UNIVERSITÉ DE
SHERBROOKE

Projet Approximation - Etude de k-centres

IFT 800 Algorithmique

Eliott THOMAS — 21 164 874
thoe2303@usherbrooke.ca

Travail présenté à
Manuel Lafond

Université de Sherbrooke
Département d'informatique
Date de remise : 28 octobre 2021

Table des matières

1 Introduction

Dans ce rapport, nous allons présenter 2 solutions au problème des k-centres.

Le problème des k-centres est le suivant : On possède un nuage de p points (formalisé par un graphe complet), chaque point étant en n dimensions dans la version la plus générale du problème. Le but est de trouver k centres parmi les p points tel qu'ils soient les "*mieux*" séparés possible.

Hypothèses de travail :

1) Il existe plusieurs moyen de juger la séparation des "*clusters*" ainsi formés. Le critère utilisé pour évaluer la séparation des clusters sera la distance maximale d'un point au centre le plus proche.

Formellement cela donne : $max_{v \in V} min_{c \in C} dist(v,c)$ Avec :

— V : l'ensemble des sommets du graphe
— C : l'ensemble des centres retenus par l'algorithme
— dist : une fonction d'évaluation de distance entre 2 sommets du graphe.

2) Il a également été choisi que chaque point aurait uniquement 2 dimensions plutôt que n. Ceci se fera sans perte de généralité mais permettra de nécessiter moins de puissance de calcul. Par ailleurs, il est ainsi possible de visualiser la séparation des clusters en 2 dimensions, contrairement au même problème pour $n > 3$.

3) Les distances entre les sommets doivent vérifier l'inégalité triangulaire.

Finalement, la suite du rapport sera construite de la manière suivante : 2 algorithmes d'approximations seront présentés et comparés. Ils seront jugés sur leur ratio d'approximation, dans le pire des cas et en moyenne.

2 2-approximation vue en cours

2.1 Principe de l'algorithme Il s'agit de la 2-approximation vue en cours.

Le principe est le suivant :

1. On commence par calculer les distance entre tous les points et les ordonner en ordre croissant
2. Puis on itère sur ces distances pour trouver les c=|C| centres des c clusters.
3. Puisqu'on souhaite k centres, on ne retourne la solution que si c <=k. Si on souhaite exactement k centres il suffit de compléter arbitrairement jusqu'à k centres et on ne perd pas de généralité.
4. Pour trouver les centres pour une distance donnée, on doit annoter tous les points comme centre ou voisin d'un centre. On place un centre puis son voisinage, puis un autre centre. On continue comme cela jusqu'à ce qu'on ai annoté tous les points.

On trouve ci-dessous le pseudo code formalisant les 2 fonctions principales nécessaires au bon fonctionnement de l'algorithme.

```
getCentres(P, dj)
    C = {}
    TANT QUE |P|>0:
        Soit p dans P
        Soit R = { q dans P : D(p,q) <=dj}
        C.insert(p)
        P = P \ {R}
    FIN TANT QUE
    return C

kCentreAprox(P,k):
    Calculer d1, ...., dm
    POUR i de 1 a m :
        C = getCentres(P, 2 di)
        Si |C| <= k
            return C
    FIN POUR
```

2.2 Implémentation de l'approximation Vous pourrez retrouver l'intégralité du code commenté en annexe. Les parties jugées pertinentes à détailler le seront ici.

```
12  centers = [(10, 80), (20, 20),(80, 20), (80, 60)] # les coordonnées des centres des blobs
13  cluster_std = [8, 10,5,7] # les écarts type de chaque blob (plus d'écart = plus d'eclatement du blob)
14
15  P, y = make_blobs(n_samples=10, cluster_std=cluster_std, centers=centers, n_features=k,random_state=42)
16  colors=["red", "blue", "green", "purple"] # pour pouvoir différencier les blobs
17
18  P=P.tolist() # pour transformer l'array en list.
```

FIGURE 1 – Structure du code pour la mise en place des points

Cette structure de code est reprise dans les 2 implémentations. Elle permet de créer artificiellement un certain nombre de clusters, ici 4. Il a été décider d'imposer un nombre limité de points car la vérification de l'optimal se fait grâce à un algorithme de force brute qui teste toutes les solutions et la meilleure en ressort. C'est en particulier nécessaires lors des tests sur de nombreuses instances.

