AAC: Séance 2

François Lepan
Benjamin Van Ryseghem

29 novembre 2012

1 Question 1

NP étend **ExpTime** car tous les problèmes NP-complet possèdent un algorithme de résolution de complexité *exponentielle*.

2 Question 2

2.1 Définition du certificat

Un certificat pour ce problème peut être une liste ordonnée des sommets tous distincts à parcourir afin de couvrir entièrement le graphe.

Les données en entrée sont sous la forme d'une matrice de taille (n,n), où n est le nombre de sommets du graphes.

On vérifie bien alors que :

$$\begin{array}{rcl} \text{taille du certificat} & \leqslant & \text{taille des données} \\ n & \leqslant & n^2 \end{array}$$

Vérifié le certificat revient à vérifier sa taille.

2.2 Vérification du certificat

Un algorithme de vérification du certificat est linéaire par rapport au nombre de sommets, puisqu'il s'agit principalement de vérifier sa taille.

```
boolean A(matrice_de_données_de_taille_n_n, G){
        Si taille de G différent de n retourner Faux;
        retourner Vrai;
}
```

2.3 Preuve que *TSP* est un problème *NP*

Nous venons de voir que le certificat est borné polynomialement par la taille des données d'entrée. De plus l'algorithme de vérification du certificat à une complexité polynomiale.

Ces deux propriétés prouvent que le problème *Travelling Salesman Problem* est **NP**.

2.4 Valeurs possible du certificat

Étant donné qu'un certificat est une permutation des n sommets du graphe, alors il existe $n!$ permutations possibles.

3 HamiltonCycle $\leqslant_p$ TSP

3.1 Transformation

Afin de montrer que HamiltonCycle $\leqslant_p$ TSP, il faut et il suffit de trouver une application *red* telle que :

$$u \in \text{HamiltonCycle} \Leftrightarrow red(u) \in \text{TSP}$$

La principale différence entre une instance de HamiltonCycle et une instance de TSp est qu'HamiltonCycle nécessite une matrice(n,n) de booléens là où TSP nécessite une matrice(n,n) d'entiers.

red est alors une application transformant une matrice(n,n) de booléens en une matrice(n,n) d'entiers :

$$\begin{array}{rcl} M_{n,n}(\mathbb{B}) & \rightarrow & M_{n,n}(\mathbb{N}) \\ u & \mapsto & red(u) \end{array}$$

où $\mathbb{B} = \{\text{vrai, faux}\}$.

Pour cela définissons l'application δ telle que :

$$\begin{array}{rcl} \mathbb{B} & \rightarrow & \{0,1\} \\ u & \mapsto & 1, \text{ si u est } \textit{vrai} \\ & & \infty, \text{ si u est } \textit{faux} \end{array}$$

Nous allons alors définir *red* en fonction de δ :

$$\begin{array}{ccc} & M_{n,n}(\mathbb{B}) \rightarrow M_{n,n}(\mathbb{N}) & \\ \begin{bmatrix} b_{1,1} & \dots & b_{1,n} \\ b_{2,1} & \dots & b_{2,n} \\ & \vdots & \\ b_{n,1} & \dots & b_{n,n} \end{bmatrix} & \mapsto & \begin{bmatrix} \delta(b_{1,1}) & \dots & \delta(b_{1,n}) \\ \delta(b_{2,1}) & \dots & \delta(b_{2,n}) \\ & \vdots & \\ \delta(b_{n,1}) & \dots & \delta(b_{n,n}) \end{bmatrix} \end{array}$$

L'application *red* est évidement polynomiale en n, puisqu'il suffit d'itérer sur les n^2 termes de la matrice.

Il reste à fournir à TSP une longueur maximale. En choisissant n comme longueur maximale, on a alors qu'il existe un chemin pour TSP seulement s'il existait un cycle Hamiltonien. On alors :

$$u \in \text{HamiltonCycle} \Rightarrow red(u) \in \text{TSP}$$

Afin de montrer l'implication inverse, prenons une telle instance de TSP. Si cette instance vérifie TSP, cela implique qu'il existe un cycle dont la longueur totale est inférieur à l.

