Sujet de TP n°1 du module « Méthodes de résolution des problèmes »

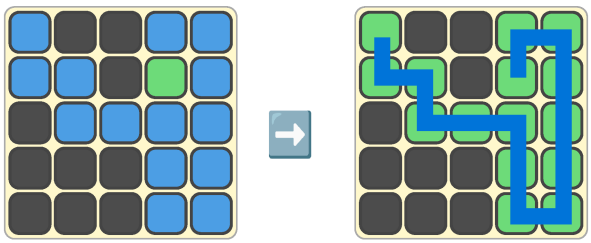
Année universitaires : 2024/2025

1. Introduction

On considère un jeu utilisant un plateau (en deux dimensions) composé de cases bleues, vertes et noires. Les cases vertes et noires ne peuvent pas être manipulées. Lorsqu'on touche une case bleue, elle devient verte. Au début du jeu, une seule case est verte. Le but est de transformer toutes les cases bleues en vert en respectant les deux règles suivantes :

1. On doit commencer à partir d'une case verte.
2. La prochaine case à transformer doit être adjacente (gauche, droite, haut, bas) à la case qui vient d'être transformée. En d'autres termes, si la transformation se fait en mettant un stylo sur la première case verte, on devrait transformer toutes les cases sans lever le stylo.

La figure suivante montre un plateau initial et le plateau objectif :



En réalité, la résolution de ce problème équivaut à trouver un chemin hamiltonien dans un graphe. Ce problème est connu dans la littérature pour être NP-complet. L'objectif de ce TP est d'étudier comment résoudre ce problème en utilisant les heuristiques tout en les comparant aux méthodes de recherche aveugle. L'exemple donné ici est illustratif, le but du TP est de résoudre le problème pour un plateau de taille 12×12.

On modélise le plateau de jeu par une matrice de valeurs $\{0, 1, 2\}$:

- 0 : la case est noire.
- 1 : la case est bleue.
- 2 : la case est verte.

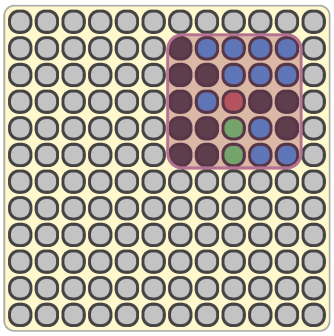
Ainsi, le problème est considéré comme résolu s'il ne reste aucun 1 dans la matrice (voir le code Python fourni).

Pour résoudre le problème, il est demandé d'implémenter les méthodes de recherche suivantes :

- Recherche en profondeur.
- Recherche en largeur.
- Recherche aléatoire.
- Recherche A avec $f = a.p_5 + b.p_3 - \lfloor \frac{d}{c} \rfloor$, avec :
  - $a \in \{0, 1\}$, $b \in \{0, 1\}$ et $c \in \{10, 50\}$ sont des constantes (il faut essayer plusieurs combinaisons).
  - p_k correspond à des pénalités liées à la résolution d'une version simplifiée du problème avec une matrice $k \times k$, comme expliqué plus bas.
  - d : est la profondeur de la recherche.

2. Calcul de p_k

Supposons qu'à un moment donné, la recherche arrive à point du plateau et doit décider s'il est intéressant de l'explorer ou non. Pour décider, on résout une version simplifiée du problème initial en considérant un plateau $k \times k$. Considérons la figure suivante (pour $k = 5$) :



Il s'agit ici de mesurer l'intérêt du point rouge au centre du grand carré marron. Dans ce cas, on réduit le problème aux cases du carré marron et on essaie de couvrir le maximum de cases bleues à partir du points rouge (dans ce cas 7). Il reste alors 4 cases non-couvertes. Dans ce cas, $p_5 = 4$. Le code de cette fonction (implémentée par une recherche par retour arrière) est fourni dans le code du mini-projet. Vous devriez, cependant, écrire le code qui extrait la partie de la matrice de taille $k \times k$. Notez que pour des raisons de performances, nous utiliserons uniquement les valeurs $k = 3$ ou $k = 5$.

3. Travail demandé

- Télécharger et comprendre le code Python fourni.
- Le travail présenté doit **impérativement** étendre le code fourni (tout autre code ne sera pas accepté).
- Une fois la solution trouvée, vous devrez afficher le chemin trouvé.
- Tester les codes écrits avec le benchmark fourni. Comparez les performances des différentes méthodes.
- Le travail se fait par monôme, binôme ou trinôme.
- Le TP doit être hébergé sur Github.
- Afin d'améliorer les performances de recherche, utilisez l'interpréteur `pypy3` au lieu de `python3`.