TP1

Heuristiques & Algorithmes de Recherche

Etienne Collin | 20237904,
Marylou Fauchard | 20218608

Intelligence Artificielle : Introduction - IFT3335

Section A

Professeur Jian-Yun Nie

Université de Montréal

Hiver 2024
À remettre le 16 Février 2024 à 23:59

Table des matières

Table des matières **1**

1 Introduction **2**
1.1 Situation et objectif 2
1.2 Analyse du troisième critère 2

2 Heuristiques **2**
2.1 Naked Pair 2
2.2 Hidden Pair 2
2.3 Comparaison de performances et discussion 2

3 Algorithmes **3**
3.1 Hill climbing 3
3.2 Simulated annealing 3
3.3 Comparaison de performances et discussion 4

4 Fonctionnement du code **5**

1 Introduction

1.1 Situation et objectif

Dans le cadre du cours d'introduction à l'intelligence artificielle, nous devions travailler sur les heuristiques et les algorithmes utilisés pour résoudre le fameux jeu du sudoku.

1.2 Analyse du troisième critère

Lors de la deuxième question, nous devions changer le troisième critère. En effet, dans le code fourni, nous sélectionnons la case comme celle ayant le moins de chiffres possibles, ou autrement dit, le moins d'indices, possible.

La question à l'étude était de voir l'importance de choisir une bonne case de départ. Comme la table suivante le montre, pour tous les cas, on a qu'il vaut mieux choisir la case avec le moins d'indices possibles que d'en choisir une au hasard. On voit que le temps moyen est toujours plus haut pour cette deuxième technique, et que le temps maximal est largement supérieur en utilisant une case aléatoire qu'en utilisant la stratégie qui était déjà implémentée. Il est important de noter que peu importe la technique, tous les sudokus ont été résolus. Les sudokus 99* sont générés aléatoirement et peuvent être insolubes.

	Version minimale				Version aléatoire			
Nombre de sudokus	95	99*	100	1000	95	99*	100	1000
Temps moyen (ms)	4.77	1.49	1.40	1.55	11.18	3.75	1.55	1.78
Temps maximal (ms)	20.70	3.89	2.11	4.56	81.15	208.17	3.07	7.51
Hz (sudokus/s)	210	671	717	646	89	208	647	560

2 Heuristiques

Pour chaque heuristique, nous expliquerons premièrement ce qu'elle est supposée faire. Par la suite, nous expliquerons comment nous l'avons avons implémentée. Finalement, nous ferons une analyse de performance entre les heuristiques, incluant l'heuristique nulle.

2.1 Naked Pair

Intuitivement, on sait que si deux cases qui sont respectivement dans le voisinage de l'autre et que chacune de ces deux cases ont seulement 2 indices possibles et que ces indices sont les mêmes, alors les autres cases dans le voisinage commun ne peuvent pas prendre ces indices. On définit le voisinage d'un carré comme les autres cases qui sont dans le même carré (grille 3x3), la même ligne ou la même colonne.

Lors de l'implémentation, nous avons regardé, pour le sommet concerné, si ce dernier a exactement deux possibilités, s'il avait un voisin avec les mêmes possibilités. Si c'est le cas, il y a une naked pair et nous avons enlevé ces indices des cases voisines communes lorsque c'était le cas.

À noter que d'autres versions de cette heuristique ont été considérées, mais qu'aucune n'apportait une amélioration significative. Il était plus pertinent de rapporter uniquement le cas simple. En effet, nous avons regardé si le naked triplet pourrait être une bonne heuristique, mais les temps ne sont pas mieux et la logique étant la même, pour rendre un rapport et un code plus simple, cette technique n'a pas été plus analysée.

2.2 Hidden Pair

Cette deuxième heuristique affecte une paire de cases lorsque ces deux cases ont deux indices qui ne se retrouvent nul part ailleurs dans les cases voisines communes à la paire. Ainsi, pour voir si une case est dans une hidden pair, nous avons regardé s'il y avait une case voisine avec au moins deux indices en commun. Lorsque c'est le cas, on vérifie que les indices communs qui sont uniques à ces deux cases là, c'est-à-dire uniques à la paire, sont au nombre de deux. Dans ce cas, on peut enlever pour la paire tous les indices qui ne sont pas ceux de la paire de possibilités.

