IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

ITD - Cas d'application
Le problème du voyageur de commerce

IMT Atlantique - cursus ingénieur - année 2
Gilles Simonin
D'après un travail de Damien Prot - Axel Grimault

Présentation du problème

Le problème du voyageur de commerce

- On considère un ensemble de N villes à visiter et l'ensemble des distances entre chaque paire de ville. Le problème du voyageur de commerce est de trouver le circuit le plus court permettant de visiter l'ensemble des villes.

Le problème est modélisé à l'aide d'un graphe G=(V,A) où les sommets représentent les villes et les arcs représentent les chemins. Un poids, correspondant à la distance entre les deux extrémités, est associé à chaque arc. La solution optimale correspond au circuit de poids minimal qui passe par l'ensemble des sommets dans le graphe.

Modèle linéaire

Soit c le vecteur de coût de chaque arcs de A.
Soit x le vecteur d'incidence tel que :
$\forall e \in A, x_e = 1$ si l'arête e est dans le circuit et $x_e = 0$ sinon.

Le TSP peut être décrit de la manière suivante :

$$\text{minimise } z = \sum_{e \in A} c_e . x_e$$

tel que :

$$\sum_{e \in \delta(i)} x_e = 2 \qquad \forall i \in V$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subset V, S \notin V$$

$$x_e \in \{0, 1\} \qquad \forall e \in E$$

Modèle linéaire

Le TSP peut être décrit de la manière suivante :

$$\text{minimise } z = \sum_{e \in A} c_e . x_e \tag{1}$$

$$\text{tel que :}$$

$$\sum_{e \in \delta(i)} x_e = 2 \qquad \forall i \in V \tag{2}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subset V, S \neq V \tag{3}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{4}$$

avec $\delta(i)$ l'ensemble des arcs adjacents à i
et $E(S)$ l'ensemble des arcs ayant les deux extrémités dans S

Le TSP

- En anglais : Traveling Salesman Problem.
- Un des terrains de jeux les plus fréquentés en optimisation.
- TSPLib: librairie d'instances. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
- Exemple : circuit optimal sur 24978 villes en Suède : (source www.tsp.gatech.edu).

Ressources bibliographiques

- D.L. Applegate, R.E. Bixby, V. Chvatal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.
- G. Reinelt. The Traveling Salesman: Computational Solutions for TSP Applications, volume 840 of Lecture Notes in Computer Science (LNCS). Springer-Verlag, Berlin Heidelberg, Heidelberg, 1994.
  http://www.springerlink.com/content/br49t9h4l7mm/?MUD=MP

Objet de l'exercice

- Concevoir une méthode heuristique permettant de résoudre des problèmes de TSP.
- A votre disposition :
  - Un *framework* logiciel en Java : un ensemble de librairies et classes permettant de lire une *instance* du TSP et d'en visualiser la solution. Il faut y ajouter l'algorithme de résolution.
  - 10 *instances* test : 10 jeux de données représentant différentes versions du problème à résoudre.
- Contrainte :
  - La méthode ne doit pas dépasser la limite de temps (60sec. pour l'instant).
  - Pas d'appel à des librairies extérieures (ex.: glpk) : que du Java.

Déroulement du travail

- 13h45 en salle TP + 15h de travail personnel.
- 20 groupes de 3 et 2 groupes de 4
- Pas de groupes composés uniquement de doubles diplômes ou échanges de crédits.

- **Rendu**
  - Deadline : Lundi 5 novembre à 23H32, dépôt sur Campus.
  - Une archive nommée TSP_nom1_nom2_nom3 contenant l'ensemble des classes développées (il faut rendre les fichiers .java).
  - Un rapport de 4 pages maximum présentant le ou les algorithmes développés, une bibliographie sérieuse, un code commenté et propre, puis enfin une synthèse des expérimentations.

Évaluation du travail

- **Noté sur 20 points :**
  - 10 points pour le rapport + les commentaires/javadoc du code
  - 5 points sur le travail individuel lors des séances + gestion du GitHub
  - 5 points sur la performance
- **Compétition :**
  - Si le programme ne fonctionne pas, ou pas comme demandé : 0.
  - Performance en dessous d'un certain seuil de l'optimal : 2.
  - Une seule poule entre tous :
    - Répartition des points selon une gaussienne.

