Casamian Marius (GRP IA) / cm205773

Introduction à l'Intelligence Artificielle
Licence 2 – Portail Sciences et Technologies

Travaux dirigés N°1 : Résolution de problèmes

Objectifs du TP :

1. Comprendre l'algorithme A* pour la résolution du jeu du taquin (3 x 3).
2. Utilisation du langage Python
3. Comprendre et comparer des méthodes heuristiques
4. Rédiger le compte rendu d'observations et d'expériences du TP.

Résolution du TP :

Après avoir téléchargé l'archive sur le GitHub du créateur.
Nous obtenons plusieurs documents :

- Le script Python principal (8puzzle.py) permet de résoudre le taquin en utilisant deux heuristiques (**Distance Manhattan** et **Numbers of misplaced tiles**) et qui imprime dans la console l'état initial, intermédiaire et final.
- A retenir que **Distance Manhattan** est basée sur la somme des distances horizontales et verticales entre la position actuelle de chaque tuile et sa position cible dans l'état final. **NMT*** compte le nombre de tuiles qui sont mal placées par rapport à l'état final.
- Deux fichiers texte contenant des exemples de puzzle. La première ligne contient l'état initial et la deuxième l'état final, chacune incluant des nombres entre 0-8.
- Un fichier README.md contenant toutes les explications et instructions pour savoir comment exécuter le script, quelles méthodes heuristiques utilisées ainsi que des exemples.
- Un fichier LICENCE

Après la lecture du README.md on exécute le script 8puzzle.py sur les deux instances du problème.

Exécution :

- python 8puzzle.py 1 1 < puzzle02.txt

Cette ligne permet d'exécuter le fichier 8puzzle.py en utilisant l'argument heuristique de la méthode Manhattan (1) et l'argument de sortie 1 pour les états tout en redirigeant grâce à la flèche la sortie standard dans le fichier puzzle02.txt.

Résultat :

```
1 2 0 3 4 5 6 7 8
1 0 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
```

On peut voir que le résultat de cette exécution est bien le résultat attendu, puisque on part de l'état initial du fichier puzzle02.txt et on obtient bien l'état final contenu dans ce dernier. Par ailleurs on obtient le même résultat pour la deuxième heuristique (Tuile mal placés) en exécutant sur le terminal la ligne suivante : python 8puzzle.py 2 1 < puzzle02.txt. On en déduit alors que le temps de résolution du jeu du taquin dépends de la méthode heuristique utilisée et de la différence entre l'état initial et l'état final, par exemple pour le fichier puzzle20.txt le temps de résolution est plus long que pour le fichier puzzle02.txt dû à un état initial bien plus différent de l'état final.

Dans les consignes 4 et 5 du TP, on nous demande de comparer les deux heuristiques en générant des états initiaux au hasard et en chronométrant l'exécution de A*. Pour ce faire on créer un script python (contenu dans le fichier zippé) nommé 8puzzle_v2.py. On commence par importer les modules nécessaires à savoir random pour générer les états, time pour le chronomètre et enfin importlib pour l'utilisation du module 8puzzle dans un autre fichier.

De plus, on sait que pour qu'un état soit résoluble, sa parité doit être la même que celle de l'objectif cela signifie que la parité d'un état représente le nombre total d'inversions dans cet état. Si le nombre total d'inversions et pair on dit d'un état de parité pair et inversement. Ainsi on code une fonction **is_solvable()** qui prend en paramètre un puzzle et qui vérifie trois choses :

- La liste contient 9 chiffres
- Chaque élément est un chiffre
- Que le nombre total d'inversion est pair

