INF4215 – Rapport de laboratoire

Recherche dans un espace d'états

Julien Antoine
1813026

Abdelhadi Temmar
1805815

14 février 2015

1 Recherche arborescente

1.1 Représentation du problème

1.1.1 Représentation d'un état

Un état est composé de :

— une liste des coordonnées des points restant à couvrir

— une liste des antennes placées (position et rayon)

— le coût requis par les antennes déjà placées

1.1.2 Etat initial

L'état initial est donc composé de l'ensemble des points à couvrir, d'un ensemble vide (représentant les antennes placées), et d'un coût de 0.

1.1.3 Actions possibles

Une seule action est possible à partir d'un état : l'ajout d'une nouvelle antenne. Cependant, avant de chercher les antennes que l'on peut possiblement ajouter, nous vérifions que le nombre d'antennes de l'état ne dépasse pas le nombre de points initiaux et que le coût de l'ajout d'une nouvelle antenne additionné à celui des antennes déjà placées ne dépasse pas le coût maximal. En effet, nous ne voulons pas que notre programme retourne une solution avec un coût supérieur à celui qu'on obtiendrait si on ne plaçait qu'une seule antenne couvrant tous les points.

Nous recherchons ensuite la liste des antennes que l'on peut ajouter, toujours en vérifiant que cet ajout n'engendre pas une solution avec un coût supérieur au coût maximal. Si aucun point ne se trouve sur le périmètre maximal de l'antenne, nous décidons de ne pas la considérer. Nous ne voulons pas de rayon trop grand inutilement.

1.1.4 État final

Un état est final si l'ensemble de points qu'il reste à couvrir est vide.

1.2 Type de recherche choisi

Après plusieurs tests, l'algorithme de recherche en profondeur s'est avéré le plus efficace pour notre recherche arborescente. Il apporte une solution en environ une minute pour une grille de 50×50 sans qu'on ait à faire aucune vérification particulière sur un nœud après ajout d'une antenne.

L'algorithme de recherche au moindre de coût ne retourne toujours pas de résultat en 1h de recherche pour une taille de grille de 50×50. Nous avons donc essayé d'utiliser l'algorithme A* et de trouver une heuristique capable d'améliorer la recherche de moindre coût. Malheureusement aucune de nos heuristiques n'a donné de résultat concluant.

2 Recherche locale

Un deuxième type de recherche est la recherche locale. Contrairement à la recherche arborescente vue dans la section précédente, la locale part d'un état et essaie de l'améliorer sur base de satisfaction de contraintes.

2.1 Représentation du problème

2.1.1 Représentation d'un état

Un état est simplement composé d'une liste des antennes placées (position, rayon et coordonnées des points couverts), par exemple :

```
[(20,15,7,[(22,19),(14,15),(17,18)])]
```

signifie qu'une seule antenne est positionnée en $(20, 15)$ avec un rayon de taille 7, et qu'elle couvre les points $(22, 19)$, $(14, 15)$ et $(17, 18)$.

2.1.2 Etat initial

L'état initial est celui où une antenne de rayon 1 est placée sur chaque point à couvrir.

2.1.3 Actions possibles

Les actions possibles à partir de chaque état est une fusion entre deux antennes. Cette action est faisable uniquement si le coût de l'antenne résultante de la fusion de deux antennes est inférieur au coût total des deux antennes. Lors de la fusion, l'antenne résultante se situe au barycentre des points couverts par chacune des deux antennes fusionnées, et a pour rayon la distance entre ce barycentre et le point couvert par l'antenne le plus éloigné.

2.1.4 Etat final

L'état final est atteint lorsque plus aucune fusion n'est possible.

2.2 Résultats

Cette méthode est considérablement plus rapide que la recherche arborescente, puisque le nombre d'opérations effectuées est bien moindre. Sur une grille de 50×50, on obtient une solution proche de l'optimale (voir figure 1) en moins d'une milliseconde.

