UNIVERSITÉ LIBRE DE BRUXELLES

INFO-F-302 Informatique Fondamentale

Projet : Problèmes de Satisfactions de Contraintes et Utilisation de l'Outil ChocoSolver

Le premier objectif de ce projet est de modéliser divers problèmes en problèmes de satisfaction de contraintes (CSP). Le deuxième objectif est d'implémenter un programme résolvant ces problèmes en utilisant ChocoSolver, un outil de résolution de contraintes basé sur JAVA avec lequel vous allez être amenés à vous familiariser par vous-même. Le rapport, présentant avec précision vos réponses aux différentes questions, ainsi que votre code source, devront être rendus pour le **dimanche 21 mai**.

1 L'Outil ChocoSolver

ChocoSolver est un programme permettant de décider les CSP. Si le problème admet une solution, une interprétation des différentes variables satisfaisant les contraintes est retournée.

`http://www.choco-solver.org/`

Un tutoriel en expliquant l'utilisation est disponible en ligne. Les diverses questions abordées dans ce projet correspondent à des variations du problème des huit dames.

`http://choco-tuto.readthedocs.io/en/latest/`

2 Problèmes d'échecs

Soient n, k_1, k_2 et k_3 des entiers positifs. Considérons un échiquier de dimensions $n \times n$. Nous avons à disposition k_1 tours, k_2 fous et k_3 cavaliers.

— Le problème d'*indépendance* consiste à déterminer s'il est possible d'assigner à chacune des pièce une position distincte sur l'échiquier de sorte qu'aucune pièce ne menace une autre pièce.

— Le problème de *domination* consiste à déterminer s'il est possible d'assigner à chacune des pièce une position distincte sur l'échiquier de sorte que chaque case soit occupée ou menacée par au moins une pièce.

Exemple 1 Supposons que l'on vous donne $n = 3$, $k_1 = 1$, $k_2 = 1$ et $k_3 = 1$. Il existe alors une assignation valide pour le problème d'indépendance, illustrée par la figure ci-contre. Toutefois, le problème de domination n'admet pas de solution.

F	*	*
C	*	*
*	T	*

Exemple 2 Supposons que l'on vous donne $n = 3$, $k_1 = 2$, $k_2 = 1$ et $k_3 = 1$. Il existe alors une assignation valide pour le problème de domination, illustrée par la figure ci-contre. Toutefois, le problème d'indépendance n'admet pas de solution.

T	*	F
*	T	*
*	*	C

Question 1. Exprimer une instance quelconque du problème d'indépendance (i.e. la donnée de $I = (n, k_1, k_2, k_3)$) par un CSP équivalent, en exposant de manière claire les variables, leurs domaines respectifs, et les contraintes à respecter.

Question 2. Exprimer une instance quelconque du problème de domination (i.e. la donnée de $I = (n, k_1, k_2, k_3)$) par un CSP équivalent, en exposant de manière claire les variables, leurs domaines respectifs, et les contraintes à respecter.

Question 3. Implémenter un programme qui, étant donné une instance de l'un des deux problèmes, en calcule une solution à l'aide de ChocoSolver. Pour faciliter le test de vos outils, nous fixons un format d'entrée et de sortie. Pour l'entrée, les options doivent être passées en ligne de commande : `-i` pour le problème d'indépendance, ou `-d` pour le problème de domination, puis la taille de l'échiquier (`-n` suivi d'un entier), le nombre de tours (`-t`), de fous (`-f`) et de cavaliers (`-c`). Par exemple `-i -n 4 -t 2 -f 1 -c 2` correspond à l'instance du problème d'indépendance définie par $n = 4$, $k_1 = 2$, $k_2 = 1$ et $k_3 = 2$. S'il n'existe aucun assignement valide des variables, votre programme affiche "pas de solution". Sinon, la sortie est un tableau composé de n lignes de n caractères séparés par des espaces, qui représente une solution. Les cases vides sont désignées par des étoiles (*), quant aux cases occupées, elles portent l'initiale de la pièce correspondante (T(our), F(ou), C(avalier)). Pour l'exemple, une sortie admissible est

```
T * * *
* T * *
* * C *
* * F C
```

Question Bonus. De manière générale, un type de pièce peut être modélisé comme une fonction associant à chaque position de l'échiquier l'ensemble des cases menacées par cette pièce depuis cette position. Par exemple, la tour peut être définie comme la fonction associant à toute position (i, j) l'ensemble des positions (i', j') telles que $i = i'$ ou $j = j'$. Généraliser le programme de la question 3 de manière à considérer des pièces quelconques.

Question 4. Implémenter un programme qui, étant donné un entier n, calcule le nombre minimal de cavaliers permettant de dominer un échiquier de taille n. La sortie produite doit être composée d'un nombre, correspondant à la solution du problème, suivi d'un tableau la représentant. Par exemple, si l'entrée est `4`, la sortie attendue est

```
4
* * * *
* C C *
* C C *
* * * *
```

Référez vous au tutoriel pour apprendre à minimiser des quantités avec ChocoSolver.

3 Surveillance de musée

Dans un musée, on souhaite positionner des capteurs laser afin de protéger de précieuses œuvres. Le coût de tels capteurs étant particulièrement élevé, il est souhaitable d'en minimiser le nombre, tout en assurant une surveillance absolue de tout l'espace vide.

Question Exprimer le problème par un CSP, et implémenter un programme le resolvant à l'aide de CHOCOSOLVER.

L'entrée attendue est un fichier contenant un tableau rectangulaire représentant la topologie de la salle à surveiller. Chaque case est vide, ou occupée par un obstacle (*). On suppose que les quatre murs sont toujours des obstacles.

La sortie attendue est composée d'un nombre, correspondant au nombre minimal de capteurs nécessaires, suivi d'un tableau, représentant une solution. Seules les cases vides peuvent contenir un (et un seul) capteur. Chaque capteur pointe dans une direction (N(ord), S(ud), E(st) ou O(uest)), et permet la surveillance de toutes les cases comprises entre sa position et le premier obstacle dans la direction correspondante. Notez bien qu'un capteur constitue un obstacle.

Voici un exemple d'entrée, suivi par une sortie admissible.

```
* * * * * * * * * *
*   * * *         *
*                 *
*   * *   * *     *
*   * *   * *     *
*       * *       *
* * * * * * * * * *
```

```
6
* * * * * * * * * *
* S * * * E       *
*                 O *
*   * *   * *     *
*   * *   * *     *
* E     N * * N *
* * * * * * * * * *
```