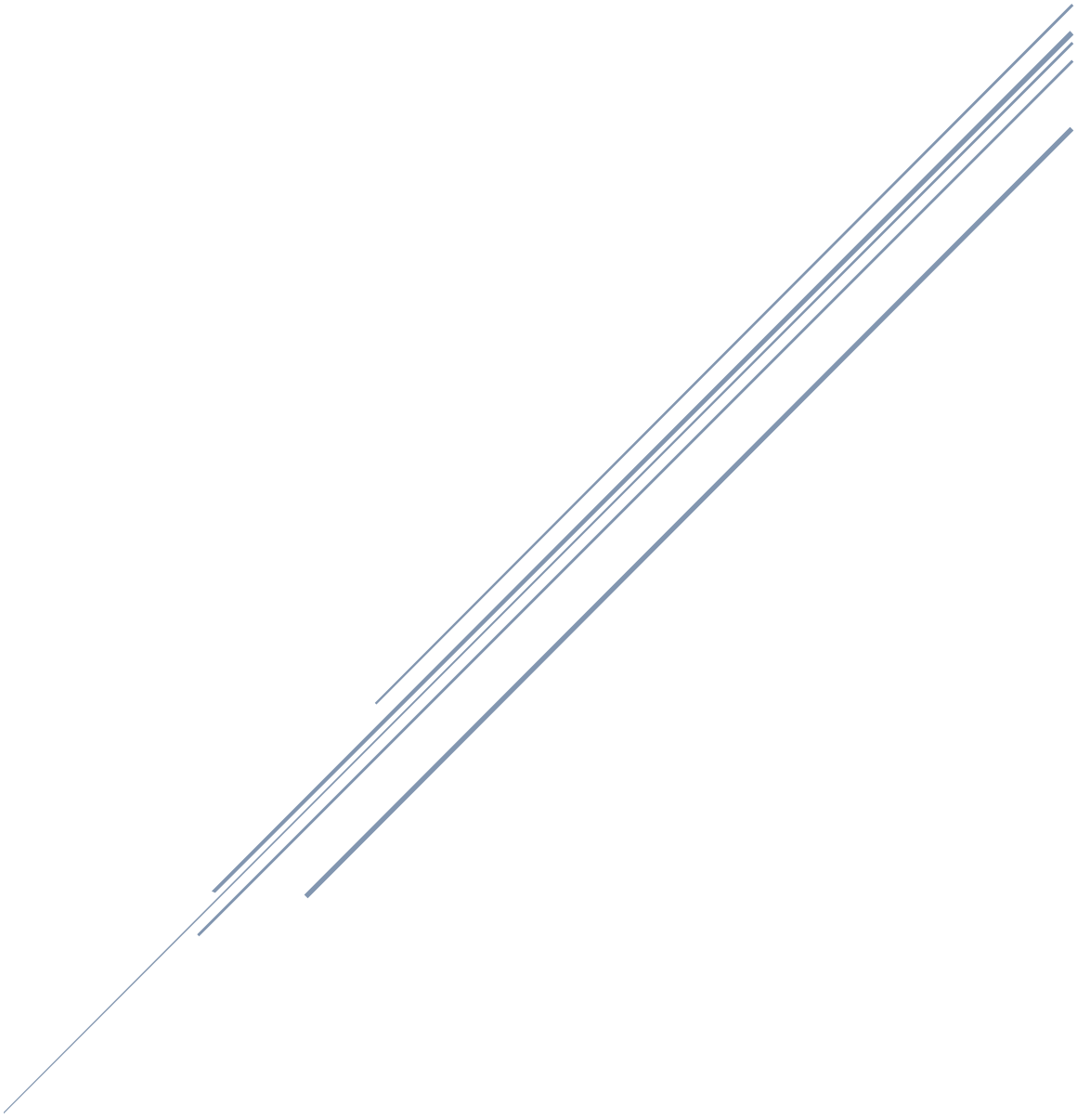
Projet 1 d'intelligence artificielle

Recherche dans des graphes d'états



BEAUPLET Gabriel | SAMRI Achraf | Elèves-ingénieurs 2016-2017

Sommaire

Introduction

Le jeu du Taquin a été inventé vers 1870 aux Etats-Unis par Sam Lloyd. Le Taquin est décliné en divers tailles, le jeu le plus répandu est un plateau de 4 par 4 (“15 Puzzle” en anglais). Le Taquin se présente sous la forme d’un plateau dans lequel se trouvent n carreaux coulissants ainsi qu’une case vide. Chaque carreau est numéroté de 0 à n*n-1. Le but est de remettre en ordre les carreaux en les faisant glisser. Dans le cadre de notre projet, nous utiliserons des Taquins de taille 3X3 comme dans l’exemple ci-dessous.

