Université de Lille

FST- Département Informatique

ACT - TP- Les propriétés NP

2018–2019
Master 1 d'Informatique

Le voyageur de commerce

Objectif: L'objectif du TP est essentiellement de "concrétiser" les notions de propriété NP, de réduction polynomiale, de propriété et problème $NP-$durs. Il permet également d'aborder la génération d'objets combinatoires.

A faire: Le TP a trois sections, une sur la notion de NP, une sur celle de réduction polynomiale, une dernière (sans implémentation) sur le lien optimisation/décision. Le TP est à rendre la semaine du 19 novembre sous Prof sous forme d'une archive contenant le code et un (bref) rapport avec les réponses aux questions et vos choix d'implémentation. Vous disposez de quelques exemples de données pour tester.

Le problème central que l'on va étudier est TSP, le problème du voyageur de commerce (Traveling Salesman Problem) dont le but est de trouver une tournée de villes à visiter la plus courte possible. Le problème de décision est de décider, étant donné les distances entre n villes et une longueur maximale de tournée, si il existe une tournée qui visite les villes une et une seule fois, revient au point de départ, dont la longueur soit inférieure ou égale à la longueur donnée. Formellement, on a donc:

Travelling Salesman Problem (TSP)

Donnée

n, un entier – un nombre de villes

D, une matrice (n, n) d'entiers – elle représente les distances, qu'on suppose entières

l, un entier – la longueur maximale "autorisée", entière

Sortie

Oui, s'il existe une tournée possible, i.e.

$tour : [0 \dots n-1] \to [0 \dots n-1]$ – une **permutation** des n villes

telle que:

$D(tour[n-1], tour[0]) + \sum_{i=0}^{n-2} D(tour[i], tour[i+1])_i \leq l$,

Non, sinon.

Exemple: soit $n = 4$ et D donnée par les distances suivantes:

	A	B	C	D
A	0	2	5	7
B	7	0	8	1
C	2	1	0	9
D	2	2	8	0

La longueur minimale d'une tournée est 9 avec la tournée A, C, B, D, A. Donc, le problème aura une solution pour $l = 9$, il n'en aura pas pour $l = 8$.

Le problème TSP est très connu [1], très étudié et a de nombreuses applications - voir par exemple le site http://www.math.uwaterloo.ca/tsp/. Il y a de nombreuses versions du problème, par exemple:

- symétrique ou non, selon que l'on suppose que la distance entre i et j est la même que celle entre j et i.
- supposant ou non que les distances vérifient l'inégalité triangulaire (ce qui est le cas pour la notion mathématique de distance): $d(i, j) \leq d(i, k) + d(k, j)$.

  On se placera ici a priori dans le cas général (non nécessairement symétrique, sans condition sur la distance).

[1] Un film Traveling Salesman est sorti il y a quelques années, l'intrigue étant basée sur la résolution de ce problème.

1 Qu'est-ce qu'une propriété NP?

A faire avant d'implémenter!

Q 1. *La propriété est NP.*

NP

L est dit NP si il existe un polynôme Q, et un algorithme polynomial A à deux entrées et à valeurs booléennes tels que:

$$L = \{u / \exists c, A(c,u) = Vrai, |c| \leq Q(|u|)\}$$

Définir une notion de certificat pour le problème. Comment pensez-vous l'implémenter? Quelle sera la taille d'un certificat? La taille des certificats est-elle bien bornée polynomialement par rapport à la taille de l'entrée?

Proposez un algorithme de vérification associé. Est-il bien polynomial?

Ne vous pressez pas avant de fixer la notion de certificat et son implémentation; la suite de la section précise les fonctionnalités attendues pour un certificat et vous aidera à choisir votre implémentation.

Q 2. *NP = Non déterministe polynomial*

Q 2.1. *Génération aléatoire d'un certificat.* Proposez un algorithme de génération aléatoire de certificat, i.e. qui prend en entrée une instance du problème, et génère aléatoirement un certificat de façon à ce que, pour une instance donnée, chaque certificat ait une probabilité non nulle d'apparaître.

Pour aller plus loin: ici, il est uniquement demandé que chaque certificat ait une probabilité non nulle d'apparaître. Pour d'autres applications, on peut demander que tous les certificats soient équiprobables, i.e. aient la même probabilité d'apparaître. Est-ce le cas avec votre algorithme?

Q 2.2. Quel serait le schéma d'un algorithme non-déterministe polynomial pour le problème?

Q 3. *$NP \subset ExpTime$*

Q 3.1. Pour une instance donnée, pouvez-vous donner un ordre de grandeur du nombre de certificats?

