Théorie de la complexité des algorithmes
Problème des K-centres

FRANÇOIS HERNANDEZ - LÉO PONS
CentraleSupélec
February 6, 2017

Contents

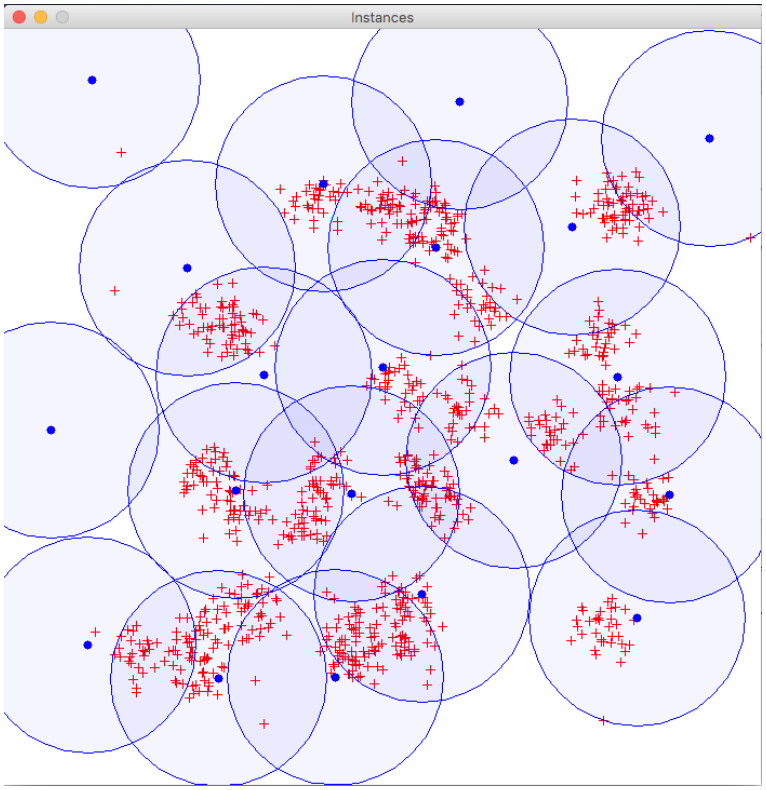
Introduction

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, dans sa forme 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, toutes héritant de la classe abstraite `Generateur` ;
- `Graphics` : contient les différentes classes associées à l'affichage des instances ;
- `Resolution` : contient les différents algorithmes de résolution sous forme de classes filles de la classe générique abstraite `Algo` ;
- `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 classes 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

Nous avons implémenté 4 algorithmes de résolution différents *(détails en partie III)* :

- `Exact` : Calcule exactement la solution optimale. L'algorithme teste naïvement toutes les solutions possibles récursivement et est donc très coûteux en temps.
- `DeuxApprox` (approximation) : On construit la solution centre par centre : on commence par un point quelconque puis on cherche parmi les points non encore choisis celui qui est le plus éloigné de notre ensemble de centres. On itère ainsi jusqu'à avoir choisi k centres.
- `Dominant` (approximation) : On commence par calculer la liste des distances entre tous les points, qu'on trie par ordre croissant. Ensuite, on prend la première distance et on cherche une solution avec des disques de rayons égaux à cette distance. Si on trouve, on renvoie la solution, sinon, on passe à la distance suivante.
- `Descente` : Cet algorithme est un peu particulier : il améliore une solution déjà existante (en testant des permutations de centres).

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

[...]
```

```
11 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

Le problème K-centre, dans sa forme de problème de décision, est NP-complet, c'est à dire qu'il suit les conditions suivantes :

- il est possible de vérifier une solution en temps polynomial ;
- il est possible de le réduire en un problème de la classe NP par une réduction polynomiale.

II.1 Problème d'ensemble dominant

Soit un graphe $G = (S, A)$. Un ensemble dominant pour G est un sous-ensemble D de l'ensemble des sommets S de G tel que tout sommet qui n'appartient pas au dominant soit adjacent à un de ses sommets.

Le problème d'ensemble dominant consiste à déterminer, selon G et un entier naturel k, si G possède un ensemble dominant d'au plus k sommets.

$dom(G)$ est la taille de l'ensemble dominant le plus petit possible pour le graphe G. Trouver un ensemble dominant de taille minimale est un problème NP-complet. Il est possible de réduire le problème K-centre en un problème d'ensemble dominant afin de démontrer sa NP-complétude.