“724506831”
Etat initial

“012345678”
Etat final

Le Projet

Le jeu du taquin est un exemple classique pour enseigner A*. A partir d'un état initial quelconque, le but est de trouver la solution en un minimum de coups si elle existe.

Afin de trouver la solution au taquin, nous utiliserons deux sortes d'algorithmes : informé et non informé.

Pour la première catégorie, nous utiliserons l'algorithme A*. C'est un algorithme de recherche de chemin dans un graphe entre un état initial et un état final, tous deux connus. Il utilise une "heuristique", c'est en quelque sorte une évaluation de la distance restante jusqu'à l'état final. Nous utiliserons deux heuristiques : la première, que l'on nommera h1 compte le nombre de cases mal placées tandis que la seconde, notée h2 est la somme de la distance de Manhattan entre l'emplacement courant de chaque case à sa position finale.

Quant aux algorithmes non informés, nous utiliserons les algorithmes de recherches en profondeur et en largeur (DFS et BFS).

La difficulté

La difficulté pour le taquin est le nombre de combinaisons possibles. Lorsque le nombre de combinaisons possibles est faible, nous pouvons utiliser une recherche gloutonne. Cependant, avec le taquin 3X3, il existe 9!=362880 possibilités. C'est un chiffre encore acceptable comparé aux 16!=20 922 789 888 000 combinaisons possible du Taquin 4X4.

La seconde difficulté est d'avoir un état initial solvable. En effet, seulement la moitié des états du taquin 4X4 sont solvables. Néanmoins, dans la suite du rapport, nous ne parlerons pas des états non solvables car tous les états initiaux fournis sont solvables.

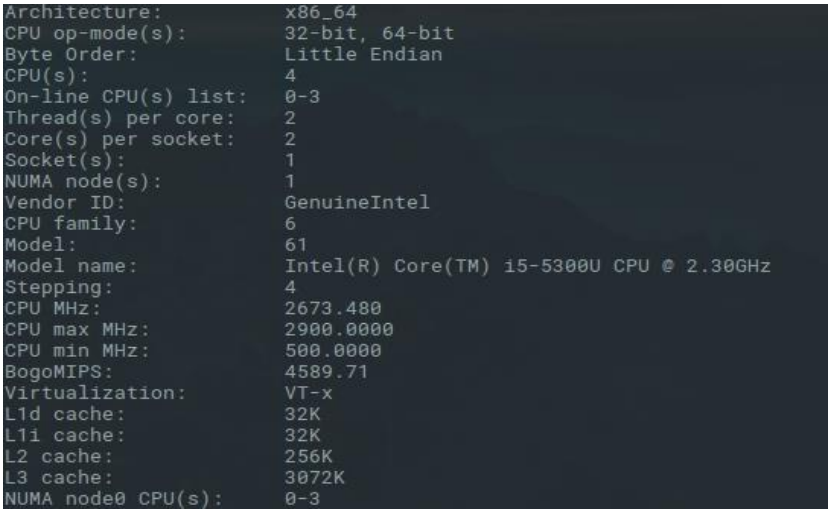
Notre matériel

Nous avons complété les codes fournis pour les 3 algorithmes et avons résumé dans le tableau de la partie « Résultats » les moyennes des temps de résolution pour chacun des algorithmes.

