Florian Hassler 13.11.2017

Dimitri Lizzi

Méthodes d’optimisation
TP1

Table des matières

Série 1: Recherche aveugle et heuristique

Quatre stratégies de recherche ont été implémentées:

- **Brute** : recherche par force brute (aveugle). Les états suivants sont ajoutés à une queue non triée. Cette stratégie n'a pas été mesurée avec le puzzle 15 car elle est très lente.
- **CountIncorrect** : recherche heuristique. Les états suivants sont ajoutés dans une queue triée (algorithme heap queue) selon le nombre de tuiles mal placées dans le puzzle. Les états avec le moins de tuiles mal placées sont traités en premier.
- **Manhattan** : recherche heuristique. Les états suivants sont ajoutés dans une queue triée selon la somme de la distance manhattan de chaque tuile du puzzle.
- **Hybrid** : recherche heuristique. Les états suivants sont triés en fonction de la somme des métriques des deux stratégies précédentes (somme de manhattan + nombre de tuiles incorrectes)

Protocole de test:

Chaque stratégie a été lancée 10 fois, sur deux puzzles différents. La durée retenue est la moyenne de ces dix exécutions.

Mesures sur puzzle-8

5	4	0
6	1	8
7	3	2

Stratégie	**Etapes de la solution**	**Itérations**	**Durée moyenne [µs]**	**Temps moyen par itération [µs]**
CountIncorrect	47	501	7958.05	15.88
Manhattan	37	133	3726.62	28.02
Hybrid	75	716	20384.55	28.47
Brute	25901	156286	1235334.63	7.90

Mesures sur puzzle-15

2	1	3	15
7	10	4	8
14	6	13	5
11	9	12	0

Stratégie	**Etapes de la solution**	**Iterations**	**Durée moyenne [µs]**	**Temps moyen par itération [µs]**
CountIncorrect	127	3861	76838.6364	19.9
Manhattan	205	37484	1717378.831	45.82
Hybrid	209	9697	503884.5062	51.96

Conclusions

La méthode CountIncorrect semble être celle qui s'exécute le plus rapidement car la durée d'une itération est plus courte. Selon les puzzles, la méthode qui trouve la solution en moins d'étapes varie.

La méthode brute a un temps d'itération très rapide mais les solutions trouvées sont loin d'être optimales et il faut énormément d'itérations pour arriver à une solution.

Série 2: Satisfaction des contraintes

Afin de pouvoir faire une comparaison de l'efficacité de la résolution de sudoku par la satisfaction de contrainte, deux algorithmes différents ont été implémentés.

Backtracking simple

L'algorithme de backtracking simple ou recherche en profondeur sans heuristique est très simple.
Il s'agit simplement de prendre la première case vide disponible et d'y insérer la première valeur possible. Puis, on fait appel à nouveau la fonction jusqu'à ce que le sudoku soit résolu ou jusqu'à arriver à un état sans solution. Dans le deuxième cas, on retourne en arrière puis on choisit une des autre valeur possible afin d'essayer à nouveau d'aller le plus loin possible dans la résolution.
Si toutes les combinaisons de valeur possible ont été essayées dans toutes les cases vides, le sudoku n'est pas résolvable.

Backtracking heuristique

Cet algorithme utilise l'heuristique de la variable la plus contrainte. Dans le cas du sudoku, il s'agit de la case vide pour laquelle il y a le moins de valeur possible. Cette heuristique est souvent celle que nous utilisons inconsciemment lorsque nous résolvons un sudoku à la main.
A chaque appel de la fonction de résolution, toutes les valeures possible de toutes les cases vides sont calculées. De cette manière, cela réduit de manière conséquente le choix de valeur à insérer et il est également facile de faire un classement des cases le plus contraintes. De plus, on vérifie que le nombre de case vide soit égal à la taille du tableau des cases les plus contraintes. Si ce n'est pas le cas, cela veut dire que la dernière valeur insérée rend impossible la résolution du sudoku.
L'algorithme prend à chaque fois la première case du tableau des cases les plus contraintes et insère sa première valeur possible. Puis on appel à nouveau la fonction qui va mettre à jour les tableau des valeurs possibles et des cases les plus contraintes jusqu'à arriver à remplir totalement le sudoku.
Si toutes les combinaisons de valeur possible ont été essayées dans toutes les cases vides, le sudoku n'est pas résolvable.
Il devrait s'agir de la stratégie la plus rapide.

