# Rendu TP n°03

## Qu'est-ce qu'une propriété NP ?

### Question 01 : la propriété est NP

**Certificat pour le problème :**

`c`, une permutation des `n` villes où `c[i]` est un numéro de ville choisi pour `0 <= i <= n`.

**|u|, taille de la donnée :**

En tenant compte de l'encodage des entiers, `|u| = n*logn + n*n*logn + l*logl` où : - `n*logn` est la taille occupée par `n`, le nombre de villes. - `n*n*logn` est la taille occupée par `D`, la matrice n*n - `l*logl` est la taille occupée par `l`, la longueur maximale autorisée.

**|c|, taille du certificat**

En tenant compte de l'encodage des entiers, `|c| = n*logn`. **De ce fait, la condition `|c| <= Q(|u|)` est vérifiée.**

**Algorithme de vérification**

L'algorithme de vérification du certificat consisterait à parcourir chaque ville du certificat (la permutation des n villes) pour vérifier que la condition `D(tour[n - 1]; tour[0]) + (SOMME POUR I DE 0 à n-2 D(tour[i], tour[i + 1])) <= l` soit correcte.

L'algorithme est donc bien polynomial.

**Conclusion**

**La propriété TSP est NP**

---

### Question 02 : NP = Non-déterministe polynomial

#### 2.1. Génération aléatoire d'un certificat

Voir le code.

#### 2.2. Algorithme non-déterministe

Le schéma d'un algorithme non-déterministe polynomial pour le problème serait :

```
GÉNERER un certificat aléatoire
VÉRIFIER QU'IL EST VALIDE (appliquer la vérification du certificat)
```

De ce fait, puisqu'un algorithme non-déterministe est correcte à partir du moment où une seule de ses exécutions renvoie vrai et que les méthodes de génération d'un certificat aléatoire et de vérification du certificat sont polynomiales, nous avons un algorithme non-déterministe polynomial.

### Question 03 : NP C ExpTime

#### 3.1. Ordre de grandeur du nombre de certificat

L'ordre de grandeur du nombre de certificat est **factoriel**.

#### 3.2. Un ordre d'énumération

L'ordre choisi est de classer toutes les possibilités possibles suivant l'ordre des numéros de ville dans le certificat.

Un certificat est considéré "plus petit" qu'un autre ssi : - Les numéros de ville du certificat sont plus petits que l'autre où le numéro de ville le plus à gauche l'emporte.

Par exemple, - `[0,1,2,3,4]` < `[1,0,3,2,4]` car 0 < 1 - `[0,1,2,3,4]` < `[0,1,3,2,4]` car 2 < 3

#### 3.3. Algorithme du British Museum

Pseudo-code :

```
certificat = PREMIER CERTIFICAT DE L'ORDRE
TANT QUE certificat EXISTE
    VERIFIER certificat
    SI certificat EST OK
        RETOURNER vrai
    SINON
        certificat = certificat suivant
RETOURNER faux
```

En bref, l'algorithme consiste à tester toutes les combinaisons de certificats possibles à partir du premier élément de l'ordre choisi et en passant au suivant à chaque itération. Sa complexité est donc **factorielle**.

---

## Réduction polynomiale

### Question 01 :

#### 1.1 : HamiltonCycle se réduit polynomialement dans TSP

Il faut donc montrer que nous pouvons transformer un problème HamiltonCycle en un problème TSP de manière polynomiale.

#### La transformation polynomiale

Nous pouvons transformer la donnée comme suit :

| Donnée HamiltonCycle | Donnée TSP |
|---|---|
| `n`, le nombre de sommets | `n`, le nombre de ville |
| `D`, la matrice de booléens | `D` (voir la définition en-dessous) |
| | `l = 0` |

La matrice `D` des distances de TSP est définie par : - `D(i,j) = 0` ssi il y a un arc entre les sommets `i` et `j` - `D(i,j) = 1` sinon - pour tout `i` et `j`, `0 <= i <= n-1` et `0 <= j <= n-1`

#### La transformation est polynomiale

- La valeur du `n` ne bouge pas d'un problème à l'autre
- La valeur de `l` est fixe (0)
- Pour construire `D`, il suffit de parcourir la matrice du problème originel HamiltonCycle et donner un `D(i,j) = 0` quand la matrice HamiltonCycle donne `true` sur `i,j` et `D(i,j) = 0` quand on obtient `false`

La transformation est donc bien calculable par un algorithme polynomial.

#### La transformation est correcte

- Si une instance du problème de HamiltonCycle a une solution, on obtiendra, après transformation, une instance de TSP qui a une solution.
    - Un graphe qui possède un cycle hamiltonien aura, après transformation, une tournée de ville dont la longueur sera de 0.
- Si une instance du problème de HamiltonCycle n'a pas de solution, on obtiendra, après transformation, une instance de TSP qui n'a pas de solution.
    - Un graphe qui ne possède pas de cycle hamiltonien ne peut pas posséder de tournée de ville dont la longueur est <= 0 car s'il n'y a pas d'arcs entre deux sommets dans le problème de HamiltonCycle, on obtient une distance de 1 d'un sommet vers l'autre et donc on obtiendrait une tournée de ville où l > 0

### 1.2. Implémentation

Voir le code.

### 1.3. Qu'en déduire de TSP ?

Comme HamiltonCycle est NP-dur, on en déduit que TSP est également NP-dur. De ce fait, comme TSP est NP et NP-dur, on peut déduire que TSP est **NP-complet**.

Le fait que TSP soit NP-dur montre que trouver un algorithme polynomial pour résoudre TSP impliquerait P = NP, qui est peu probable. Cela justifie le fait qu'on n'ait pas trouvé un algorithme polynomial pour décider la propriété.