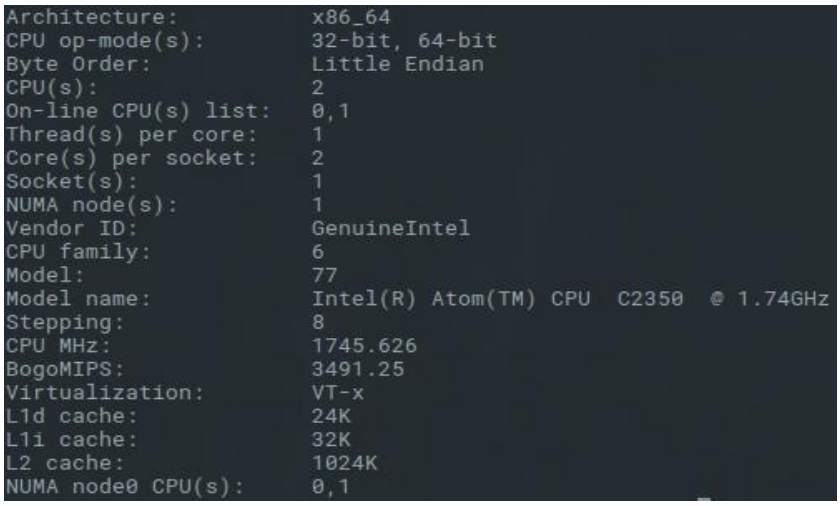
Par ailleurs, nous avons remarqué que la puissance de la machine qui exécutait l'algorithme était déterminante. Pour le prouver, nous avons exécuté l'algorithme A* avec l'heuristique de Manhattan sur le serveur (qui se trouve dans les locaux de 'Online.net') et sur un PC personnel.

Nous communiquons avec le serveur via SSH .Voici les spécifications techniques des deux appareils :

Le PC :



Le serveur :



Il apparait clairement que le serveur est beaucoup moins puissant que le PC. Les performances seront-elles impactées ?

Notre interface

La consigne du projet mentionnait deux choses :

- Calculer le temps moyen de résolution pour n coups
- Afficher le chemin empreinté

Nous laissons donc la possibilité de tester les algorithmes sur tout le fichier fourni ou bien seulement sur un état initial particulier.

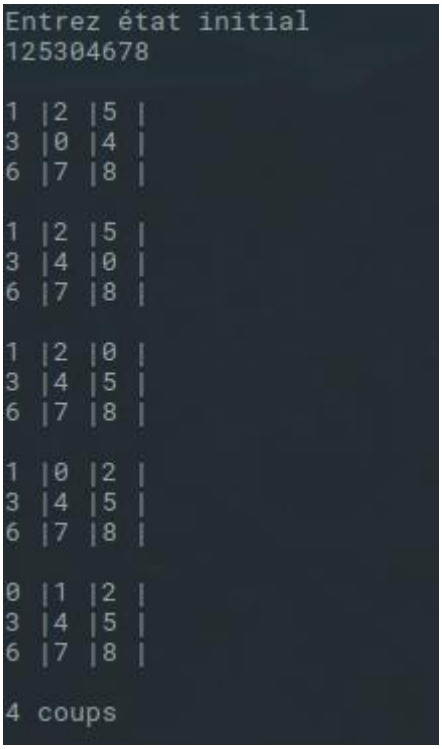
```
1 : Tester BFS sur tout le fichier
2 : Tester DFS sur tout le fichier
3 : Tester Astar sur tout le fichier
4 : Tester les 3 sur tout le fichier

5 : Tester BFS pour un état intial et afficher le chemin
6 : Tester DFS pour un état intial et afficher le chemin
7 : Tester Astar pour un état intial et afficher le chemin (Manhattan par défault)
```

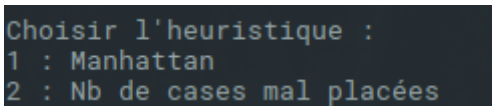
Voici un exemple si vous choisissez l'une des trois premières options :

Nb de coups	Temps
2	7.332545E-4(2)
4	2.86853E-4(2)
5	2.7090766666666667E-4(3)
6	3.00635E-4(2)
7	5.86914625E-4(16)
8	3.632277727272727E-4(22)
9	5.298741315789473E-4(38)

Avec un seul chemin :



Vous pouvez par ailleurs choisir l'heuristique pour l'algorithme A* :



Nos résultats

Ces tableaux récapitulent les temps de résolution des taquins (en secondes) pour les différents algorithmes BFS, DFS et A*.