Q 3.2. *Un ordre d'énumération*

Une méthode classique pour énumérer les certificats (soit définir un itérateur sur les certificats) consiste à s'appuyer sur une notion d'ordre total sur les certificats associés à une instance et donc ensuite les énumérer dans cet ordre, partant du "plus petit" pour l'ordre, et utilisant la notion de successeur pour passer d'un certificat au suivant. Quel ordre proposez-vous?

Q 3.3. *L'algorithme du British Museum* Comment déduire de ce qui précède un algorithme pour tester si le problème a une solution? Quelle complexité a cet algorithme?

Implémentation

On va donc implémenter les notions et algorithmes évoqués ci-dessus.

Attention: Le but ici est d'illustrer la notion de NP, et non de trouver le meilleur algorithme pour résoudre le problème d'optimisation; trouver des algorithmes efficaces pour TSP sera l'objet du prochain TP.

On devra donc être capable de lire une instance du problème, lire une proposition de certificat, vérifier si un certificat est valide, vérifier si le problème a une solution en essayant tous les certificats, "vérifier aléatoirement" si le problème a une solution en générant aléatoirement un certificat.

Sur le portail (cf fin du sujet) est proposé un embryon d'architecture et de programme de test en Java. **Vous pouvez en choisir une autre ou utiliser d'autres langages (C, CAML, HASKELL, PYTHON, ...)**. Vous veillerez à documenter votre code, à respecter l'esprit et à faciliter le test en respectant le schéma ci-dessous. Pour tester, on pourra donc avoir un programme qui lit l'instance du problème dans un fichier et:

. en mode "vérification" propose à l'utilisateur de saisir un certificat et vérifie sa validité.

. en mode "non-déterministe", génère aléatoirement un certificat, le teste et retourne Faux si il n'est pas valide, "Vrai" sinon (avec éventuellement la valeur du certificat).

. en mode "exploration exhaustive" génère tous les certificats jusqu'à en trouver un valide, si il en existe un et retourne Faux si il n'en existe pas de valide -la propriété n'est donc pas vérifiée- , "Vrai" sinon (avec éventuellement la valeur du certificat trouvé).

Vous avez à votre disposition quelques exemples de données. Attention, la longueur maximale (le l de la donnée dans la définition du problème) n'est pas dans le fichier. Donc, l'usage pourra être pour Java : java testTSP <file> <mode> <longueurMaxi> avec comme modes (au moins) -verif, -nondet, -exhaust.

Attention! Ne pas utiliser la version "exploration exhaustive" sur des problèmes de grande taille!

2 Réductions polynomiales

On va maintenant implémenter des réductions entre problèmes. On va étudier les problèmes de cycle et chemin hamiltoniens dans un graphe dirigé. On représentera ici les graphes sous forme de leur matrice d'adjacence.

Hamilton Cycle

Donnée

n, un entier – un nombre de sommets

D, une matrice (n, n) de booléens – elle représente les arêtes

Sortie

Oui, s'il existe un cycle hamiltonien, i.e. $ham : [0..n-1] \to [0..n-1]$ – une **permutation** des n villes telle que:

$D(ham(i), ham(i+1)) = true$, pour tout i, $0 \leq i \leq n-2$

$D(ham(n-1), ham(0)) = true$

Non, sinon.

Hamilton Path

Donnée

n, un entier – un nombre de sommets

D, une matrice (n, n) de booléens – elle représente les arêtes

Sortie

Oui, si il existe un chemin hamiltonien, i.e. $ham : [0..n-1] \to [0..n-1]$ – une **permutation** des n villes telle que $D(ham(i), ham(i+1)) = true$, pour tout i, $0 \leq i \leq n-2$

Non, sinon.

Ces deux propriétés sont connues $NP-$complètes.

Q 1. *Une première réduction*
Q 1.1. Montrer que $HamiltonCycle$ se réduit polynomialement dans TSP, i.e. $HamiltonCycle \leq_P TSP$.
Q 1.2. *[À coder]* Implémenter la réduction polynomiale de $HamiltonCycle$ dans TSP. Vous pouvez tester avec les données fournies.
Q 1.3. Qu'en déduire pour TSP?
Q 1.4. Pensez-vous que TSP se réduise polynomialement dans $HamiltonCycle$? Pourquoi?

Q 2. $HamiltonPath \leq_P HamiltonCycle$
Q 2.1. Montrer que $HamiltonPath$ se réduit polynomialement dans $HamiltonCycle$.
Q 2.2. *[À coder]* Implémenter une réduction polynomiale $HamiltonPath$ dans $HamiltonCycle$. Vous pouvez tester avec les données fournies.