2.3 Comparaison de performances et discussion

Pour chacune des heuristiques, ainsi que l'heuristique nulle, et avec le critère de choisir la case avec le moins de possibilités, nous allons montrer les temps moyens et maximaux pour 100 et 1000 puzzles. Les temps qui sont dans le tableau ont été obtenus en faisant une moyenne du temps de résolution sur 1000 exécutions du fichier de 100 sudokus et 1000 sudokus.

	Sans heuristique		Naked Pair		Hidden Pair	
Nombre de sudokus	100	1000	100	1000	100	1000
Temps moyen (ms)	1.50	1.59	1.53	1.65	1.59	1.68
Temps maximal (ms)	2.31	4.71	2.20	4.75	2.36	4.88

Contrairement à ce qui est souhaité, on ne voit pas une différence écrasante pour les heuristiques, avec des temps plus longs dans certains cas que seulement la recherche qui était faite sans heuristique. Nous avons essayé, au meilleur de nos capacités, d'optimiser le code, en remplaçant par exemple les fonctions "set.intersection()" par "&". Ce dernier changement a amélioré les temps. Toutefois, les améliorations n'étaient pas suffisantes pour améliorer significativement les temps. Aussi, même si nos heuristiques améliorent pas le temps, cela ne veut pas dire qu'elles sont mauvaise. Effectivement, le nombre d'itérations pourrait être plus petit pour résoudre le problème, mais chaque itération pourrait être plus lente à cause de la complexité ajoutée. Nous hypothetisions une amélioration des résultats plus importantes provenant de l'utilisation d'heuristiques. Les variations de performance entre nos heuristiques sont d'environ 10% et ne sont peut-être pas suffisant pour discréditer les heuristiques implantées. D'autre part, il faut savoir que ces heuristiques, bien qu'elles soient populaires, ne sont peut-être pas les plus efficaces. Il aurait peut-être été plus efficace de considérer un mélange d'heuristiques ou simplement d'autres heuristiques.

3 Algorithmes

Pour chacun des deux algorithmes implantés, nous allons expliquer en quoi ils consistent en quelques mots et les points qui, selon nous, sont important à apporter quand à notre implémentation. Ensuite, nous analyserons les perfomances des deux algorithmes selon leur temps d'exécution, selon le nombre de sudokus qu'ils peuvent résoudre et nous présenterons également des graphiques sur le nombre de conflits à travers les itérations pour chaque méthode.

3.1 Hill climbing

La méthode de hill climbing consiste à comparer le nombre de conflits entre deux versions d'un jeu de sudoku. Une version a deux valeurs interchangées par rapport à l'autre. On adopte la solution avec le moins de conflits et on continue. Lors du parsing du tableau de sudoku initial, on remarque que si le parsing assigne les chiffres possibles aux cases sans prendre en compte les contraintes de lignes et de colonnes, alors le hill climbing ne résout aucun sudoku. Or, le nombre de sudokus résolut grimpe significativement à environ 44% si on change la méthode d'initialisation pour un qui considère ces contraintes.

3.2 Simulated annealing

La méthode de simulated annealing permet de trouver des solutions lorsque le hill climbing aurait normalement été pris sur un maximal local. En effet, en ayant une température, qui changera au cours du programme, on aura une certaine probabilité de sauter à une autre solution qui n'apporte pas d'améliorations immédiates. Lors de l'implémentation, nous avons premièrement mis une limite au nombre d'itérations qui seront effectuées à 10000. Afin de ne pas faire trop d'étapes inutiles, nous avons également imposé de terminer l'algorithme lorsque le sudoku est résolut, ou si la température est inférieur à un seuil que nous avons choisit à 10^{-10}. La variable t, qui représente ladite température, doit avoir une valeur initiale. Comme indiqué dans les instructions, nous avons testé premièrement avec une valeur $t = 3$ puis par la suite pour d'autres valeurs telles que $t = 1$. Nous avons également joué avec la "constante de scheduling". C'est-à-dire, la valeur qui multiplie t à chaque itération de l'algorithme. Nous avons testé plusieurs valeurs, mais celles qui avaient le plus grand impact sont 0.99 et 0.999.

