TP1 – IFT3335
Jeu de Sudoku
(à rendre au plus tard le 17 mars 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 implanter 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 locale avec recuite simulé (simulated anealing)
- algorithme best-first

Ces algorithmes sont déjà implantés dans le package en Java et en Python qui accompagne le livre de référence (Russell & Norvig). Vous pouvez les adapter en faisant

des modifications nécessaires (voir ci-après), et faire fonctionner ces programmes pour ce problème Sudoku.

3.1. Modifications et ajouts à faire dans les programmes existants (sur le site du livre):

- Ajouter un comptage dans ces algorithmes pour compter le nombre de nœuds explorés. Ceci vous permettra de faire des analyses sur la performance.
- 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. (15%) (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%) Utiliser l'algorithme de profondeur d'abord. Dans cet algorithme, à chaque tour, on choisit une case à remplir par un chiffre possible. Vous pouvez utiliser une stratégie quelconque pour ordonner les cases vides et en choisir une à explorer.

3. (20%) Utiliser 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. Ceci va probablement engendre des conflits sur une ligne ou dans une colonne. Ensuite, on utilise Hill-Climbing pour tenter d'améliorer la configuration le plus possible, en inter-changeant (swap) deux des chiffres remplis dans un carré. L'amélioration consiste à réduire le nombre de conflit global (sur les lignes et dans les colonnes), ou (de façon équivalente) d'augmenter les nombres de chiffres différents sur chaque ligne et chaque colonne. Pour définir cette valeur heuristique, vous pouvez vous référer à un article par Lewis : Metaheuristics can solve sudoku puzzles. Cet article est affiché sur Studium.

4. (20%) Utiliser le recuit simulé. Hill-Climbing peut être coincé sur un optimum local sans pouvoir arranger tous les chiffres correctement. On peut améliorer cet algorithme en utilisant le recuit simulé (simulated anealing). Cet algorithme est décrit dans le cours, et aussi dans l'article de Lewis. Vous êtes demandés à implanter une version simplifiée de l'algorithme décrit dans 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 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.

5. (15%) Utiliser l'algorithme Best-first avec f=h (best-first vorace) qui tente de mettre un chiffre dans une case vide à chaque fois, sans violer les contraintes. Ici, vous êtes invité à utiliser vos imaginations et essayer d'implanter des heuristiques nouvelles. Une première heuristique simple est la suivante : à partir d'une configuration, on détermine les chiffres qu'on peut mettre dans chaque case sans violer les règles. Une case est plus contraignante si elle peut accepter moins de chiffres. On va alors choisir à remplir un chiffre dans la case la plus contraignante d'abord. La fonction heuristique (h) pour une configuration candidate est alors définie comme le nombre de chiffres possibles qu'on peut mettre dans la case choisie. Par exemple, si une configuration candidate est produite en plaçant un chiffre dans une case, cette configuration aura la valeur heuristique correspondant au nombre de différents chiffres possibles qui pouvaient être mis dans la case. Par rapport à un placement aléatoire, l'avantage de tenter 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.
   Il y a d'autres heuristiques décrits par Angus Johnson dans http ://www.angusj.com/sudoku/hints.php. Vous êtes encouragé à implanter plus d'heuristiques.

6. (10%) Comparer les algorithmes en les faisant fonctionner sur 100 configurations de départ (voir la liste des configurations sur Studium). 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). Faites votre analyse personnelle selon ce que vous observez dans ces tests (la complexité de l'algorithme, le taux de succès, …).

À rendre

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

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 le programme correspondant à chaque algorithme que vous avez implanté.
2. Indiquez dans votre rapport comment vos programmes doivent fonctionner. 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 du livre), 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 17 mars avant 23:59. Vous devez remettre le rapport et les programmes sur Studium.
Ce TP compte pour 15% dans la note globale. Chaque jour de retard entraîne 2 points de pénalité (sur 15 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