Q 3. Montrer que $HamiltonPath$ se réduit polynomialement dans TSP.

Q 4. *Bonus* Les notions de cycle et chemin hamiltonien sont souvent définies dans le cas des graphes non dirigés. Que pensez-vous de la complexité des propriétés dans ce cadre?

Propriétés versus Problèmes d'Optimisation

Au problème de décision TSP, on peut associer deux problèmes d'optimisation:

TSPOpt1

Donnée

n, un entier – un nombre de villes

D, une matrice (n, n) d'entiers – elle représente les distances, qu'on suppose entières

Sortie

l, minimale telle qu'il existe une tournée possible, i.e.

$tour : [0 \dots n-1] \to [0 \dots n-1]$ – une **permutation** des n villes

telle que:

$D(tour[n-1], tour[0]) + \sum_{i=0}^{n-2} D(tour[i], tour[i+1]) = l$,

TSPOpt2

Donnée

n, un entier – un nombre de villes

D, une matrice (n, n) d'entiers – elle représente les distances, qu'on suppose entières

Sortie

Une tournée de longueur minimale, i.e.

$tour : [0 \dots n-1] \to [0 \dots n-1]$ – une **permutation** des n villes

telle que:

$D(tour[n-1], tour[0]) + \sum_{i=0}^{n-2} D(tour[i], tour[i+1])$ soit minimale dans les tournées possibles.

Q 5. Montrer que si $TSPOpt1$ (resp. $TSPOpt2$) était P, la propriété TSP le serait aussi ; qu'en déduire pour $TSPOpt1$ (resp. $TSPOpt2$)?
Q 6. Montrer que si la propriété TSP était P, $TSPOpt1$ le serait aussi.
Q 7. *Plus dur...* Montrer que si la propriété TSP était P, $TSPOpt2$ le serait aussi.

Voici un embryon d'architecture qui **n'est qu'un exemple. Elle peut être ignorée/modifiée/améliorée.**

```
//la classe abstraite des problèmes de décision...
public abstract class PbDecision { public abstract boolean aUneSolution(); }

public interface Certificat {
//saisie au clavier de la valeur du certificat
public void saisie();

//affichage du certificat
public void display();

//modification aleatoire de la valeur du certificat
//chaque valeur possible doit pouvoir etre generee
public  void alea();

//on munira les valeurs possibles du certificat pour une instance d'un ordre total pour l'enumeratio

//affecte au  certificat la premiere valeur pour l'ordre choisi
public void reset();

//retourne vrai si la valeur est la derniere dans l'ordre choisi, faux sinon
public boolean estDernier();

//modifie la valeur du certificat en la suivante pour l'ordre, pas defini si la certificat est le de
public void suivant();

//retourne True Ssi le certificat  est correct pour l'instance du pb associée
//remarque: un certificat est donc ici associé toujours a une instance du pb
public  boolean estCorrect();
}

... class PblTSP extends PblDecision{   /* A compléter*/}

//la notion de certificat pour le problème JSP
... class  CertificatTSP implements Certificat{/*A compléter, doit implémenter l'interface}

//La partie reduction
....class PblCycleHamilton  extends PblDec{
...
//reduction polynomiale qui retourne  une instance équivalente de TSP
public PblTSP redPolyToTSP(){...}

// en une ligne en utilisant redPolyToJSP()  et aUneSolution de JSP
public boolean aUneSolution() {...}
}

....class PblCheminHamilton  extends PblDecision{
...
    //les reductions polynomiales
    public PblCycleHamilton redPolyToCycleHamilton(){...}
    public PblTSP redPolyToTSP(){.../*en une ligne*/...}

    public boolean aUneSolution() {.../*en une ligne*/...}
}
```

```
//test TSP...
public static void main(String[] arg) throws Exception {
   if (arg.length < 3)
    System.out.println("java test  mode file.atsp lg");
   else {
        int lg=Integer.parseInt(arg[2]);  //la longueur maximale est  le dernier argument
        //lire l'instance du probleme dans le fichier de donnees dont le nom est en argument
        //créer  l'instance du probleme...
     }
   // les differents modes
   if (arg[0].equals("-verif")) {
          //lire un cetificat proposé,  sortir le résultat de la vérification
          }
   else if (arg[0].equals("-nondet")) {
          //générer aléatoirement un certificat
          // sortir le résultat de la vérification et evt le certificat
          }
   else if (arg[0].equals("-exhaust"))  {
         //générer tous les certificats jusqu'au dernier ou jusqu'à un trouver un de valide
         //sortir le résultat  et evt le certificat valide   ...
    }
   else System.out.println("erreur de mode");
}
```