École Polytechnique de Montréal

INF4215
Introduction à l'intelligence artificielle

Travail pratique #1

Soumis par :
Chali-Anne Lauzon 1636918
Marie-France Miousse 1638639

15 février 2015

Question 1

Expliquez ce que fait le code suivant :

```
def fct(f, g, p, l):
    return map(lambda x: f(g(x)),
               [x for x in l if p(x)])
```

Réponse :

La fonction map est une fonction qui prend deux arguments : une fonction et une liste d'objets sur lesquels appliquer cette fonction. Lambda est utilisé pour créer de petites fonctions vagabondes.

Le code ici retourne une liste dont chaque élément qui retourne vrai à une fonction p(x) a subi l'opération f rond g.

Question 2

Quels sont les points forts et les faiblesses de vos implémentations ?

Réponse :

Dans le cas de la recherche arborescente A*, l'avantage est que, peu importe la taille du problème, si une solution existe, l'algorithme permettra de trouver celle dont le coût est minimal. Toutefois, sauvegarder tous les états intermédiaires peut provoquer un débordement au niveau de la mémoire disponible, même si les états ne prennent pas beaucoup de place individuellement.

La recherche locale par escalade permet prendre très peu d'espace mémoire, parce qu'il est impossible de remonter dans l'arbre. Un seul nœud est traité à la fois, et il est remplacé par un de ses enfants qui minimise le plus l'heuristique. Bien que la résolution soit beaucoup plus rapide, il est possible de tomber dans un minimum local. Ainsi, il n'est pas toujours possible de trouver la solution même si elle existe.

Explications de l'implémentation

La base des deux algorithmes est identique. Nous avons considéré que, puisque le problème est le même, la définition d'une pièce de tangram ainsi que d'un puzzle de tangram ne changerait pas selon l'implémentation.

Une pièce de tangram est un tableau en deux dimensions qui possède une liste d'autres pièces représentant ses orientations possibles. Un puzzle de tangram, qui sert à la résolution du problème, est un tableau en deux dimensions tel que passé en paramètre.

Pour déterminer les pièces à placer, on commence par trier celles qui nous sont données en entrée par ordre inverse de nombre de cases occupées. On place une seule pièce à la fois : la plus grosse. Si cette pièce ne peut pas être placée, aucune solution n'existe. Cette implémentation nous garantit que chaque niveau de l'arbre de recherche correspond à une pièce, plutôt qu'à toutes les

positions possibles de toutes les pièces disponibles à ce moment. Le coût est inversement proportionnel à la taille de la pièce. Plus la pièce prend d'espace, moins le coût sera élevé. L'heuristique correspond à la différence entre le nombre de pièces à placer et le nombre de pièces restantes.

Le seul changement apporté aux implémentations fournies par le professeur est la liste de nœuds pour la recherche locale par escalade. En effet, pour repartir d'un état aléatoire, il fallait avoir un état. La liste permet de repartir d'un des états passés pour sortir d'un minimum local. Malheureusement, aucune solution n'a encore été trouvée à l'aide de cette implémentation. Cela nous amène à penser que la définition du problème devrait être modifiée, mais nous n'avons toujours pas trouvé de contraintes qui pourraient restreindre la recherche pour mener à une solution. Nous savons qu'il y a un problème dans cette implémentation, et que nous n'aurions pas dû recourir à une liste de nœuds. Cela brise en effet le choix de l'implémentation, qui ne doit traiter qu'un et un seul nœud afin d'économiser de l'espace mémoire.

Pour la compétition, l'implémentation par recherche arborescente sera utilisée.