TP1 – IFT3335
Jeu de Sudoku
À rendre au plus tard le16 février 2024 avant 23:59
À faire en groupe de 2 personnes
Ce TP correspond à 15% dans la note globale

1. Buts du TP

1. Comprendre comment utiliser des heuristiques pour la recherche dans l'espace d'états
2. Comparer les performances de différents algorithmes.
3. Développer des heuristiques pour ce jeu

2. Le jeu de Sudoku

Ce jeu d'origine japonaise est représenté en une grille de 9x9, séparée en 9 carrés de 9 cases. Certaines cases contiennent déjà un chiffre au départ, de 1 à 9. Le but est d'arriver à remplir les cases vides de telle façon que chaque carré, chaque ligne et chaque colonne contiennent les chiffres 1-9 sans répétition. Autrement dit, on ne doit pas trouver 2 chiffres identiques dans un même carré, une même ligne ou une même colonne.

Voici une configuration de départ (à gauche) et sa solution (à droite) :

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Ce jeu est l'objet de nombreuses recherches, et beaucoup d'heuristiques ont été développées pour trouver des solutions de façon efficace, c'est-à-dire de réussir avec le moins de coups tentés possible. Vous pouvez trouver une bonne description du jeu et des heuristiques sur la page de Wikipédia :
http://fr.wikipedia.org/wiki/Sudoku
Une autre page Wikipédia donne une description plus courte, mais plus utile pour votre implantation dans ce TP :
https://fr.wikipedia.org/wiki/Algorithmes_de_r%C3%A9solution_des_sudokus

3. Le cadre de base pour votre travail

Le but principal de ce TP n'est pas de créer un programme qui résout un Sudoku en un temps record, mais de pratiquer l'implantation de différents algorithmes de recherche dans l'espace d'états, de comparer leur performance, et de développer quelques heuristiques utiles vous-mêmes. Sudoku sert de support pour cette fin.
Vous pouvez trouver une implantation de base faite par Norvig ici :
http://www.norvig.com/sudoku.html

Le programme en Python est aussi affiché sur Studium. Vous pouvez utiliser cette implémentation comme base.

Dans cette implantation, Norvig a utilisé 3 idées :

1. Recherche sous contrainte : En examinant les chiffres dans le même carré et dans les mêmes lignes et colonnes, on peut déterminer les chiffres qu'on peut mettre à une position (chiffres candidats). Ces contraintes aident beaucoup à réduire les chiffres qu'on peut tenter à chaque position.
2. Recherche en profondeur avec retour en arrière (backtracking) : C'est une recherche en profondeur d'abord, qui permet de revenir en arrière sur un placement tentative fait avant, si jamais le jeu se bloque. L'algorithme correspond bien à l'algorithme de recherche en profondeur d'abord présenté dans le cours.
3. Ordonner les positions : Il ordonne les positions à essayer de telle manière qu'une position avec un plus petit ensemble de chiffres candidats sera placée avant. On va donc tenter de placer un chiffre candidat d'abord à une telle position. La raison est expliquée dans la page web de Norvig. Ce critère est un critère heuristique typique pour ce jeu.

Cette implantation fournit un cadre de base pour faire vos implantations. Elle fournit aussi une façon de modéliser le jeu en espace d'états. Le travail que vous devez faire est d'implanter quelques autres algorithmes pour le jeu, d'ajouter vos propres heuristiques et de comparer leurs performances.

4. Travail à réaliser :

Voici les travaux que vous devez réaliser.

1. (10%) Faire fonctionner le programme de Norvig (en Python) sur un ensemble de cas de Sodoku (voir les 100 configurations de test sur Studium).

2. (10%) Désactiver le 3ième critère qui choisit la case avec le moins de possibilités sous la ligne
   `## Chose the unfilled square s with the fewest possibilities`
   et le remplacer par choisir une case et un chiffre au hasard (ou dans un ordre prédéterminé). Ceci revient à une recherche pure en profondeur d'abord sans aucune heuristique. Vous allez comparer, dans votre rapport, la performance (taux de réussite et le temps) de cette version simplifiée avec la version originale de Norvig pour voir l'effet d'utiliser cette heuristique.

3. (20%) Implanter au moins une autre heuristique, en plus du 3ième critère de Norvig. Ici, on ne vous impose pas quelles heuristiques à utiliser. Si vous n'êtes pas spécialiste de ce heu, vous pouvez chercher dans les articles sur le Web (ainsi que les articles sur Studium) pour trouver une ou quelques heuristiques à implémenter. Ces heuristiques devraient permettre d'améliorer la performance des algorithmes implantés décrits en haut (vous devez le confirmer dans vos tests). Un des articles que vous pouvez lire est celui d'Angus Johnson dans http://www.angusj.com/sudoku/hints.php (une traduction en français est aussi disponible sur http://www.angusj.com/sudoku/ (Cliquer sur « French translation ») qui décrit un ensemble d'heuristiques possibles. Même si on vous exige d'implanter seulement une heuristique, vous êtes encouragés à essayer plusieurs heuristiques pour voir quelle heuristique fonctionne mieux.