2.3 Analyse de l'approximation Dans cette sous-section, nous allons analyser les performances de cette approximation en moyenne et dans le pire des cas.

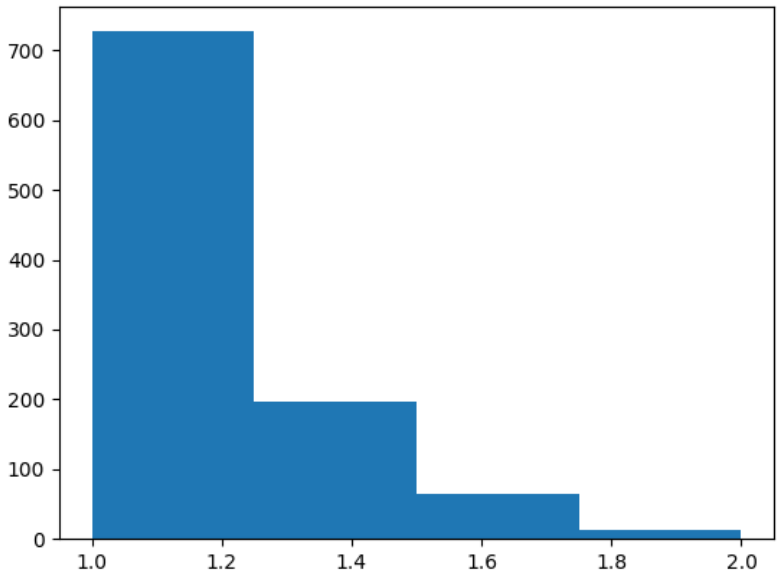

FIGURE 2 – Répartition des score d'approximation pour la 2-approximation vue en cours sur 1000 instances

Grâce à l'histogramme présenté en figure 2, on peut voir que dans plus de 70% des cas, l'algorithme retourne au maximum 1.25 fois la solution optimale.

Dans le diagramme en bâton présenté en figure 3, on observe même que en moyenne, le ratio du premier algorithme est 1.18 fois pire que l'optimal. De plus, la courbe rouge permet de confirmer en pratique que l'on a bien une 2-approximation. On a bien max_APP qui tend vers 2.

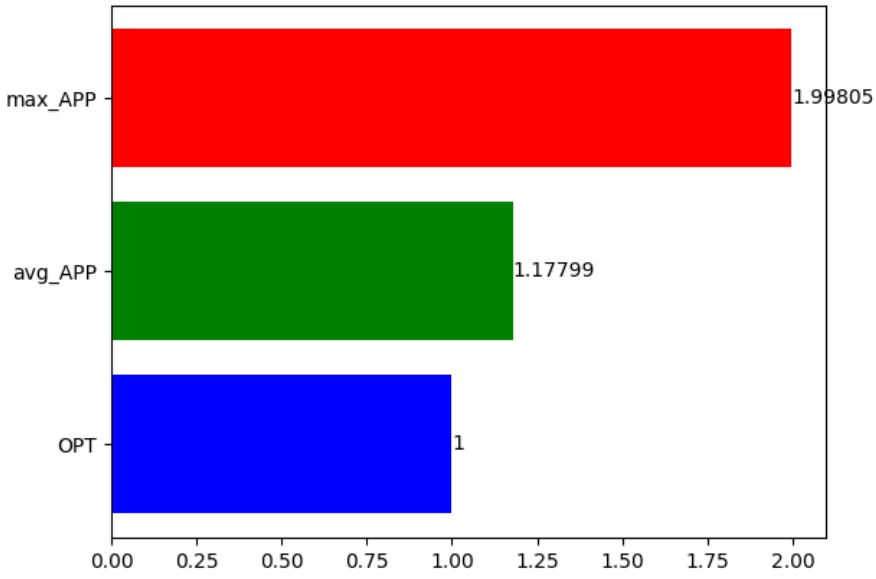

FIGURE 3 – Comparaison entre la solution optimale, le ratio en moyenne et le ratio dans le pire des cas sur 1000 instances

3 Best Possible Heuristic

L'algorithme présenté ci-après a été mis en place par Dorit S. Hochbaum et David B. Shmoys dans *Mathematics of Operations Research (Mai 1985).*

3.1 Principe de l'algorithme Pour pouvoir aborder la suite de la présentation, il faut commencer par introduire certaines notions propres aux graphes.