H1= Nombre de tuiles mal placées

H2= Distance de Manhattan

	BFS	**DFS**	**A*(H1)**	**A*(H2)**	**Meilleur**
2	0.00202824	0.039383634	5.014625.1E-4	7.342095E-4	A*(h2)
4	0.00208815	7.4122E-5	2.43442.1E-4	3.080615E-4	DFS
5	0.00151008	0.016027493	3.2036266666E-4	2.80605666666E-4	A*(h2)
6	0.00166336	1.20489E-4	3.084385E-4	7.680145E-4	DFS
7	0.00197849		5.302105625E-4	6.765766875E-4	A*(h1)
8	0.00295301		4.0694063636E-4	6.54162681818E-4	A*(h1)
9	0.00581648		4.1414173684E-4	2.23645342105E-4	A*(h2)
10	0.00537968		2.3854440277E-4	3.37048069444E-4	A*(h1)
11	0.01433486		3.7456220253E-4	2.26637772151E-4	A*(h2)
12	0.03946887		4.0228933663E-4	2.27559277227E-4	A*(h2)
13	0.13700033		7.474657623762	4.14763881188E-4	A*(h2)
14	0.32837571		0.001090330495	3.34505594059E-4	A*(h2)
15	1.33787415		0.002315114851	8.00004029702E-4	A*(h2)
16			0.003810794637	8.86039554455E-4	A*(h2)
17			0.01318752928	0.0014900175544	A*(h2)
18			0.02640660911	0.00175649831	A*(h2)

	BFS	DFS	A*(H1)	A*(H2)	Meilleur
19			0.078039255643	0.00370938034653	A*(h2)
20			0.151862331841	0.00714871468316	A*(h2)
21			0.825960473198	0.01334178756435	A*(h2)
22			2.095116220306	0.02535251724752	A*(h2)
23			5.309136752702	0.05531318085148	A*(h2)
24			9.333392133613	0.09087382956435	A*(h2)
25				0.15935426361386	A*(h2)
26				0.28598105494059	A*(h2)
27				0.7813489872772	A*(h2)
28				1.0479344906237	A*(h2)
29				3.05442485947524	A*(h2)
30				4.4857760240792	A*(h2)
31				5.48683982936842	A*(h2)
32				21,6058976	A*(h2)

Maintenant, nous allons comparer l'exécution du même algorithme sur deux machines différentes :

	A*(H2) PC	A*(H2) Serveur	Meilleur
2	7.342095E-4	0.0034881545	PC
4	3.080615E-4	0.001336827	PC
5	2.80605666666E-4	0.00134404733333333	PC
6	7.680145E-4	0.0014656895	PC
7	6.765766875E-4	0.0012808056875	PC
8	6.54162681818E-4	0.00132096436363636	PC
9	2.23645342105E-4	8.2954042105263E-4	PC
10	3.37048069444E-4	6.24439875E-4	PC
11	2.26637772151E-4	6.3232473417721E-4	PC
12	2.27559277227E-4	5.2753982178217E-4	PC
13	4.14763881188E-4	0.001056215306930	PC
14	3.34505594059E-4	0.0012158293267326	PC
15	8.00004029702E-4	0.001923316623762	PC
27	0.7813489872772	8.753937982524752	PC

Notre analyse

Informé VS Non informé

Au vu du premier tableau récapitulatif, il est clair que les algorithmes non informés ne rivalisent pas avec A*.

Bien que DFS parvienne à être le meilleur algorithme pour les parties qui se résolvent en 4 et 6 coups, il ne parvient pas à résoudre des parties de 7 coups en un temps raisonnable. Par ailleurs, lorsque nous analysons les résultats de DFS, on s'aperçoit que pour le second état initial, il trouve une solution de 434 coups (où la solution minimale est de 2 coups). Par conséquent, non seulement il ne parvient pas à trouver une solution raisonnable rapidement mais en plus, il est très loin de tout le temps donné la solution optimale.

Pour ce qui est de BFS, il parvient à résoudre des états initiaux rapidement mais le problème est que le nombre de branches à explorer augmente de manière exponentielle et ralentit considérablement la découverte d'une potentielle solution. Ainsi, la recherche dans un temps raisonnable n'est pas possible lorsqu'une partie se résout en plus de 15 coups.

Le choix de l'heuristique

Le choix de l'heuristique est lui aussi très important. Lorsque nous regardons les parties qui peuvent se résoudre en 24 coups (ou 23,22…), on voit qu'en moyenne la première heuristique met 9 secondes contre 0.09 pour la seconde. La seconde heuristique permet donc en moyenne de résoudre des parties 100 fois plus rapidement que la première. Une telle différence est plus que significative et incite donc à trouver la meilleur heuristique avant d'appliquer A* à un problème (cf Figure1).

Les choix algorithmiques

Nous pensons que choix de la représentation du puzzle par un string peut être contesté. Avant que vous ne nous donniez votre algorithme à compléter, nous avions fait le nôtre de A à Z en modélisant le puzzle par un tableau 2D de « byte » et avons remarqué une meilleure utilisation du CPU et une vitesse légèrement supérieure.