### 1.4. Pensez-vous que TSP se réduise polynomialement dans HamiltonCycle ?

- TSP est NP
- HamiltonCycle est NP-dur
- On sait qu'une propriété NP-dur contient toute la complexité de NP
- **TSP se réduit bel et bien polynomialement dans HamiltonCycle**

---

## Question 02

### 2.1. HamiltonPath se réduit polynomialement dans HamiltonCycle

#### La transformation polynomiale

La transformation vers un problème de HamiltonCycle consiste à : - Conserver le même graphe que HamiltonPath - Ajouter un sommet spécial (que l'on va nommer "**FIN**") qui possédera un arc vers tous les sommets du graphe et dont tous les sommets du graphe posséderont un arc vers lui.

Nous pouvons ainsi transformer la donnée comme suit :

| **Donnée HamiltonPath** | **Donnée HamiltonCycle** |
|---|---|
| `n`, le nombre de sommets | `n + 1`, le nombre de sommets + le sommet **FIN** |
| `D`, la matrice de booléens | `D` où on rajoute les données du sommet FIN tel que `D(FIN,i) = true` et `D(i,FIN) = true`, pour `0 <= i <= n` |

#### La transformation est polynomiale

- La valeur du `n` est incrémentée pour accepter le sommet FIN
- Pour construire `D`,
    - Il suffit de conserver la matrice de HamiltonPath
    - Pour chaque sommet du graphe, rajouter `D(FIN,i) = true` et `D(i,FIN) = true`

La transformation est donc bien calculable par un algorithme polynomial.

#### La transformation est correcte

- Si une instance du problème de HamiltonPath a une solution, on obtiendra, après transformation, une instance de HamiltonCycle qui a une solution.
    - Un graphe qui possède un chemin hamiltonien aura, après transformation, conserver son chemin hamiltonien
    - De plus, comme le sommet FIN est pointé par tout le monde et pointe tout le monde, avoir un chemin hamiltonien impliquerait directement qu'on ait également un cycle puisque le sommet FIN fait le lien entre le début et la fin du chemin.
- Si une instance du problème de HamiltonPath n'a pas de solution, on obtiendra, après transformation, une instance de HamiltonCycle qui n'a pas de solution.

- Un graphe qui, de base, ne possède pas de chemin hamiltonien ne pourra pas devenir un cycle hamiltonien même après le rajout du sommet FIN

**2.2. Implémenter une réduction polynomiale HamiltonPath dans HamiltonCycle.**

Voir le code.

---

### Question 03 : montrer que HamiltonPath se réduit polynomialement dans TSP

La relation <=p (réduction polynomiale) est transitive : comme `HamiltonPath <=p HamiltonCycle` et `HamiltonCycle <=p TSP`, on déduit que `HamiltonPath <=p TSP`

### Question 04 : complexité des propriétés Hamilton dans un graphe non-dirigé

Dans un graphe non-dirigé, le nombre d'arcs est deux fois moins important que dans sa version dirigée. De ce fait, le nombre de cycles/chemins possibles, qui est de l'ordre de (n-1)! dans un graphe dirigé sera de (n-1)!/2 dans un graphe non-dirigé.

Par conséquent, même si le nombre de possibilités est réduit, nous restons dans un problème **NP-dur** (voir NP-complet).

---

## Propriétés versus Problèmes d'optimisation

### Question 05 :

S'il existe un algorithme polynomiale pour trouver la longueur minimale d'une tournée possible (`TSPOpt1`), alors il existe un algorithme polynomiale pour vérifier qu'il existe une tournée de longueur <= l (`TSP`). Il suffit d'appliquer l'algorithme TSPOpt1 et de vérifier que la longueur minimale obtenue est <= l.

S'il existe un algorithme polynomiale pour trouver la tournée possible de longueur minimale (`TSPOpt2`), alors il existe un algorithme polynomiale pour vérifier qu'il existe une tournée de longueur <= l (`TSP`). Il suffit d'appliquer l'algorithme `TSPOpt2` et de vérifier que la longueur de la tournée minimale obtenue est <= l.

On en déduit donc que si `TSPOpt1`/`TSPOpt2` était P, cela montrerait qu'il existe un algorithme polynomial pour un problème NP-dure et donc on aurait P = NP car la propriété NP-dure `TSP` contient toute la complexité de NP.

---

### Question 06 : si TSP était P, TSPOpt1 le serait aussi

Si on peut tester en temps polynomial qu'il existe une tournée de villes pour une longueur maximale

(TSP), alors nous avons un algorithme polynomial pour trouver la longueur minimale d'une tournée de ville (```TSPOpt1``) en utilisant un algorithme **naïf** de cette forme :

```
somme = SOMME DE TOUTES LES DISTANCES // obtenu en parcourant la matrice D
POUR i DE 1 à somme
    SI [il existe une tournée de villes pour l=i]
        RETOURNER i
RETOURNER somme
```

Ainsi, la complexité de `TSPOpt1` est au plus `(somme des distances) * complexité de TSP`.

*Note : Il est possible d'utiliser la dichotomie pour améliorer la complexité de l'algorithme mais le but ici est seulement de démontrer que le problème d'optimisation est polynomial si la propriété est polynomiale.*

---

## Question 07 : Si TSP était P, TSPOpt2 le serait aussi

Si le problème de décision TSP était P, nous aurions le problème d'optimisation TSPOpt2 P également. En effet, nous pourrions trouver TSPOpt2.

Nous devrions d'abord trouver la longueur minimale possible (avec TSPOpt1 qui serait P, comme prouvé dans la question précédente). Ensuite, tant qu'on n'a pas trouvé la tournée complète, on choisit un des sommets et on teste s'il existe une tournée de longueur minimale contenant le sommet choisi (`TSP`). De ce fait, on construit la tournée optimale en temps polynomial.