- Ensemble indépendant : Il n'y a aucun sommet adjacent dans l'ensemble I.
- Ensemble fortement stable : Ensemble indépendant S tel que pour tout sommet x pas dans S, x possède au maximum un voisin dans S.
- Ensemble dominant : Ensemble D tel que pour tout sommet x pas dans D, x possède au moins un voisin dans D.
- G(W) = (V,Ew) avec Ew les arêtes tq we <= W.

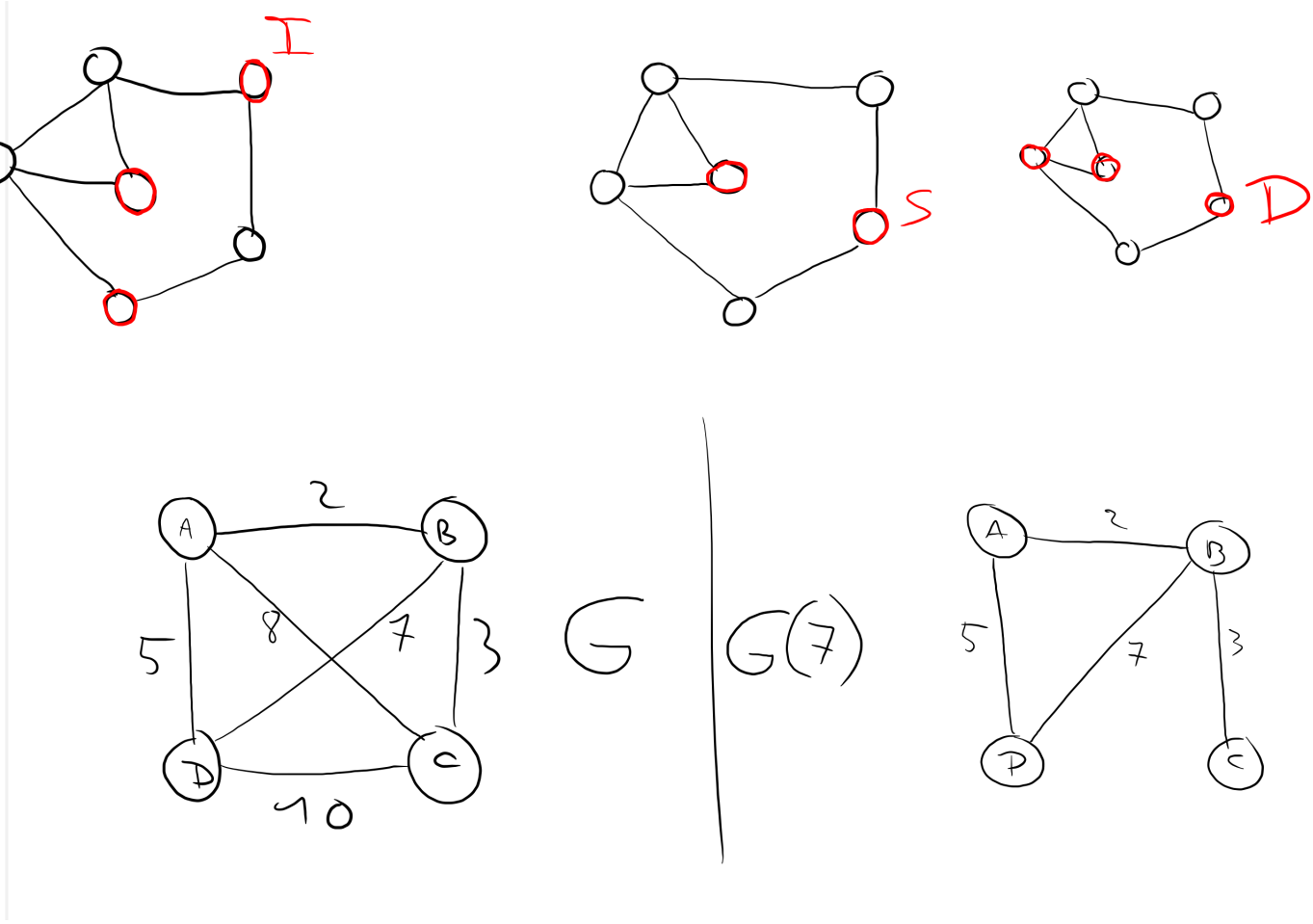

FIGURE 4 – Exemples pour les notions précédemment introduites

On peut remarquer que trouver la solution au problème des k-centres revient à trouver la valeur minimum de W tel que G(W) possède un ensemble dominant de taille inférieur ou égale à k.

On pose S^* cet ensemble dominant et w(S^*) le coût associé, c'est à dire OPT. Malheuresement trouver ce S^* est également un problème NP.

On introduit à présent la notion de carré d'un graphe telle que pour G=(V,E) on a G^2=(V,E^2) avec E^2 tous les chemins de longueur au plus 2 que l'on peut trouver dans G.

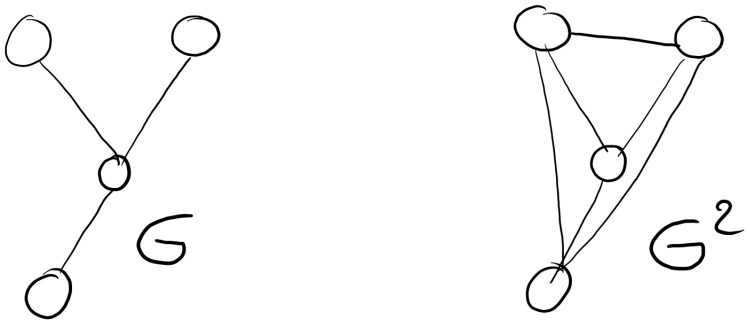

FIGURE 5 – Exemples du carre d'un graphe

On va montrer qu'il est suffisant de trouver un ensemble dominant faisable respectant certaine propriétés additionnelles.

On introduit à présent la notion de voisinage d'un sommet, noté $N_G(u)$ tel que $N_G(u)$ contient tous les points adjacents à u, y compris u.