Ensuite, grâce à la fonction **generate_solvable_puzzle()** on s'assure de générer un puzzle qui est solvable à coup sûr.

```
initial = puzzle.EightPuzzle(generate_solvable_puzzle())
goal = puzzle.EightPuzzle('1 2 3 4 5 6 7 8 0')
h1 = puzzle.EightPuzzle.tile_switches_remaining
h2 = puzzle.EightPuzzle.manhatten_distance
output = puzzle.EightPuzzle.state_transition

print(initial.a_star(goal, h2, output))
start_time = time.time()
print(initial.a_star(goal, h2, output))
temps_exec = time.time() - start_time
print("Temps execution: ",temps_exec,"s")
```

Enfin on génère l'état initial grâce à cette fonction dans le but d'obtenir l'état final '1 2 3 4 5 6 7 8 0'.

```
C:\Users\Marius\Desktop\TP_IA\TP_1_TAQUIN\8puzzle>python3 8puzzle_v2.py
5 0 2 6 7 1 8 4 3
5 7 2 6 0 1 8 4 3
5 7 2 0 6 1 8 4 3
5 7 2 8 6 1 0 4 3
5 7 2 8 6 1 4 0 3
5 7 2 8 0 1 4 6 3
5 7 2 8 1 0 4 6 3
5 7 2 8 1 3 4 6 0
5 7 2 8 1 3 4 0 6
5 7 2 8 1 3 0 4 6
5 7 2 0 1 3 8 4 6
5 7 2 1 0 3 8 4 6
5 0 2 1 7 3 8 4 6
0 5 2 1 7 3 8 4 6
1 5 2 0 7 3 8 4 6
1 5 2 7 0 3 8 4 6
1 5 2 7 4 3 8 0 6
1 5 2 7 4 3 0 8 6
1 5 2 0 4 3 7 8 6
1 5 2 4 0 3 7 8 6
1 0 2 4 5 3 7 8 6
1 2 0 4 5 3 7 8 6
1 2 3 4 5 0 7 8 6
1 2 3 4 5 6 7 8 0
5 0 2 6 7 1 8 4 3
5 7 2 6 0 1 8 4 3
5 7 2 0 6 1 8 4 3
5 7 2 8 6 1 0 4 3
5 7 2 8 6 1 4 0 3
5 7 2 8 0 1 4 6 3
5 7 2 8 1 0 4 6 3
5 7 2 8 1 3 4 6 0
5 7 2 8 1 3 4 0 6
5 7 2 8 1 3 0 4 6
5 7 2 0 1 3 8 4 6
5 7 2 1 0 3 8 4 6
5 0 2 1 7 3 8 4 6
0 5 2 1 7 3 8 4 6
1 5 2 0 7 3 8 4 6
1 5 2 7 0 3 8 4 6
1 5 2 7 4 3 8 0 6
1 5 2 7 4 3 0 8 6
1 5 2 0 4 3 7 8 6
1 5 2 4 0 3 7 8 6
1 0 2 4 5 3 7 8 6
1 2 0 4 5 3 7 8 6
1 2 3 4 5 0 7 8 6
1 2 3 4 5 6 7 8 0
Temps execution:  3.7841289043426514 s
```

- Le chronomètre indique 3.8s pour l'heuristique Manhattan distance en passant de la configuration '5 0 2 6 7 1 8 4 3' à '1 2 3 4 5 6 7 8 0'.

- Le chronomètre indique 2m59s pour l'heuristique Numbers of misplaced tiles en passant de la configuration '5 0 2 6 7 1 8 4 3' à '1 2 3 4 5 6 7 8 0'.

Conclusion du TP :

Pour conclure nous avons vu dans ce TP comment résoudre le jeu du taquin grâce à l'algorithme A* et des méthodes heuristiques.
Après plusieurs assertions et observations on peut déduire que l'heuristique Manhattan distance correspond le mieux pour la résolution du problème, contrairement à l'heuristique NMT* qui mets plus de temps pour le même état initial. Cependant cela peut dépendre des instances utilisées mais en règle générale Manhattan distance est plus performante.

Note : NMT (Nombre de tuiles mal placées)
Le code source du TP est disponible dans le dossier zippé.