Dans le prochain tableau, notons les abréviations suivantes : NH veut dire sans heuristiques, NP naked pair, HP, hidden pair. Pour ce qui est du parsing, NRP tient pour non random parsing ce qui signifie que la table a été "parsée" en prenant en compte les contraintes de lignes et de colonnes, alors que RP (random parsing) est la méthode instauré avec le hill climbing qui tient compte uniquement du voisinage dans la boite 3x3.

Simulated Annealing	T = 1								T = 3							
Constante	0.99				0.999				0.99				0.999			
Parsing	NRP			RP	NRP			RP	NRP			RP	NRP			RP
Heuristique	NH	NP	HP	NH	NH	NP	HP	NH	NH	NP	HP	NH	NH	NP	HP	NH
Temps moy (ms)	33	34	33	62	65	70	65	99	36	37	40	67	76	75	76	99
Temps max (ms)	77	78	82	80	336	341	339	347	91	84	88	92	304	315	323	346
Hz (sudokus/s)	30	29	30	16	16	14	15	10	28	27	25	15	13	13	13	10
Nbr sudokus résolus	80	72	75	51	95	95	97	99	74	78	67	56	97	97	98	99

Cette table contient beaucoup d'informations que nous essaierons de résumer. Premièrement, il est intéressant d'observer l'effet de la constante sur le taux de réussite. Une constante multiplicatrice plus petite affecte négativement le nombre de sudokus résoluts. C'est le cas pour les deux températures. Aussi, dans l'ensemble, avoir une constante multiplicatrice plus haute permet d'améliorer le nombre de sudokus résoluts, mais augmente également le temps moyen pour en résoudre un. L'effet d'augmenter la température initiale est plus subtil : les résultats semblent être meilleurs avec une plus grande température initiale, surtout

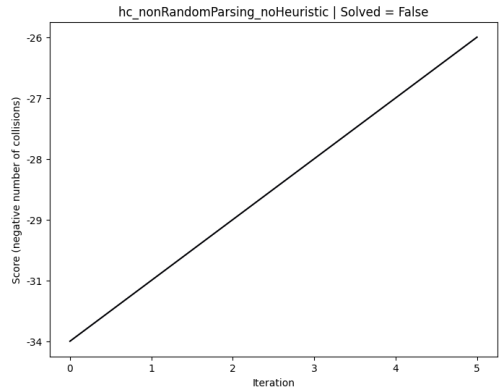
si la constante multiplicatrice est haute, et les temps sont également un peu plus haut lors de ce cas. Cela peut être expliqué par le fait que la température initiale et la constante sont intimement liées ; la constante multiplie la température, à partir de la température initiale, jusqu'à ce que la température atteigne zéro. Ainsi, il est normal que l'effet des deux variables soit le même avec un ordre de grandeur différent. Cependant, on voir que la présence des heuristiques, particulièrement Naked Pair, affecte plus grandement les résultats si on a une température initiale de 3 versus de 1. Ainsi, on voit qu'il est très important de bien choisir la constante multiplicatrice et la température initiale.

3.3 Comparaison de performances et discussion

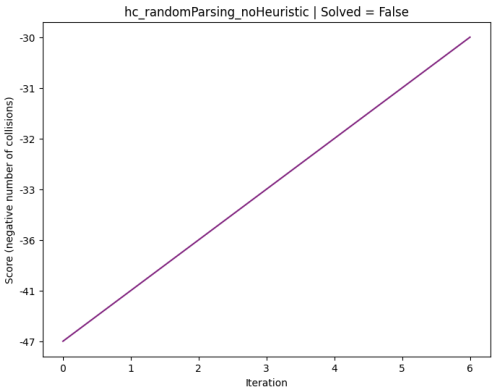
Pour analyser la différence entre les algorithmes de hill climbing et de simulated annealing, respectivement surnommés HC et SA, différentes visualisations seront utilisées telles que des graphiques de contradictions, des tables de temps et de réussite. Premièrement, regardons une table qui résume les perfomances. À noter qu'étant donné le temps nécessaire pour exécuter le code générant les statistiques de résolution, nous avons seulement effectué 30 essais. Il faut donc prendre une variabilité en compte lors de l'analyse des résultats. Aussi, étant donné la limite en taille du rapport, nous rapporterons uniquement les résultats pour les deux algorithmes en utilisant aucune heuristique.