Lemme 1 : Soit G un graphe complet. Soit G(W)^2=(V,E^2_W) le carre du graphe G(W)=(V,E_W). Si e $\in E^2_W$ alors e $\in E_{2W}$.

Preuve : Cela découle directement de l'inégalité triangulaire et que G est un graphe complet.

Note : Soit S_W un ensemble dominant d'un graphe G(W). La valeur de la solution au problème k-centre correspondant à l'ensemble de centres S_W est au plus W.

Lemme 2 : Soit $W <= w(S^*)$. Soit S tel que |S|<=k et S est un ensemble dominant dans G(W)^2. On a donc S qui est une solution faisable avec $w(S) <= 2W <= 2w(S^*)$.

Preuve : Cela découle directement du Lemme 1 et de la note ci-dessus.

Soit S^{2*} un ensemble dominant minimum pour G^2. Puisque chaque ensemble dominant pour G est dominant pour G^2 (il y a les mêmes arêtes dans G, plus d'autres), $|S^{2*}| <= |S^*|$. On veut donc trouver un ensemble dominant dans G^2 tel que : $|S^{2*}|<= |S| <= |S^*|$ **(*)**.

On va à présent se baser sur la définition d'ensemble fortement stable. On peut voir que trouver le plus petit ensemble dominant revient à trouver le plus grand ensemble fortement stable. On peut donc poser le Lemme suivant.

Lemme 3 : Soit G un graphe, FS un ensemble fortement stable et S^* l'ensemble dominant de cardinalité minimum. On a alors |FS| <= |S|.

Tout ensemble S fortement stable dans G(W) et dominant dans G(W)^2 respecte l'inégalité **(*)**. Si W<=$w(S^*)$ et |S|<=k, on a trouvé une solution faisable à notre problème.

