Théorie de la complexité des algorithmes
Problème des K-centres

FRANÇOIS HERNANDEZ - LÉO PONS
CentraleSupélec
February 6, 2017

Contents

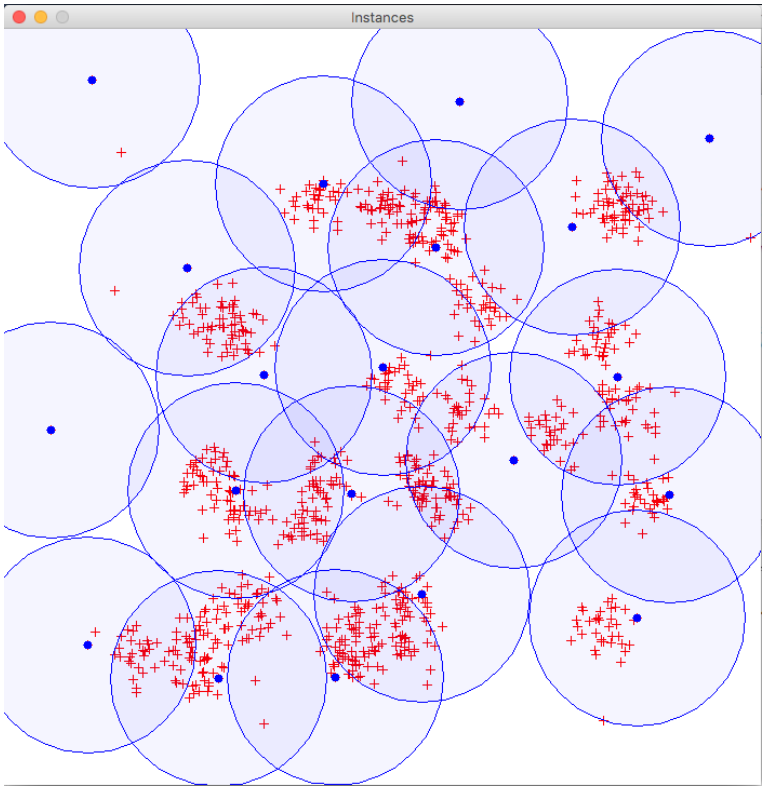
Introduction

Le problème des K-centres est un problème d'optimisation combinatoire. Le problème peut se décrire de façon informelle ainsi (Wikipedia) : étant donné n villes, il faut ouvrir une caserne de pompiers dans k villes, tel que la distance entre chaque ville et la plus proche caserne soit minimisée.

Il existe plusieurs définitions des problèmes K-centres. Ici, nous considérons la formulation suivante. Supposons un ensemble E de n points dans un espace vectoriel ou un graphe complet muni d'une fonction de distance satisfaisant l'inégalité triangulaire. Il s'agit de trouver k "centroids" afin de minimiser la distance maximale entre un point de E et le centre le plus proche.

Ce problème de décision est NP-complet.

Au cours de ce projet, nous avons mis en place un programme en Java constituant une représentation du problème, et proposant différents algorithmes de résolution sur différents ensembles d'instances.



I Implémentation

Nous avons choisi d'implémenter ce problème dans le langage Java. Nous avons mis en place des fonctions de génération et d'affichage d'instances (ensemble de points), ainsi que différents algorithmes de résolution à tester, et des méthodes de lecture et écriture des données sous format texte.

Le programme comporte ainsi les packages suivants :

- `Generation` : contient les différentes classes de génération d'instances, ainsi que celle associée à la lecture des fichiers .txt ;
- `Graphics` : contient les différentes classes associées à l'affichage des instances ;
- `Resolution` : contient les différents algorithmes de résolution ;
- `Main` : contient le main qui permet de lancer les algorithmes de test, ainsi que la définition des différents objets utilisés ;

I.1 Objets utilisés

Les objets utilisés dans ce programme sont définis de la façon suivante :

- `Point` : ensemble de coordonnées, ici `x` et `y` dans l'espace à deux dimensions ;
- `Instance` ensemble de n `Points` sous forme d'`ArrayList`, et un attribut k définissant le nombre de centres attendus, contient également une méthode permettant d'exporter l'instance au format .txt ;
- `Solution` : ensemble de k `Points` constituant les centres d'une `Instance`, ainsi qu'un attribut *rayon* définissant le rayon minimal pour lequel tous les points de l'instance sont couverts.

