TP1 – IFT3335
Jeu de Sudoku
(à rendre au plus tard le 9 oct. avant 23 :59)

1. Buts du TP

1. Comprendre comment on peut formuler un problème d'IA comme un problème de recherche dans l'espace d'états
2. Comprendre comment développer des heuristiques
3. Pratiquer sur l'implantation (en Python ou en Java) et utilisation des algorithmes pour un cas concret

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 ligne ou une 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. Vous pouvez trouver une bonne description du jeu et des heuristiques sur la page de Wikipédia :
http://fr.wikipedia.org/wiki/Sudoku

3. Le travail dans ce TP

Le but de ce TP n'est pas de créer un programme qui résout un Sudoku en un temps record. Sudoku sert de support pour pratiquer certains algorithmes présentés dans le cours :

- recherche en profondeur d'abord
- recherche locale Hill-Climbing
- recherche heuristique

Ces algorithmes sont déjà implantés dans le package en Java et en Python qui accompagne le livre de référence (Russell & Norvig). Vous avez seulement à faire quelques modifications dans ces implantations (voir ci-après), et faire fonctionner ces programmes pour ce problème Sudoku.

3.1. Modifications et ajouts à faire :

- Ajouter un comptage dans ces algorithmes pour compter le nombre de nœuds explorés.
- Imposer une limite de N nœuds explorés pour ces algorithmes. Si à l'intérieur de cette limite, l'algorithme ne trouve pas une solution, il échoue. Ceci simule un temps limite pour le jeu.
- Implanter des méthodes qui vérifient si mettre un chiffre dans une case engendrera des conflits dans un carré, dans une ligne et dans une colonne.
- Déterminer les chiffres qui peuvent aller dans une case sans violer les contraintes.
- Implanter des fonctions heuristiques.

3.2. Travail à réaliser :

1. (20%) (Sur papier, dans votre rapport) Formuler ce problème comme un problème de recherche dans l'espace d'états. Pour cela, il faut définir les éléments suivants :
    - Définir la notion d'état pour ce problème et proposer une représentation (structure de nœud) ;
    - Définir l'état de départ et l'état but (ou une fonction de vérification de but) ;
    - Définir la relation de successeur ;
    - Définir le coût d'étape (si nécessaire).

    Cette formulation doit figurer dans le rapport.
2. (20%) Faire fonctionner l'algorithme de profondeur d'abord sur un ensemble de configuration de départ (voir les entrées précisées plus tard), en imposant une limite (e.g. 10 000) de nœuds explorés au maximum. Si la solution n'est pas trouvée avec cette limite imposée, le résultat est un échec. Dans vos tests plus tard, vous allez faire varier cette limite. Dans cet algorithme, à chaque tour, on choisit au hasard une case à remplir par un chiffre possible. Vous pouvez utiliser une façon quelconque pour ordonner les cases vides.
3. (20%) Faire fonctionner l'algorithme de Hill-Climbing pour ce problème. Pour cela, à partir de la configuration de départ, on remplit chaque carré avec des chiffres différents, de façon aléatoire, en respectant la contrainte de ne pas répéter des chiffres dans le même carré. Ensuite, on utilise Hill-Climbing pour tenter d'améliorer la configuration le plus possible, en inter-changeant deux des chiffres remplis dans un carré. L'amélioration consiste à réduire le nombre de conflit global (sur les lignes et dans les colonnes). La fonction heuristique est définie comme le nombre de conflits restants.
4. (20%) On peut aussi utiliser d'autres heuristiques dans ce jeu. Par exemple, à partir d'une configuration de départ, on peut déterminer les chiffres qu'on peut mettre dans chaque case. Une case est plus contraignante si elle peut accepter moins de chiffres. Une heuristique simple est de tenter de remplir un chiffre dans la case la plus contraignante d'abord. La fonction heuristique (h) pour une case est alors définie comme le nombre de chiffres possibles.