Points	Rangs
3	1
2,5	2-4
2	5-8
1,5	9-13
1	14-17
0,5	18-20
0	21-22

Comparaison des joueurs

- La compétition se déroule sur 10 nouvelles instances ayant les mêmes caractéristiques que les 10 instances test.
- Sur une instance :
  - Un certain nombre de points est attribué en fonction du rang dans la poule :

Rang :	**1**	**2**	**3**	**4**	**5**	...
Points :	6	4	2	1	0	...

- Le score final de chaque équipe est calculé sur la somme des points obtenus.

Présentation du framework

- **Disponible à l'adresse : Framework from hell**

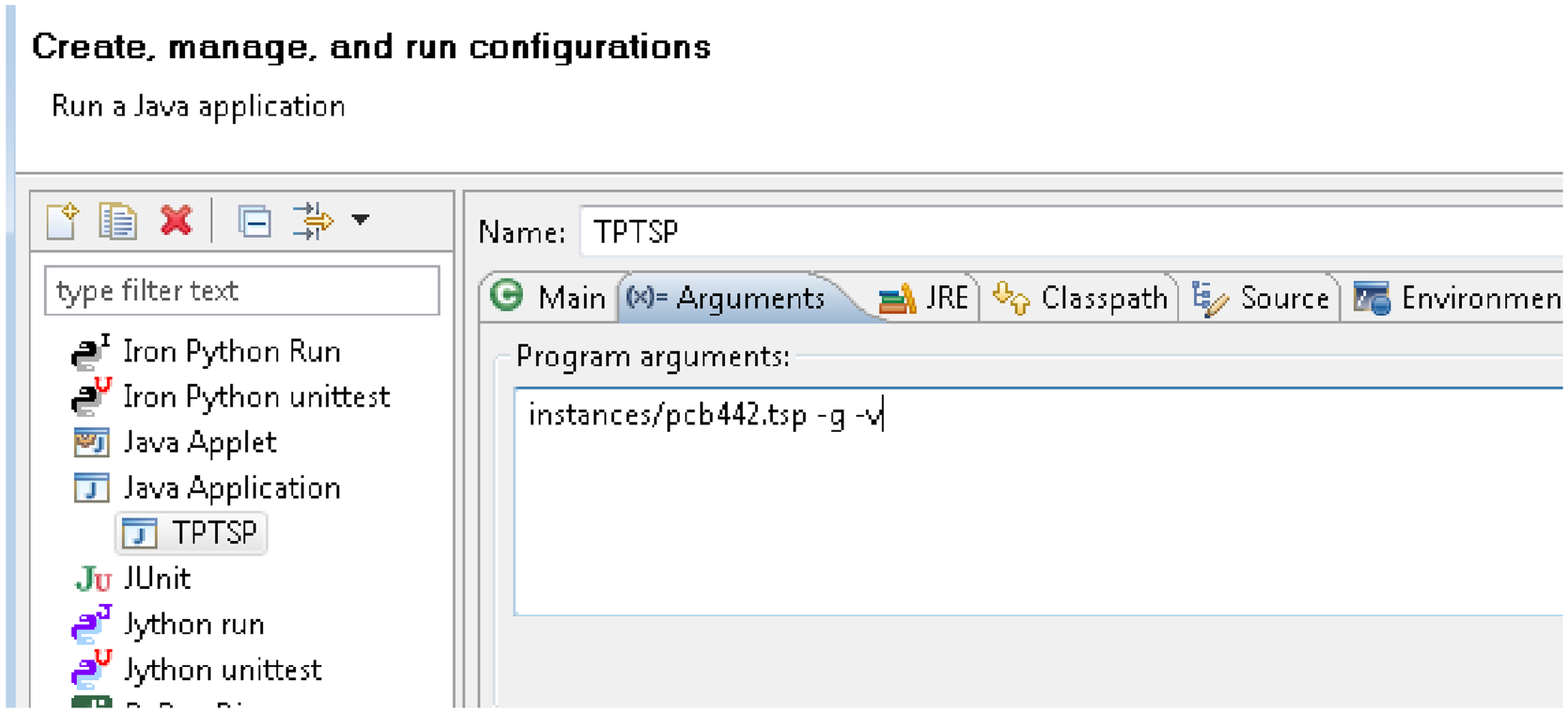
- **Composition du répertoire src :**
  - **Instance.java :** contient les données et le parser de fichier. **NE PAS MODIFIER**
  - **Solution.java :** modélise la solution.
  - **Drawing.java :** réalise la représentation graphique de la solution.
  - **Main.java :** point d'entrée dans le programme (contient la fonction main). **NE PAS MODIFIER**
  - ***TSPSolver.java : contient la méthode solde() à implémenter.***