On introduit un dernier Lemme avant la présentation de l'algorithme à proprement parler.

Lemme 4 : Soit S un ensemble fortement stable dans G. Si x n'est pas dominé pas S dans G^2, alors $S \cup \{x\}$ est fortement stable dans G.

Preuve : Procédons par l'absurde. Supposons qu'il existe un sommet v tel que tel que U=$N_G(v) \cap (S \cup \{x\})$ contient au moins 2 éléments. L'un d'eux est x car S est fortement stable dans G. Notons le deuxième élément a. On a alors un chemin de a à x en passant par v. Ainsi x est dominé par S dans G^2.

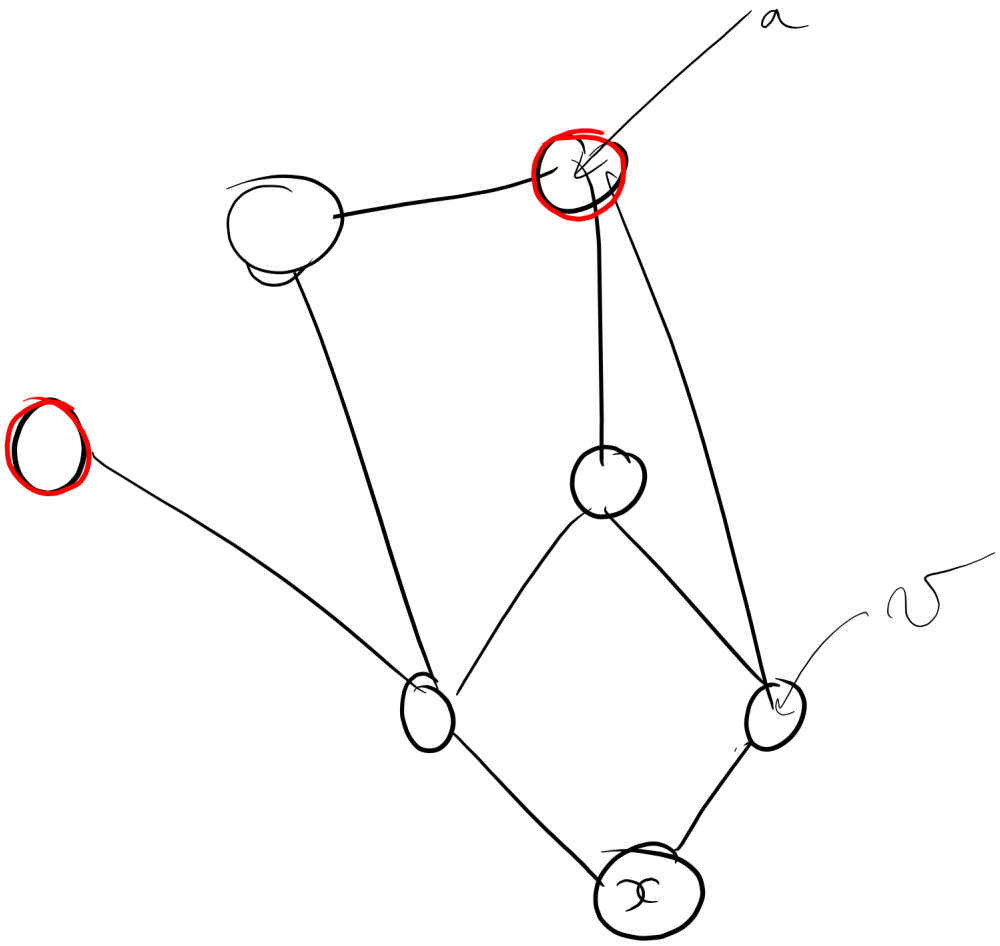

FIGURE 6 – Exemple pour la preuve du Lemme 4

On trouvera ci-dessous le pseudo code correspondant au code de l'algorithme.

Hypothèses :

— Les arêtes sont ordonnées telles que $w_{e1} <= ... <= w_{em}$

— Le graphe est stocké sous forme de liste d'adjacence.