4. (20%) Implanter l'algorithme Hill-Climbing (recherche glouton) pour ce problème. Pour cela, à partir de la configuration de départ, on remplit chaque carré 3x3 avec un de chiffres possibles au hasard, mais en vérifiant les contraintes pour le même carré. Ceci va engendre des conflits sur les lignes et les colonnes. Ensuite, on utilise Hill-Climbing pour tenter d'améliorer la configuration le plus possible, en interchangeant (swap) deux des chiffres remplis dans un carré. L'amélioration consiste à réduire le nombre de conflit global (sur les lignes et les colonnes). Si aucune amélioration n'est possible, l'algorithme s'arrête. L'algorithme est brièvement décrit dans la page Wikipédia, appelé « Recherche stochastique / méthodes d'optimisation ».

5. (20%) Utiliser le recuit simulé (simulated annealing). Hill-Climbing peut être coincé sur un optimum local sans pouvoir arranger tous les chiffres correctement (i.e. d'arriver à une solution). On peut améliorer cet algorithme en utilisant le recuit simulé. Cet algorithme est décrit dans le cours, et aussi dans un article par Lewis (voir sur Studium). Vous êtes demandé à implémenter une version simplifiée de l'algorithme décrit par Lewis. Notamment, vous utilisez la même stratégie pour diminuer la température : $t_{i+1} = \alpha \cdot t_i$ avec α=0.99. Pour la température initiale, Lewis la détermine selon la dérivation standard des gains produits par un petit nombre d'essais. Pour ce devoir, vous pouvez utiliser une stratégie simplifiée en la fixant à une valeur. La valeur de 3 peut être testée d'abord. Vous pouvez faire varier cette température de départ pour tester le comportement de l'algorithme. L'article parle aussi de recuit simulé homogène (homogeneous SA). Vous n'en tenez pas compte dans ce devoir.

   Normalement, l'utilisation du recuit simulé devrait améliorer le taux de succès sur Hill-climbing.

6. (20%) Comparer les algorithmes en les faisant fonctionner sur 100 configurations de départ (voir la liste des configurations sur Studium). On compare le pourcentage de jeux réussis (pour lesquels l'algorithme trouve la solution), ainsi que le temps utilisé. Faites votre analyse personnelle selon ce que vous observez dans ces tests (la complexité de l'algorithme, le taux de succès, …) pour voir si ce que vous observez correspond bien aux analyses faites dans le cours. Vos comparaisons et analyses doivent être décrites dans un petit rapport de 2-3 pages.

5. À rendre

Ce TP est à faire en groupe de **2 personnes** normalement. Si vous êtes incapables de former un groupe, travailler seul ou dans un groupe de 3 personnes est toléré exceptionnellement. Vous devez cependant avoir l'accord du professeur au préalable.

Vous avez deux choses à rendre : Un rapport et des programmes.

1. Un rapport de 2-3 pages doit contenir une description brève de ce que vous avez réalisés dans ce TP. Vous devez décrire l'heuristique (ou les heuristiques) que vous avez implantée pour la question 3. Votre rapport doit aussi contenir une comparaison des performances de différentes méthodes.
   Indiquez dans votre rapport également le programme correspondant à chaque algorithme que vous avez implémenté. Indiquez dans votre rapport comment vos programmes doivent fonctionner. Les programmes que vous rendez doivent s'exécuter correctement.

2. Les programmes que vous avez implémentés (intégrés dans le programme de Norvig).

Gare au plagiat :

- Il existe beaucoup de programmes de Sudoku sur le Web. Vous ne pouvez pas prendre une implantation sur le Web comme votre devoir. Si vous rendez un tel programme pris directement sur le Web, vous aurez la note de 0.
- Vous ne pouvez pas demander à GPT ou un autre modèle de langue de créer un programme pour vous.
- Si 2 groupes rendent les mêmes programmes (ou les mêmes programmes masqués), les deux groupes auront 0.

Date de remise :

La date limite pour la remise est le 16 février avant 23:59. Vous devez remettre le rapport et les programmes sur Studium. Après cette date, une pénalité de 2/15 points sera appliquée pour chaque jour de retard.

Évaluation :

Ce TP compte pour 15% dans la note globale. La même note sera donnée à tous les membre du même groupe.

6. Ressources Sur le Web

- 1000 configurations de départ sur le site Sodoku Garden (d'où les 100 configurations de test sont prises). Chaque configuration occupe une ligne, composée de 81 chiffres, correspondant aux 81 cases, avec les 9 lignes concaténées. 0 signifie que la case et vide (à remplir par votre programme).
- Vous pouvez pratiquer en ligne ici : http://www.websudoku.com