II.2 Réduction

Soit un graphe non orienté $G = (S, A)$ pour le problème de l'ensemble dominant. Considérons le graphe $G' = (S, S \times S)$ muni de la fonction de poids $d : S \times S \to \mathbb{R}$ pour ses arrêtes.

$$d(u,v) = \begin{cases} 1 & \text{si } (u,v) \in A \\ 2alpha & \text{sinon} \end{cases} \tag{1}$$

Supposons que G ait un ensemble dominant de taille inférieure ou égale à k.

$$dom(G) \leq k$$

Dans ce cas, il existe une solution de poids 1 au problème k-centre pour G'. Un algorithme d'α-approximation fournit une solution de poids inférieur ou égal à α, avec $\alpha \geq 1$ le facteur d'approximation.

S'il n'existe pas un tel ensemble dominant dans G, alors toutes les instances de k-centre auront un poids supérieur ou égal à 2α, donc supérieur à α.

Supposons qu'il existe un algorithme d'α-approximation pour le problème k-centre. Appliquons cet algorithme de décision sur G'. La solution a un poids inférieur ou égal à α, donc il existe un ensemble dominant de taille inférieure ou égale à k. Autrement, il n'existe pas de tel ensemble dominant. Il y a donc contradiction. Le problème k-centre peut se réduire au problème de l'ensemble dominant, donc il est NP-complet.

III Méthodes de résolution

III.1 Exact

Un premier algorithme est développé pour trouver la solution optimale d'une instance donnée, c'est à dire le choix de k centres qui minimise la distance maximale d'un point aux centres.

On procède par récursivité : On construit une fonction qui renvoie une solution optimale (minimisant le rayon) pour une instance donnée sachant qu'une liste de centres est déjà fixée et qu'il reste un certain nombre de centres à choisir. Cette fonction fonctionne en testant tour à tour le choix des points restants et en s'appelant récursivement avec un centre de moins à choisir. On peut donc sélectionner le meilleur point et utiliser le résultat de l'appel récursif pour les points suivant.

Notre solution principale est ainsi obtenue en appelant la fonction récursive avec le nombre de points à choisir original et avec une liste de points fixés vide. Cet algorithme est optimal mais très coûteux en terme de temps de calcul, ce qu'il le rend vite inutilisable.

III.2 DeuxApprox

Ce deuxième algorithme est une approximation qui permet de diminuer la complexité d'éxecution de l'algorithme exact. On construit la solution centre par centre : on commence par un point quelconque puis on cherche parmi les points non encore choisis celui qui est le plus éloigné de notre ensemble de centres. On itère ainsi jusqu'à avoir choisi tous les centres.

Cet algorithme ne donne qu'un résultat approché (non optimal) mais est nettement moins gourmand en calcul, ce qui le rend largement plus utilisable que l'algorithme exact.

III.3 Dominant

Ce deuxième algorithme est une approximation, un peu plus exigeante que la précédente en termes de calcul mais donnant de meilleurs résultats. Il est basé sur la réduction au dominant qui a été présentée en partie II.

On commence par calculer la liste des distances entre tous les points, qu'on trie par ordre croissant. Ensuite, on prend la première distance et on cherche une solution : on construit notre liste centre par centre, en choisissant à chaque fois le point qui possède le plus de voisins non couverts (par des disques de cette distance). Après avoir choisi k centres, on regarde si tous les points sont couverts. Si oui, on renvoie la solution, sinon, on passe à la distance suivante.

III.4 Descente

Cet algorithme est un peu particulier : il améliore une solution déjà existante. Il faut donc commencer par choisir un algorithme qui déterminera la solution à manipuler, on pourra choisir notamment un des deux algorithmes approximatifs précédemment décrits.

Pour chaque centre dans la solution obtenue par le premier algorithme, on va essayer de le permuter tour à tour avec les autres points de l'instance. Si la permutation améliore la solution, on la retient et on recommence. On s'arrête quand plus aucun échange n'améliore la solution.

Cet algorithme est assez rapide et efficace, et il surtout toujours bon à appliquer à une solution obtenue par une autre heuristique.

IV Évaluation des performances

Tests pour 10 clusters de taille 15.

