in A'} t'(a) = \frac{1}{2}\big(s + (s+k) + (2s-k)\big) = 2s.$$

Puisque $t'(b_1) + t'(b_2) = (s+k) + (2s-k) = 3s \geq 2s$, ce n'est pas le cas que les deux éléments $b_1, b_2$ appartiennent tous les deux à $B'$, ni à son complémentaire. Donc soit $b_1 \in B'$ et $b_2 \notin B'$, soit $b_2 \in B'$ et $b_1 \notin B'$

Si $b_2 \in B'$, alors l'ensemble $B' \setminus \{b_2\} \subseteq A$ a somme $k$ :

$$\sum_{a\in B'\setminus\{b_2\}} t'(a) = \sum_{a\in B'} t'(a) - t'(b_2) = 2s - (2s-k) = k$$

Si, au contraire, $b_2 \in A' \setminus B'$, alors c'est $(A' \setminus B') \setminus \{b_2\} \subseteq A$ à avoir somme $k$ :

$$\begin{aligned}\sum_{a\in(A'\setminus B')\setminus\{b_2\}} t'(a) &= \sum_{a\in A'} t'(a) - \sum_{a\in B'} t'(a) - t'(b_2) = \\ &= \Big(\sum_{a\in A} t(a) + t'(b_1) + t'(b_2)\Big) - 2s - (2s-k) \\ &= \big(s + (s+k) + (2s-k)\big) - 2s - (2s-k) = k.\end{aligned}$$

Dans les deux cas, on obtient le résultat attendu.

La réduction de Somme à Partition consiste donc en ajouter à l'ensemble les deux éléments avec leurs valeurs :

```
algorithme reduction-s-à-p(t, k):
    n := longueur(t)
    t' := tableau de longueur n + 2
    s := 0
    for i := 1 à n faire
        t'[i] := t[i]
        s := s + t[i]
    t'[n+1] := s + k
    t'[n+2] := 2s - k
    retourner t
```

ce qui prend du temps $\Theta(n)$ et nous permet de résoudre le problème Somme avec l'algorithme pour Partition :

```
algorithme algo-somme'(t, k):
    retourner algo-partition(reduction-s-à-p(t, k))
```

Si le temps de calcul de `algo-partition` est $\Theta(g(n))$, alors le temps de calcul de `algo-somme'` est $\Theta(g(n)+n)$, qui est polynomial si et seulement si $\Theta(g(n))$ est polynomial.

Les deux problèmes Somme et Partition sont donc équivalents, dans le sens que il existe un algorithme polynomial pour l'un d'entre eux si et seulement si il existe pour l'autre (mais, encore une fois, on ne sait pas encore si c'est le cas.).