— On pose $ADJ_w(x)$ les sommets accessibles depuis x avec un coût inférieur ou égal à w.

```
BestHeuristic(G=(V,E))
    SI k==|V|  , RETOURNER V
    low = 1
    high = m
    TANT QUE NON high == low +1 : % boucle principale
        mid = (low+high)//2
        S = {}
        T = V

        TANT QUE |T|!=0: % boucle interne
            S = S U {x}
            POUR v dans ADJmid(x) :
                T = T - ADJmid(v) - {v}
            FIN POUR
        FIN TANT QUE

        SI |S|<=k :
            high = mid
            S' = S
        SINON
            low = mid
        FIN SI
    FIN TANT QUE
    RETOURNER S'
```

Finalement, il ne reste plus qu'à montrer que l'algorithme "Best Heuristic" produit bien une solution S telle que $w(S) <= 2w(S^*)$.

Soit S l'ensemble fourni par l'algorithme à la fin d'une itération de la boucle *TANT QUE* **principale**.

On va d'abbord montrer que S est fortement stable dans G_{mid} et dominant dans G^2_{mid}.

Considérons le calcul de S pour une valeur de mid fixée. Au début de chaque passage dans la boucle *TANT QUE* **interne** S est un ensemble fortement stable et T est l'ensemble des sommets non dominés par S. Cela est vrai initialement, montrons que cela tiens jusqu'au bout de la boucle. Supposons que c'est vrai au début de la ième itération dans la boucle **interne**. D'après le Lemme 4, le nouveau S est fortement stable. Par ailleurs, les sommets dominés par x dans G^2_{mid} sont justement ceux que l'on a retiré de T. T reste donc l'ensemble des sommets non dominés. On a bien montré qu'à la fin de la boucle **interne**, S est un ensemble fortement stable dans G_{mid} et un ensemble dominant dans G^2_{mid}.

On a donc la bonne forme pour notre solution, montrons à présent que le ratio d'approximation est correct. On observe grâce au "sinon" à la fin de a boucle principale que G_{low} a un ensemble fortement stable de taille strictement plus grande que k (c'est la négation de la définition de W>=w(S^*) présentée au début. On a donc

$w_{low} < w(S^*)$. A la fin de la boucle **principale**, high = low + 1. On a donc $w_{high} <= w(S^*)$ (On rappel que $w(S^*)$ est un des wi. La solution S' renvoyée par l'algorithme est un ensemble dominant dans G^2_{high}. D'après le Lemme 2 on a donc : S' est une solution faisable au problème des k-centres avec un coût w(S') $<= 2w(S^*)$.

Finalement on a bien montré que l'algorithme renvoie une solution faisable au problème des k-centres tel que c'est une 2-approximation.

3.2 Implémentation de l'approximation Vous pourrez retrouver l'intégralité du code commenté en annexe.

Une note sur l'exécution du code. Pour effectuer des tests sur de nombreuses instances, lancer le code tel que fourni. Pour avoir pouvoir visualiser le résultat sur instance en particulier, commenter la partie TEST et dé commenter la partie AFFICHAGE. Pour plus d'informations, se rendre dans le fichier README du projet.

3.3 Analyse de l'approximation Dans cette sous-section, nous allons analyser les performances de cette approximation en moyenne et dans le pire des cas.