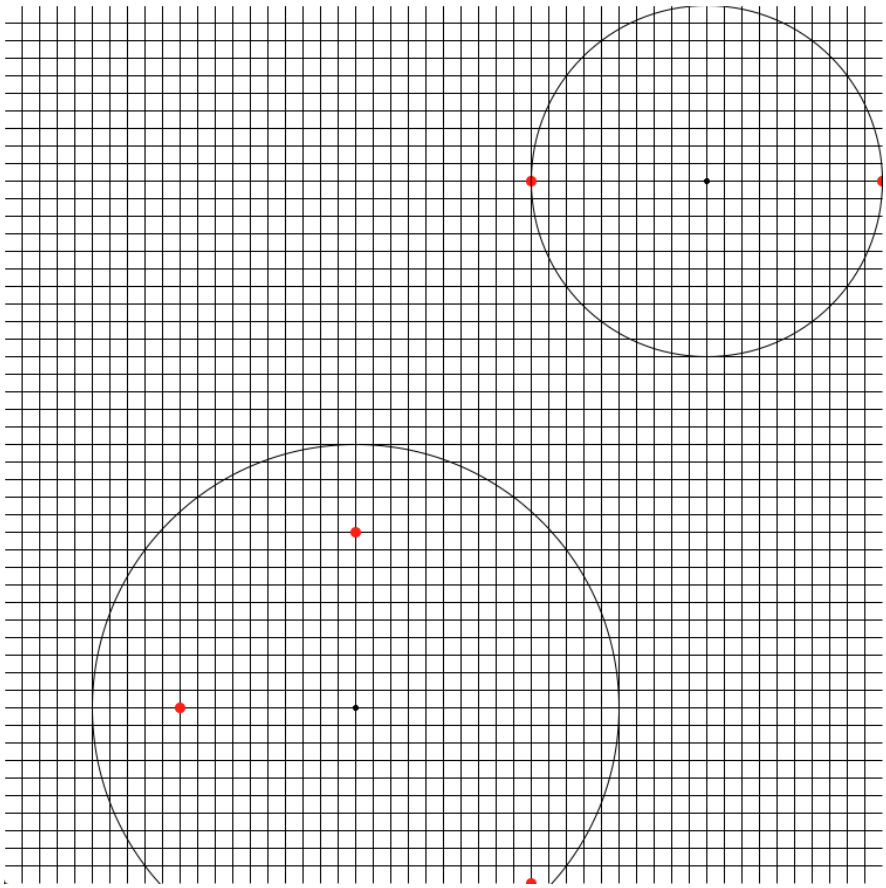

FIGURE 1 – Résultat de la recherche locale pour l'exemple de l'énoncé

3 Réponses aux questions

a) La fonction `fct` renvoie uniquement les éléments de `inputList` qui vérifient les prédicats présents dans `predList`. Par exemple,

```
fct([lambda: x < 8, lambda: x > 3], [1,2,3,4,5,6,7,8])
```

retourne les éléments compris strictement entre 3 et 8, donc [4,5,6,7].

b) Le point fort de notre recherche locale est clairement sa rapidité d'exécution. Quelle que soit la taille de la grille ou le nombre de points, elle retourne toujours un résultat endéans la seconde. Le résultat retourné est en général assez proche de l'optimal, on n'obtient jamais de résultat absurde. Sa faiblesse est qu'elle ne retourne jamais la solution optimale, sauf dans certains cas simples comprenant une symétrie par exemple.

Notre recherche arborescente a comme point fort de garantir de trouver la solution optimale. Cependant, ce point est largement contre balancé par le fait que pour des tailles de grilles plus conséquentes, elle ne produit pas de résultat avant au moins 45 minutes. Cela est notamment dû à des lacunes dans l'heuristique, que nous n'avons malheureusement pas réussi à améliorer comme nous l'aurions désiré.