Algo	Hill Climbing		Simulated Annealing			
parsing	NRP	RP	NRP	RP	NRP	RP
constante	-		0.99		0.999	
temps moy (ms)	7.06	27.98	29.40	51.21	52.64	83.06
temps max (ms)	22.62	49.27	74.88	74.86	272.39	226.73
Nb sudokus réussis sur 100	44.07	0.0	74.60	76.47	96.97	99.87

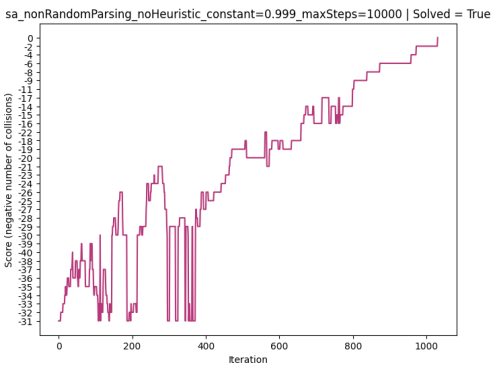
On voit que la méthode hill climbing, bien que plus rapide, résout beaucoup moins de sudokus et ce, même si on fait un parsing random (prenant seulement les informations de la boite 3x3). On remarque que le RP semble aider le SA à résoudre les sudokus, bien que plus lentement. À l'inverse, le HC n'arrive pas à résoudre des sudokus avec le RP. Les autres conclusions qu'on peut tirer de la table sont similaires à celles précédentes sur les perfomances de simulated annealing.



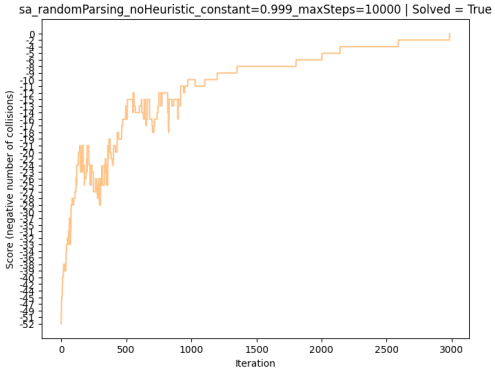
(a)



(b)



(c)



(d)

On voit de cette figure que le score, qui représente la valeur négative du nombre de contradictions, a un comportement semblable en général peu importe le parsing utilisés (non aléatoire à gauche et aléatoire à droite). Il y a beaucoup plus de variations avec simulated anealing comme il est attendu puisque l'algorithme peut choisir une solution avec plus de contradictions

selon la température. Sur l'axe du score, une valeur de 0 implique que le sudoku est résolu. On remarque alors que le hill climbing est resté pris sur un maximum local et n'a pas réussit à résoudre le sudoku.

4 Fonctionnement du code

Le fichier à exécuter est sudoku.py. Il faut s'assurer que les fichiers contenant des sudokus sont placés dans le même dossier que l'exécutable. Une fois sudoku.py exécuté, les statistiques de résolution des sudokus seront imprimées pour les différents algorithmes. Des fonctions utilitaires se trouvent entre les lignes 50 et 69, les fonctions pour le DFS et les heuristiques entre les lignes 69 et 214, les fonctions pour le hill climbing entre les lignes 214 et 388 et finalement les fonctions pour le simulated annealing entre les lignes 388 et 460. Les lignes 460 jusqu'à la fin sont dédiées aux fonctions utilitaires et de tests. La signature de la fonction "solve_all()" a été modifiée et ces modifications sont expliquées dans la docstring.