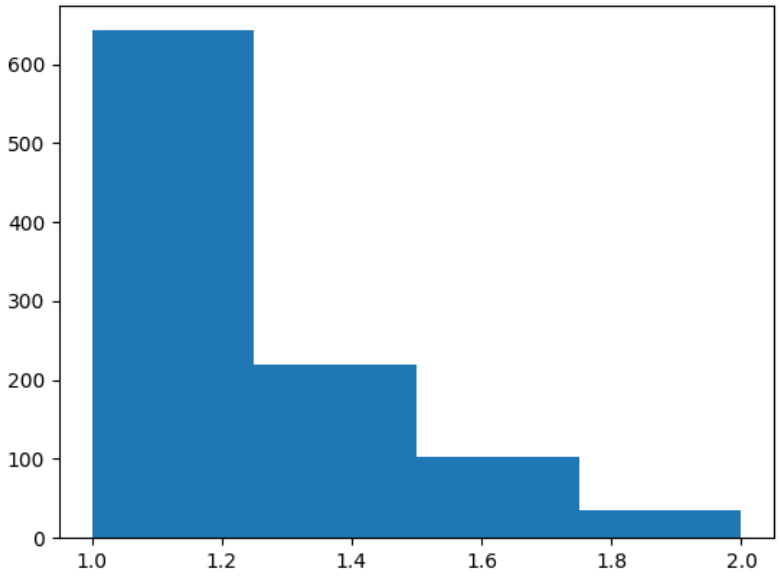

FIGURE 7 – Répartition des score d'approximation pour la 2-approximation "Best Heuristic" sur 1000 instances

Grâce à l'histogramme présenté en figure 4, on peut voir que dans plus de 60% des cas, l'algorithme retourne au maximum 1.25 fois la solution optimale. C'est nettement moins que pour l'algorithme précédent, le ratio d'approximation semble légèrement plus étalé entre 1 et 2.

Dans le diagramme en bâton présenté en figure 5, on observe qu'en moyenne, le ratio de l'algorithme "Best Heuristic" est 1.21 fois pire que l'optimal. C'est très nettement moins bon que l'algorithme précédent. Ici encore, la courbe rouge permet de confirmer en pratique que l'on a bien une 2-approximation. On a bien max_APP qui tend vers 2.

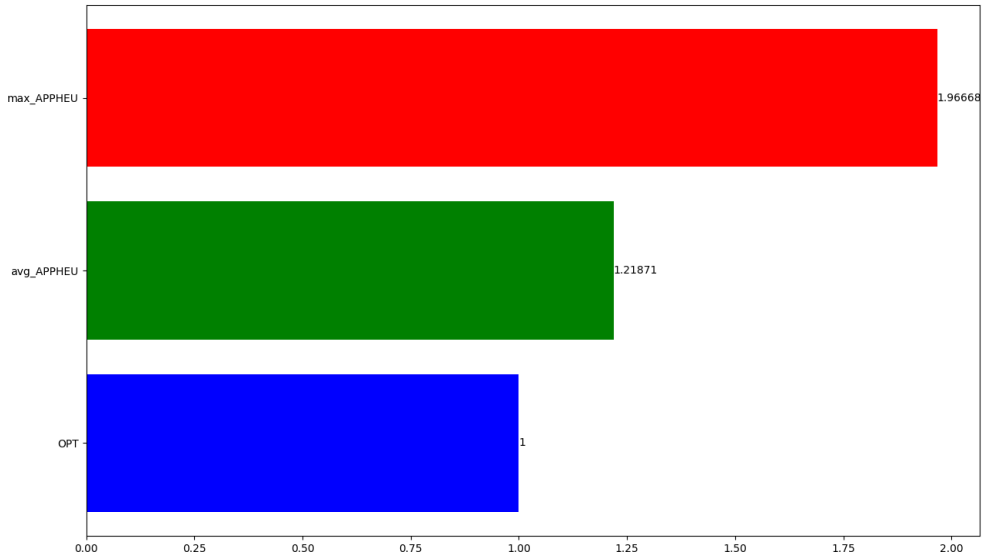

FIGURE 8 – Comparaison entre la solution optimale, le ratio en moyenne et le ratio dans le pire des cas sur 1000 instances

4 Conclusion

Finalement, on a pu voir que les 2 approches sont fonctionnelles, elles fournissent chacune une 2-approximation. Par ailleurs leurs performances en moyenne et dans le pire des cas sont très similaire. Pour réussir à les séparer il pourrait être intéressant de poursuivre cette étude en effectuant une comparaison théorique et pratique de leur complexité temporelle.

Par ailleurs, comme on peut le voir dans la figure 9 ci-contre, l'algorithme k-centre du cours renvoie parfois une solution avec moins de centres que k. Cela peut être un avantage si on sait qu'il y a au maximum k centres, mais cela pourrait être une piste d'amélioration facile à explorer si l'on sait qu'il y a exactement k centres.