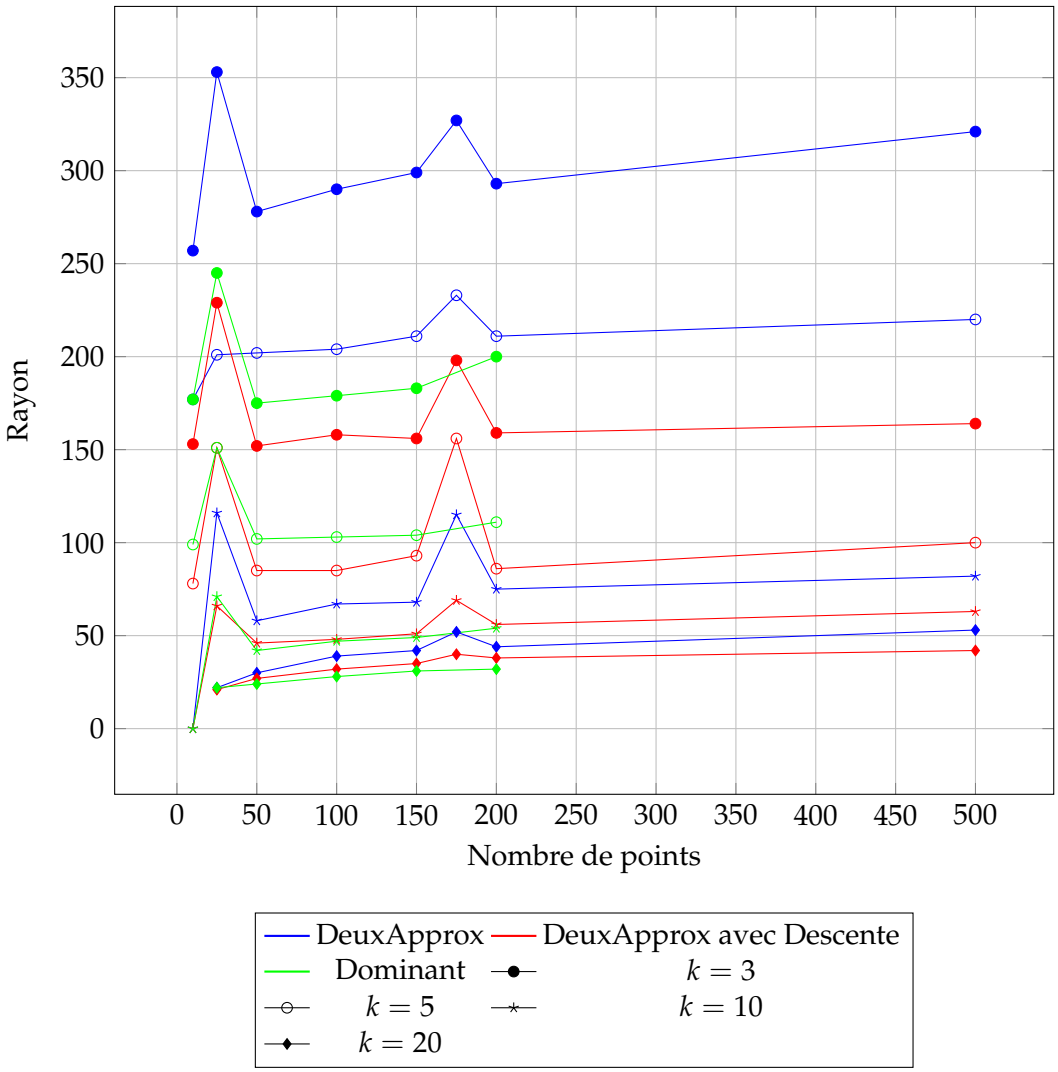


Figure 1: Rayon en fonction du nombre de points et du nombre de centres (10 clusters)