Par ailleurs, au sein de l'algorithme A*, nous avions initialement opté pour des LinkedList pour représenter « l' open list » et la « closed list ». Cependant, après de nombreux essais, nous nous sommes aperçus que les ArrayList étaient bien meilleures en termes de performance et notamment pour obtenir le nième élément.

Le choix matériel

Au vue du deuxième tableau, il est très clair que le choix matériel est primordial. Pour les parties qui se résolvent en 27 coups, on observe que le PC est 10 fois plus rapide que son serveur. Nous vous l'accordons, les machines ont des spécifications techniques complètement différentes. Néanmoins, les algorithmes de recherches de chemins sont très gourmands en mémoire et en usage de CPU, le choix matériel n'est donc pas à négliger.

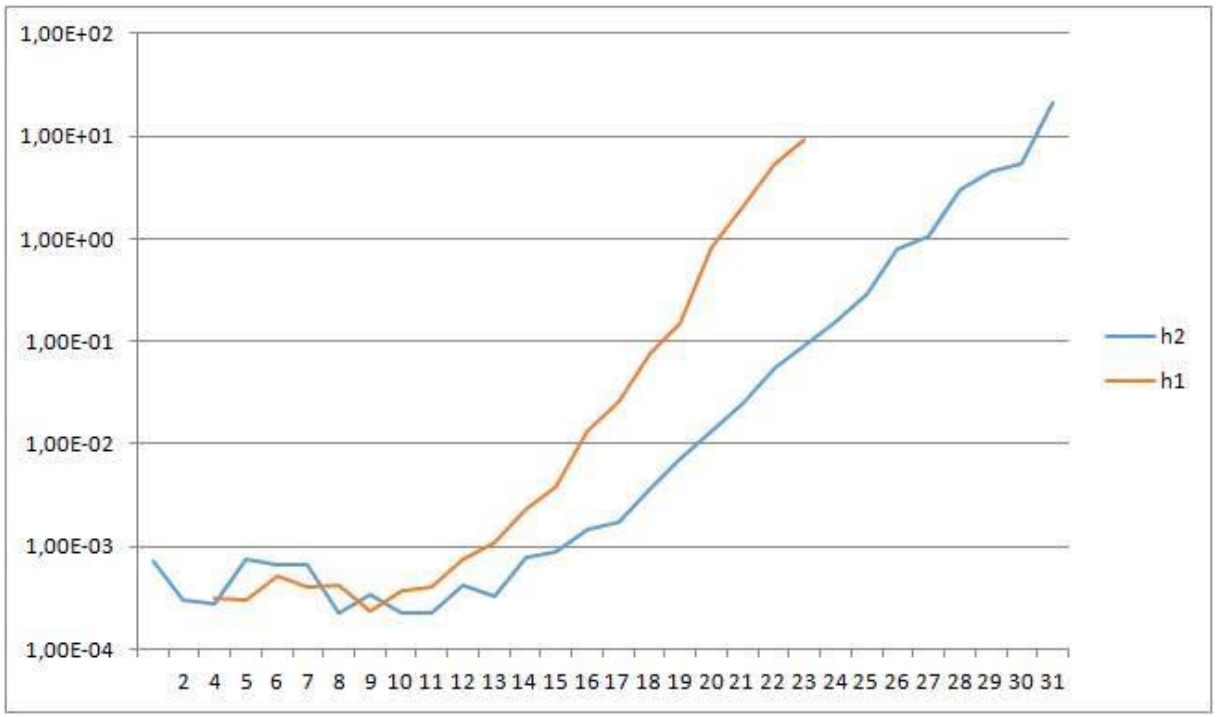


Figure 1 : Evolution des temps de calcul pour les 2 heuristiques

Conclusion

Finalement, ce projet nous a montré que les algorithmes informés sont bien meilleurs que les algorithmes non informés. Néanmoins, pour les premiers, le choix de l'heuristique est déterminant et peut diviser par 100 les temps de calculs. Par ailleurs, il ne faut pas oublier le choix matériel. En effet, même s'il peut paraitre négligeable, la vitesse de calcul de l'ordinateur et sa mémoire sont deux choses qui sont déterminantes sur les résultats. Finalement, même si A* ne donne pas toujours la meilleure solution, sa vitesse de calcul est rapide et apparait comme la solution à utiliser pour la recherche d'un chemin lorsque nous disposons d'une heuristique (plutôt que Dijkstra par exemple).