Félix La Rocque Carrier - 1621348
Mathieu Gamache - 1626377

Explication de vos implémentations:

Recherche par État:
Pour la recherche par états, nous avons surtout utilisé l'algorithme A*. L'utilisation d'un "Min heap" a beaucoup diminuer le temps de calcul de la solution. Cet structure permet d'amortir le coup du "sort" tout en gardant la solution le moins cher.

Le calcul de l'heuristique se fait en calculant le coût minimale pour couvrir X maisons, X allant de 2 au nombre de maisons totales. En prenant de coût et en le divisant pas le nombre de maison qu'il couvre nous donne un coup de couverture par maison. On prend ensuite le plus petit coup de couverture pour établir l'heuristique.

Pour rouler l'algorithme :
rouler le fichier StateSearch.py

Recherche Locale: (à utiliser pour la compétition)
Pour la recherche locale, nous avons utilisé l'algorithme de recherche en faisceau. Nous avons longtemps considérer la recherche par recuit simulé pour éviter les minimums locaux, mais le temps supplémentaire que cet recherche prend comparé à la recherche par faisceau nous a fait changer d'idée.

Nous avons aussi dus adapter la recherche car celle-ci demandait l'implémentation de la fonction IsGoal nous n'offrent pas la liberté de définir si la solution est valide. Dans notre implémentation, toutes les valeurs "Aléatoires" attribuées aux variables donnent une solution valide. L'ajout d'une liste de solution potentiel trié selon le coût de leur solution nous permet d'utiliser cet algorithme.

L'algorithme commence par attribuer des valeurs possibles à une population X. Cet attribution prend en compte les voisins les plus proches et leurs donnent une certaine probabilité d'être choisies selon cet distance et les coûts de placement versus élargissement de l'antenne.

Pour rouler l'algorithme :
rouler le fichier LocalSearch.py

Question 1 : Soit le code suivant :
def fct(predList, inputList): return filter(lambda x: all([f(x) for f in predList]), inputList)
Expliquez ce que fait cette fonction et fournissez un exemple utilisant cette fonction.

- Lamda représente une fonction anonyme, l'argument passé devient x et il applique l'instruction après le ":"

- Le all, retourne true si tous tous les éléments retourne true
- Le filter() retourne une liste de tous les éléments qui respecte la predicat

Tout cela ensemble, pour un ensemble de prédicats et de données, il va retourner les données qui respecte tous les prédicats (retourne True pour chaque prédicats).

Question 2 : Quelles sont les points forts et les faiblesses de vos implémentations ?

Recherche par État:

+ Faire du pré-calcule aide beaucoup à la performance.
- La prise de décision demande beaucoup de calcul de trouver l'antenne pour couvrir une certains ensemble de maison: qui sont des calculs pas trivial.
- L'heuristique bien qu'admissible pourrait être plus précise.

Recherche Locale:

+ Faire du pré-calcule aide beaucoup à la performance.
+ Rapide lorsqu'il existe un nombre très grand d'états possibles.
+ On peut retoucher plusieurs paramètres selon le cas de la recherche pour optimiser celle-ci en temps ou en précision.
+ Notre fonction randomize au début fonctionne très bien et est sensible au valeur de k et c.
- Beaucoup de deepcopy et le temps de ceux-ci sont la majorité du temps allouée.
- La stratégie employé est très de base et n'a pas été très optimisée.