Tests avec des instances uniformes

V Évaluation des performances

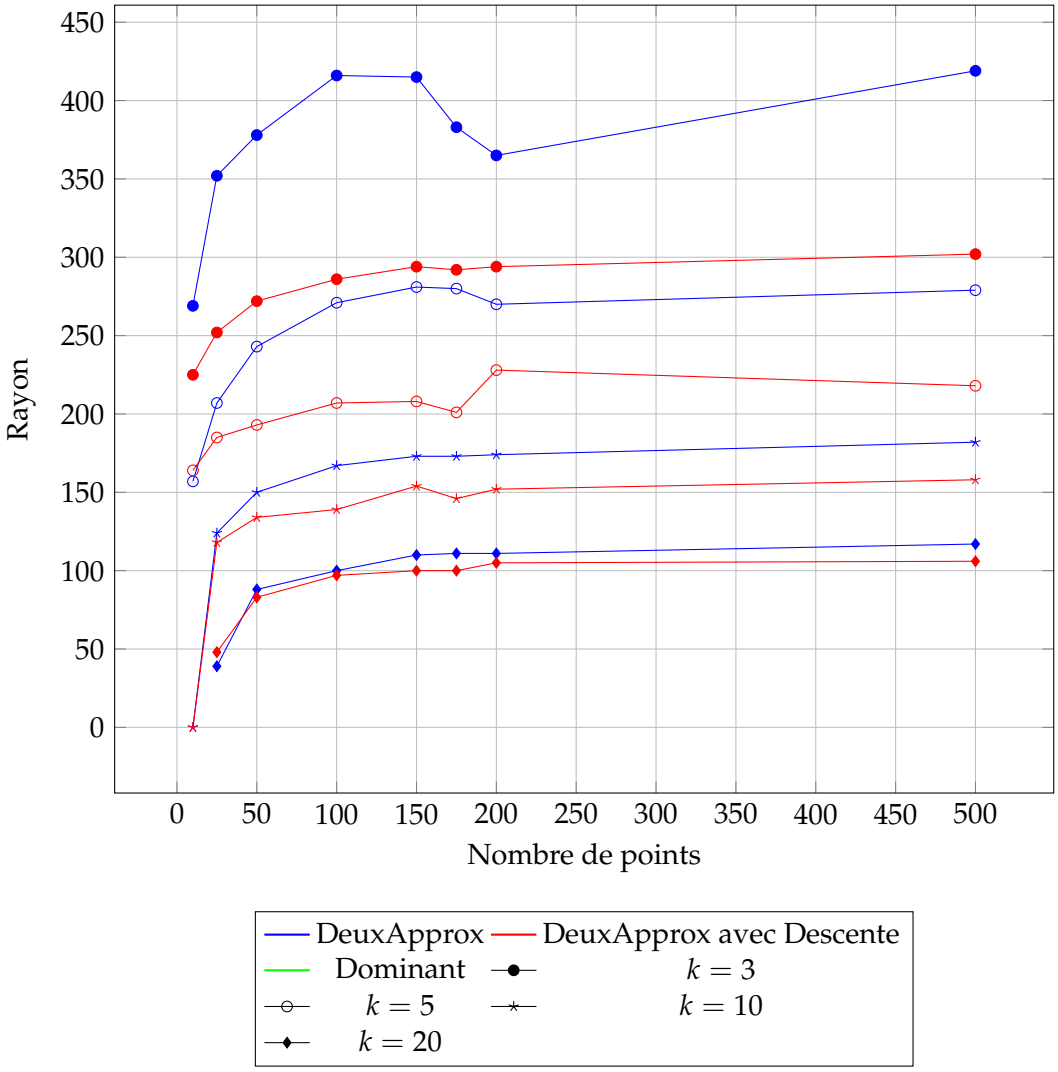


Figure 2: Rayon en fonction du nombre de points et du nombre de centres (Instances uniformes)

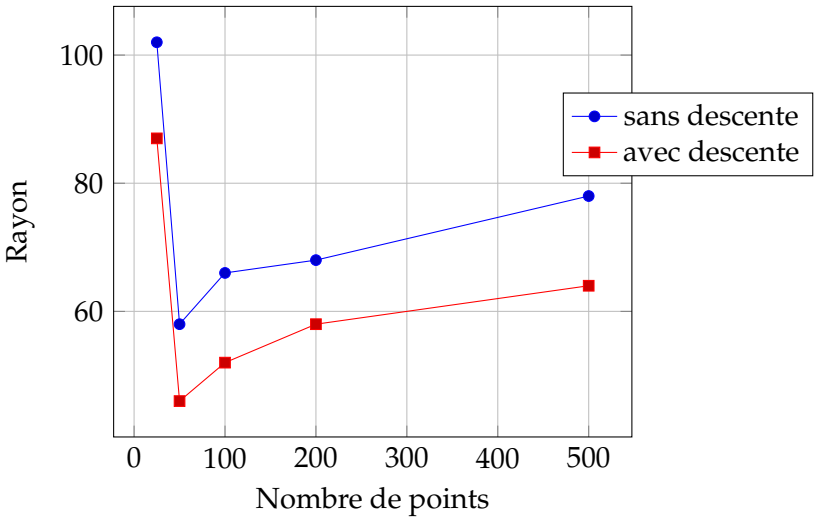


Figure 3: Léo

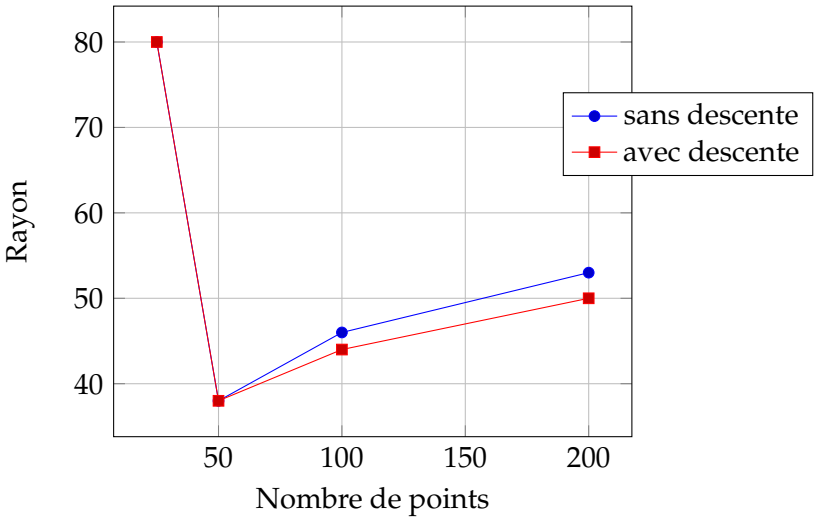


Figure 4: Léo