Présentation du framework

- Composition (suite) :
  - Répertoire instances :
    - Contient les 10 instances test (fichiers .tsp).
  - Répertoire lib
    - visuBeta.jar : librairie de visualisation.
  - Répertoire doc :
    - Contient la javadoc du framework.

Projet Eclipse

- Lancer le programme depuis Eclipse
  - Menu Run >> run configuration >> configuration TPTDP >> onglet Arguments : entrer le chemin vers l'instance à résoudre.
  - Exemple :



Lancement

- Le programme est lancé par la commande :
  - java Principale [options] datafile.tsp
- Options:
  - -help : imprime cette aide
  - -t (int) : temps maximum alloué à la résolution
  - -v : trace, niveau d'impression
  - -g : affichage graphique de la solution

Coder son algorithme

- La classe TSPSolver : c'est la classe à modifier.

- Un objet TSPSolver est créé par la classe principal.
  - Cet objet est initialisé dans Principale avec :
    - TSPSolver.m_instance : l'objet Instance qui contient les données du problème.
    - TSPSolver.m_solution *(à modifier)* : l'objet Solution qui contient la solution qui sera évaluée et retournée par le programme.
    - TSPSolver.m_time le temps maximum alloué au programme.
  - Ensuite Principale lance TSPSolver.solve()

- Il faut modifier la méthode TSPSolver.solve()

Coder son algorithme

- **Exemple :**

```
public void  solve()  throws  Exception {
    // Exemple simpliste où le sommet  i est inséré
    // en position i dans la tournée.
    for (int i=1; i <= m_instance.getNbCities(); i++)
    {
        m_solution.setCityPosition(i, i);
    }
}
```

- m_solution.setCityPosition(s, i)
  permet de modifier la solution courante.

Respecter la limite de temps

```
public void solve() throws Exception {
  long t = System.currentTimeMillis();
  long tempspasse = 0;
  while (tempspasse < m_time * 1000)
        …
        …
        tempspasse = System.currentTimeMillis() - t;
  }
}
```

Sorties

- La sortie standard est réservé au résultat affiché dans la classe Principale
- Si vous écrivez sur System.out vous changez le résultat et vous avez 0
- Pour suivre le déroulement de votre algorithme, écrivez sur la sortie System.err

```
System.err.print(" … ");
System.err.println(" … ");
```

Sortie attendue

- IMPERATIF : Si on exécute votre programme, sur System.out, on doit avoir une sortie du type :

```
..\instances\eil51.tsp;1308;19500;0
```

et rien d'autre

- Signification de la sortie :

```
Nom instance;Valeur objectif;temps (ms);code erreur
```

Explication détaillée des classes

- Lire la javadoc

Créer de nouveaux objets

- Vous pouvez parfaitement créer :
  - De nouvelles classes
    - Pour rendre plus facile / plus claire la programmation
  - Une nouvelle structure de donnée pour stocker la solution
    - Mais il faut renseigner la structure existante à la fin
- A la fin il faut rendre :
  - La classe TSPSolver.java modifiée
  - Les classes que vous avez créées

À propos du rapport

- Maximum 4 pages
- Décrire les algorithmes testés / choisis
- Faire la synthèse des expérimentations sur les instances test
- N'oubliez pas de citer vos sources si vous faites de la bibliographie / des recherches internet
- Optionnel : 4 pages d'annexes maximum pour des tableaux de résultats ou algorithmes plus détaillés
- Votre code doit être commenté, en utilisant la javadoc

Le délivrable

- **Rendu**
  - **Deadline** : Lundi 5 novembre à 23H32, dépôt sur Campus.
  - Une **archive** nommée **TSP_nom1_nom2_nom3** contenant l'ensemble des classes développées (il faut rendre les fichiers .java).
  - Un rapport de 4 pages maximum présentant le ou les algorithmes développés, une bibliographie sérieuse, un code commenté et propre, puis enfin une synthèse des expérimentations.