Protocole de test

Les deux algorithmes ont été exécutés 10 fois sur des sudokus de différentes tailles. La durée retenue est la moyenne de ces dix exécutions.

Mesures sur sudoku 9x9

```
-----------------------------------------
|  --  --  --  |  --  --  --  |  08  01  --  |
|  --  04  --  |  08  --  06  |  --  --  --  |
|  --  --  05  |  --  --  02  |  --  09  --  |
-----------------------------------------
|  08  07  --  |  04  --  01  |  09  --  --  |
|  --  --  --  |  --  07  --  |  --  --  --  |
|  --  --  04  |  05  --  09  |  --  08  03  |
-----------------------------------------
|  --  06  --  |  01  --  --  |  04  --  --  |
|  --  --  --  |  06  --  05  |  --  02  --  |
|  --  02  08  |  --  --  --  |  --  --  --  |
-----------------------------------------
```

Stratégie	**Nb appel récursif**	**Nb affectation avant premier backtracking**	**Temps moyen d'exécution [s]**
Backtracking simple	18011	5	1.6147762748973036
Backtracking heuristique	176	34	0.49272361289986294

Mesures sur sudoku 16x16

```
-------------------------------------------------------------------------
|  --  15  --  01  |  --  02  10  14  |  12  --  --  --  |  --  --  --  --  |
|  --  06  03  16  |  12  --  08  04  |  14  15  01  --  |  02  --  --  --  |
|  14  --  09  07  |  11  03  15  --  |  --  --  --  --  |  --  --  --  --  |
|  04  13  02  12  |  --  --  --  --  |  06  --  --  --  |  --  15  --  --  |
-------------------------------------------------------------------------
|  --  --  --  --  |  14  01  11  07  |  03  05  10  --  |  --  08  --  12  |
|  03  16  --  --  |  02  04  --  --  |  --  14  07  13  |  --  --  05  15  |
|  11  --  05  --  |  --  --  --  --  |  --  09  04  --  |  --  06  --  --  |
|  --  --  --  --  |  13  --  16  05  |  15  --  --  12  |  --  --  --  --  |
-------------------------------------------------------------------------
|  --  --  --  --  |  09  --  01  12  |  --  08  03  10  |  11  --  15  --  |
|  02  12  --  11  |  --  --  14  03  |  05  04  --  --  |  --  --  09  --  |
|  06  03  --  04  |  --  --  13  --  |  --  11  09  01  |  --  12  16  02  |
|  --  --  10  09  |  --  --  --  --  |  --  --  12  --  |  08  --  06  07  |
-------------------------------------------------------------------------
|  12  08  --  --  |  16  --  --  10  |  --  13  --  --  |  --  05  --  --  |
|  05  --  --  --  |  03  --  04  06  |  --  01  15  --  |  --  --  --  --  |
|  --  09  01  06  |  --  14  --  11  |  --  --  02  --  |  --  --  10  08  |
|  --  14  --  --  |  --  13  09  --  |  04  12  11  08  |  --  --  02  --  |
-------------------------------------------------------------------------
```

La complexité de ce sudoku ne permet pas à l'algorithme de backtracking simple de le résoudre dans un temps raisonnable. Le test a donc été effectué qu'une seule fois.

Stratégie	**Nb appel récursif**	**Nb affectation avant premier backtracking**	**Temps moyen d'exécution [s]**
Backtracking simple (1 exécution)	14579935	8	2733.236503942 (environ 45,50 min)
Backtracking heuristique (10 exécutions)	363	25	4.6136740791002

Conclusion

D'après les résultats des tests, la stratégie de résolution du sudoku utilisant l'heuristique de la variable la plus contrainte est systématiquement la plus rapide. En effet, son temps d'exécution est à chaque fois le plus court et de plus elle semble plus efficace car elle fait moins d'appel récursif et elle affecte beaucoup plus de cases avant de faire une première récursion pour trouver la solution du sudoku.