Donc il existe un cycle.

Ceci implique donc :

$$u \in \text{HamiltonCycle} \Leftarrow red(u) \in \text{TSP}$$

3.2 Conclusion

De ce fait, nous pouvons conclure que :

$$u \in \text{HamiltonCycle} \Leftrightarrow red(u) \in \text{TSP}$$

On sait de surcroit qu'HamiltonCycle est NP-complet, donc NP-dur. Et comme HamiltonCycle $\leqslant_p$ TSP, on en déduit que TSP est *NP-dur*.

De plus, nous avons prouvé à la section 2.3 que TSP est NP. Nous en déduisons donc que TSP est **NP-complet**.

TSP se réduit polynomialement en HamiltonCycle puisque que TSP est NP et HamiltonCycle NP-dur.

4 HamiltonPath $\leqslant_p$ HamiltonCycle

HamiltonPath est NP-complet donc NP.
HamiltonCycle est NP-complet donc NP-dur.
De ce fait, HamiltonPath $\leqslant_p$ HamiltonCycle.

5 HamiltonPath $\leqslant_p$ TSP

De plus TSP est NP-dur également, donc HamiltonPath $\leqslant_p$ TSP.

6 TSPOpt1/TSPOpt2

6.1 TSPOpt1

Si TSPOpt1 était P, alors il existerait un algorithme déterministe polynomiale de résolution de TSPOpt1, `resTSPOpt1()`.

Or s'il existe un n et une matrice D tel que l = TSPOpt1(n, D) alors TSP(n,D,length) est *vrai* si et seulement si $l \leqslant$ length.

Ceci implique donc l'existence d'un algorithme polynomiale permettant de déterminer si TSP est vrai ou non.

```
resTSP(n, D, length){
        return resTSPOpt1(n,D) <= length;
}
```

De ce fait TSP serait P aussi.

6.2 TSPOpt2

De même si TSPOpt2 était P, alors il existerait un algorithme déterministe polynomiale de résolution de TSPOpt2, `resTSPOpt2()`.

On aurait alors un algorithme de résolution de TSP comme suit

```
resTSP(n, D, length){
        cert = resTSPOpt2(n,D);
        retourn cert.length() <= length;
}
```

Or cet algorithme est polynomiale car resTSPOpt2 l'est. De ce fait, TSP serait P.

6.3 TSP - TSPOpt1

Supposons TSP P, et resTSP, un algorithme de résolution de TSP déterministe polynomiale.

```
resTSPOPT1(n, D){
        int length = n*max(D); //où max(D) retourne la plus grande valeur de D
        while(TSP(n,D,length){
                length--;
        }

        return length+1;
}
```

Cet algorithme retourne en un temps polynomiale de n un résultat pour TSPOpt1.

6.4 TSP - TSPOpt2

```
resTSPOPT2(n, D){
        int length = resTSPOpt1(n,D);
        int infinite = Integer.MAX_VAL;

        for (int i = 0; i< n; i++) {
                for (int j = 0; j< n; j++) {
                        if (D[i][j] != infinite) {
                                int old = D[i][j];
                                D[i][j] = infinite;
                                if (!TSP(n,D,l)){
                                        D[i][j] = old;
                                }
                        }
                }
        }

        int currentLine = 0;
        for (int i = 0; i< n; i++) {
                result.add(currentLine);
                for(int j = 0 ; j < n ; j++){
                        if (D[currentLine][j] != infinite) currentLine = j;
                        break;
                }
        }


        return result;
}
```

Si TSP est P alors TSPOpt1 l'est aussi ce qui implique que `resTSPOpt1` est déterministe polynomiale en n. Et si `resTSPOpt1` est polynomiale en n alors `resTSPOpt2` l'est aussi.

De ce fait TSPOpt2 est P.

François Lepan
Benjamin Van Ryseghem