    La fonction f peut être définie comme f=h, ou bien comme f=g+h, où la fonction g correspond au nombre de remplissages déjà effectués. Ces fonctions f peuvent être facilement intégrées dans l'algorithme best-first.

    L'heuristique ci-dessus est un des trucs simples utilisés par des joueurs humains (Il y a d'autres trucs décrits par Angus Johnson dans http ://www.angusj.com/sudoku/hints.php, qui pourrait vous inspirer).

Une autre heuristique un peu plus sophistiquée que vous pouvez essayer (optionnel) est de tenir compte aussi la somme des possibilités restant dans les autres cases vides après avoir placé un chiffre à une case. En d'autres mots, vous tenez compte des contraintes que ce placement créera pour les autres positions vides. On privilégiera le placement qui imposera les plus de contraintes, i.e. engendra le moins de possibilités pour les cases vides restantes.
Par rapport à un placement aléatoire, l'avantage de tester la case la plus contraignante est qu'on peut arriver plus rapidement à une conclusion, soit ce placement amènera à un échec, soit il aboutit à un succès.

5. (20%) Comparer les algorithmes en les faisant fonctionner sur 100 configurations de départ (voir la liste des configurations sur le web du cours). On compare le nombre de nœuds explorés avant de trouver la solution (ceci nécessite une modification du package pour compter le nombre de nœuds explorés), et le taux de succès avec la limite de 500, 1000, 5000, 10000 etc. de nœuds explorés (testez les nombres que vous jugez intéressants pour votre analyse). Faites votre analyse personnelle selon ce que vous observez dans ces tests (la complexité de l'algorithme, le taux de succès, …).
6. (Bonus) Vous êtes encouragés d'implanter d'autres heuristiques plus performantes. Selon votre réalisation, un bonus allant jusqu'à 2 points sur 10 pourrait être accordé selon la créativité que vous manifestez. Votre nouvelle heuristique doit avoir un taux de réussite nettement plus élevé et un nombre de nœuds explorés plus faible.

À rendre

Vous avez deux parties à rendre :

1. Un rapport contenant votre formulation du problème, une brève description de votre implantation et une analyse des fonctionnements sur les 100 exemples. Indiquez dans votre rapport également les programmes que vous rendez, et une explication sommaire des programmes (une ou deux phrases/programme).
2. Vos programmes pour l'application Sudoku doivent être incorporés correctement avec le package Java ou Python que vous utilisez. Indiquez dans votre rapport comment vos programmes doivent fonctionner avec le package. Les programmes que vous rendez doivent s'exécuter correctement sans qu'on doive faire des manipulations sur le package. Au besoin (si vous faites des modifications dans le package), vous pouvez aussi remettre le package modifié.
3. Ce TP est à faire en groupe de 2 personnes. Vous pouvez utiliser Python ou Java comme langage de programmation.

Gare au plagiat :

- Il existe beaucoup de programmes de Sudoku sur le Web, qui ne correspondent pas exactement à ce qu'on vous demande de faire. Si vous rendez un tel programme pris directement sur le Web, vous aurez la note de 0.
- Si 2 groupes rendent les mêmes programmes (ou les mêmes programmes masqués), vous aurez 0.

Date de remise :

La date limite pour la remise est le 9 octobre avant 23:59 en faisant une remise électronique : remise ift3335 TP1 <vos programmes>. Voir le document explicatif sur le web du cours si vous n'êtes pas familier avec ce programme de remise. Vous pouvez

remettre plusieurs fois, et la nouvelle remise remplacera l'ancienne si le nom du programme est identique. Votre rapport peut être remis de la même manière, ou bien par courrier électronique au dift3335@iro.umontreal.ca
Ce TP compte pour 10% dans la note globale. Chaque jour de retard entraîne 2 points de pénalité (sur 10 points).

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 concatenées. 0 signifie que la case et vide (à remplir par votre programme).
- Prof. Gordon Royle fournit un grand nombre de configurations de départ ici : http://staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php.
- Vous pouvez pratiquer en ligne ici : http://www.websudoku.com