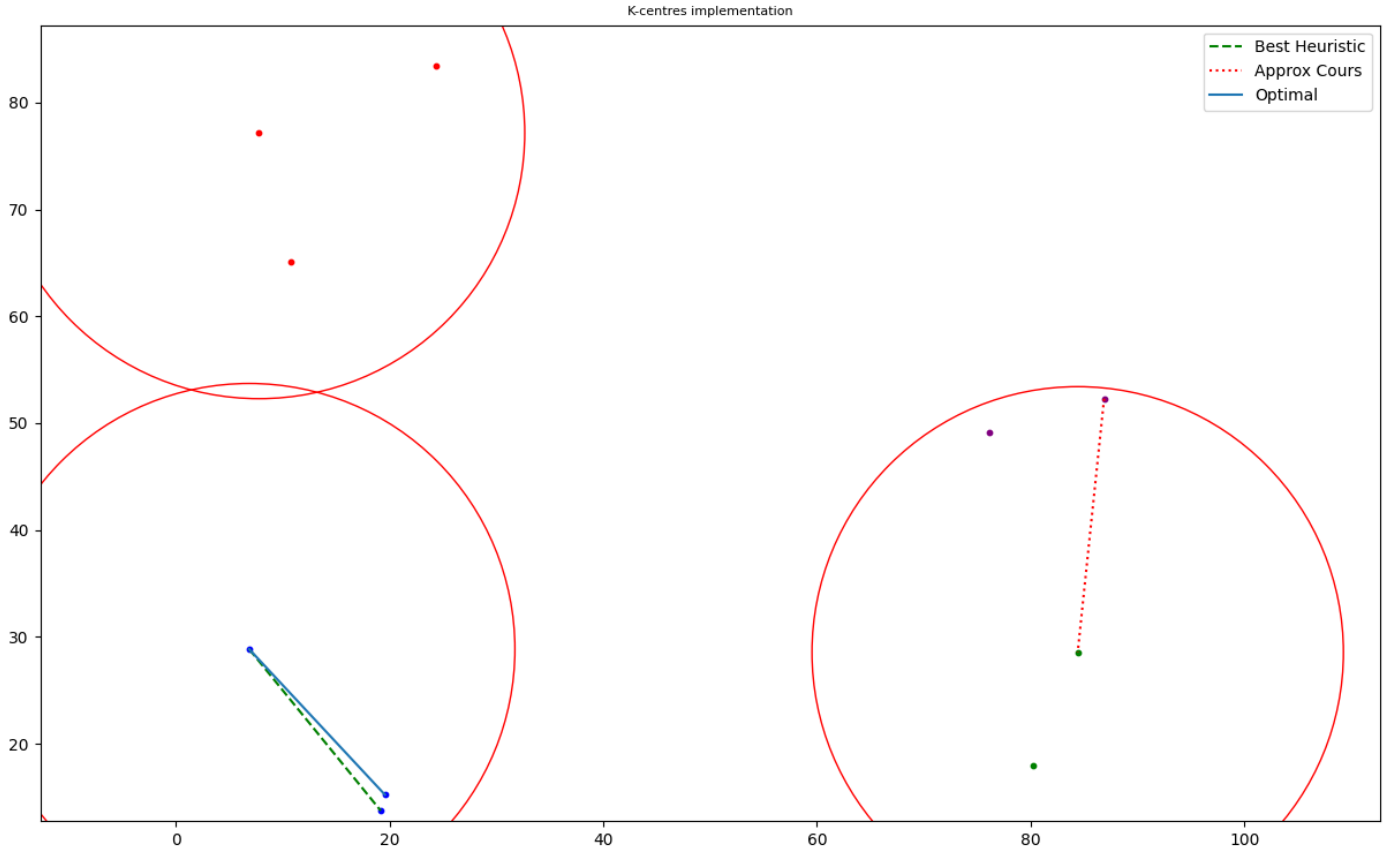

FIGURE 9 – Exemple de résultats pour une instance donnée (random state=3).

Finalement, on ne peux pas vraiment donner de réponse définitive sur quel algorithme utiliser en toute circonstance. On notera tout de même que le premier algorithme est bien plus facile à interpréter et à implémenter que le second, cela peut être un critère à considérer.

5 References

— Best possible Heuristic.