Aix-Marseille Université

Licence Informatique – Intelligence Artificielle

Projet

Dans ce projet, nous allons nous intéresser au jeu du taquin.

1 Présentation

Une instance du jeu du taquin se définit par la donnée d'une grille initiale de $n \times m$ cases et d'une grille finale de mêmes dimensions. Chaque grille est composée de $n \times m$-1 blocs et d'une case vide. L'idée est de passer de la grille initiale à la grille finale en déplaçant horizontalement ou verticalement un bloc vers la case vide. Ce bloc doit bien sûr être adjacent à la case vide.

1	2	3
7	4	5
8		6

(a)

1	2	3
4	5	6
7	8	

(b)

Les objectifs de ce projet sont d'utiliser les techniques d'IA (ici des graphes d'états) pour déterminer s'il existe un chemin permettant d'aller de l'état initial à l'état final et le cas échéant de produire un tel chemin.

2 Modélisation d'une instance

Nous allons modéliser un état sous la forme d'une grille de $n \times m$ cases. Chaque case contiendra un bloc ou sera vide. Un état correspondra donc exactement à une configuration d'une grille de taquin. L'état initial (respectivement final) est celui de la grille initiale (resp. finale). On dispose de quatre règles de production à savoir :

- déplacer un bloc d'une case vers la droite, si la case immédiatement à sa droite existe et est vide,
- déplacer un bloc d'une case vers la gauche, si la case immédiatement à sa gauche existe et est vide,
- déplacer un bloc d'une case vers le bas, si la case immédiatement au-dessous existe et est vide,
- déplacer un bloc d'une case vers le haut, si la case immédiatement au-dessus existe et est vide.

3 Travail à réaliser

(i) Implémenter les algorithmes de recherche suivant :
- Parcours en profondeur d'abord,
- Parcours en largeur d'abord,
- Parcours le meilleur d'abord.

(ii) Proposer et implémenter des heuristiques permettant de classer les états par ordre de préférence.

(iii) Comparer l'efficacité des différents parcours et heuristiques sur les jeux de données fournis (une archive contenant quelques instances est disponible sur la page AMeTICE de l'UE). Vous pouvez bien sûr créer en plus vos propres instances.

4 Format de fichier

Chaque fichier contient une seule instance. Pour simplifier la manipulation des fichiers et des blocs, on considère ici que chaque bloc est représenté par un caractère et que la case vide est matérialisé par un

caractère espace. On supposera également qu'il n'y a qu'un seul caractère espace par grille. La syntaxe de ces fichiers est la suivante :

— La première ligne contient le nombre n de lignes d'une grille de jeu.
— Les n lignes suivantes contiennent la grille initiale à raison d'une ligne de la grille par ligne du fichier.
— De même, les n dernières lignes contiennent la grille finale.

5 Rendu du travail

Vous devrez fournir le code source de votre implémentation ainsi qu'un rapport. Ce rapport devra contenir :

1. Un manuel d'utilisation de votre programme.
2. Les explications concernant votre implémentation des algorithmes de parcours.
3. Une description des heuristiques que vous proposez.
4. Une analyse de la comparaison des différents parcours et heuristiques accompagnée de tableaux et/ou courbes.

Le rendu se fera via un espace dédié sur la page AMeTICE de l'UE.

6 Quelques conseils

6.1 Structures de données et langage

Il est fortement conseillé de prendre le temps de la réflexion avant d'arrêter un choix pour vos structures de données et pour le langage dans lequel vous allez les implémenter. Vous pouvez implémenter séparément chacun des parcours. Toutefois, si on considère l'algorithme général vu en cours, il serait préférable de n'implémenter qu'une fois cet algorithme et de jouer sur la nature de la structure *Ouvert* pour obtenir chacun des trois parcours demandés. Dans un tel cas, une approche orientée objets semble préférable. Pour des raisons d'efficacité chronométrique, un langage comme C++ semble plus pertinent que Java ou Python par exemple.

6.2 Débogage et évaluation expérimentale

Si vous développez en C ou C++, durant la phase de débogage, je vous invite à compiler en utilisant les options `-Wall` et `-g3` du compilateur `gcc` ou `g++` et d'utiliser alternativement un débogueur comme `gdb` et un programme vérifiant tous les accès à la mémoire comme `valgrind`. Une fois votre programme au point, il est conseillé de le compiler avec l'option d'optimisation `-O3` du compilateur afin d'obtenir un exécutable le plus performant possible.

Quel que soit le langage, dans le cas de figure où vos expérimentations sont effectuées sur un ordinateur portable, afin de ne pas biaiser les comparaisons, il est important de s'assurer que la politique de gestion de l'énergie est réglée sur la puissance maximum et que l'ordinateur est bien branché à une prise électrique. Quel que soit le type d'ordinateur utilisé, il est également recommandé de ne pas exécuter d'autres programmes durant l'expérimentation, ni d'exécuter plusieurs parcours en même temps. En effet, même si votre ordinateur dispose de plusieurs cœurs, cela a tendance à fausser les observations.