I.2 Génération d'instances

Nous avons choisi d'implémenter deux méthodes principales de génération d'instances :

- `Uniforme` : génère n `Points` de façon aléatoire dans l'espace défini (en conservant des marges afin de ne pas avoir de points trop proches des bords, pour des raisons d'affichage) ;
- `Cluster` : détermine n_{Cl} centres de clusters de façon aléatoire dans l'espace défini (en conservant une nouvelle fois des marges). On génère ensuite n `Points` répartis uniformément sur les différents clusters. Les coordonnées de chaque point sont déterminés aléatoirement selon une loi normale, centrée sur le cluster correspondant et d'écart-type `sigma`. La taille des clusters est donc définie par `sigma` et est commune à tous les clusters. Elle est définie à la création de l'objet.

I.3 Affichage des données

Afin d'avoir une représentation graphique claire des différentes instances et des résultats des algorithmes de résolution, nous avons choisi de définir une interface graphique à l'aide des outils `Swing`. Cela consiste en deux classes, `Fenetre` et `Panel`. `Fenetre` hérite de la classe `JFrame` et définit les caractéristiques de la fenêtre qui contiendra le `Panel`. `Panel` hérite de la classe `JPanel` et contient les méthodes associées au dessin des représentations. La méthode `paintComponent` dessine les différents éléments dans un `bufferGraphics`, attribut d'une image.

I.4 Algorithmes de résolution

LEO

I.5 Conversion en fichier texte

Un standard d'export des instances a été défini afin de pouvoir sauvegarder et échanger différentes instances de test. Celui-ci est le suivant.

```
nombre d'instances i

n1, k1
x1, y1, x2, y2, [...], xn1, yn1

n2, k2
x1, y1, x2, y2, [...], xn2, yn2

[...]

ni, ki
x1, y1, x2, y2, [...], xni, yni
```

La méthode `createFile` de la classe `Instance` permet de créer de tels fichiers à l'aide de la classe `FileWriter` de Java.

La classe `Importer` du package `Generation` permet de lire de tels fichiers à l'aide des classes `FileReader` et `BufferedReader` de Java.

II Théorie, NP-complétude et approximation

III Méthodes de résolution

III.1 DeuxApprox

III.2 Descente

III.3 Exact

III.4 Dominant

IV Évaluation des performances

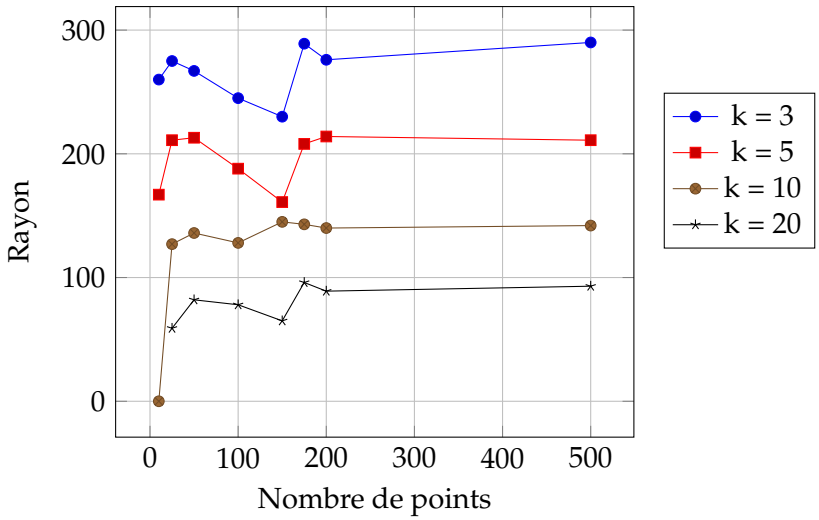


Figure 1: Rayon en fonction du nombre de points et du nombre de centres