Master Informatique 2020-2021
M1 ANDROIDE – Résolution de Problèmes

TME 1 – Résolution de CSPs

1 Introduction au package constraint

Le module constraint de Python permet de résoudre des CSPs sur domaines finis (["red", "blue", ..., "green"], [1, 2, ..., n], ...) avec différents algorithmes (Backtrack, Recursive backtracking, Minimum conflicts solver, ...). (Voir la documentation de la librairie : http://labix.org/doc/constraint/).

Installation

- Le module constraint de Python est installé en salles ppti 14-15/408 (accessibe en SSH via par exemple "ssh -X ppti-14-408-04"). Attention le module est installé pour Python2.7.
- Pour une installation personnelle : pip install python-constraint
- Pour utiliser le module : from constraint import *

Définition et résolution d'un CSP

Ci-dessous quelques fonctions utiles à la définition et à la résolution d'un CSP:

- Problem : Classe permettant de définir un CSP.
- addVariable : Ajout d'une variable au modèle. Elle se définit par son nom et son domaine de la manière suivante : pb.addVariable(nom, domaine). Par exemple: pb.addVariable("a", [1,2,3])
- addConstraint : Ajout d'une contrainte au modèle. Elle peut être définie par une fonction que l'on définit, par exemple : pb.addConstraint(lambda a, b: a*2 == b, ("a", "b")). Ou en utilisant une contrainte prédéfinie, par exemple : pb.addConstraint(AllDifferentConstraint()) pour indiquer que la contrainte AllDiff concerne toutes les variables du problème ou encore pb.addConstraint(AllDifferentConstraint(), [a, b, c]) pour indiquer que la contrainte concerne les trois variables a, b et c.
- getSolution : Résolution et récupération d'une solution possible.
- getSolutions : Résolution et récupération de toutes les solutions possibles sous la forme d'un dictionnaire.

Exemple 1 : Problème des 8 tours (nombre maximum de tours sans attaque mutuelle sur un échéquier)

```
from constraint import *

# Dimension du problème
n = 8

# Création du problème
pb = Problem()

# Création d'une liste python cols de dimension n
(numéro de colonnes associé aux tours (une tour par ligne))
cols = range(n)

# Ajout de cols dans le problème. Chaque élément de cols a pour domaine {1, ..., n}
```

```
pb.addVariables(cols, range(n))

# Ajout de la contrainte AllDiff
pb.addConstraint(AllDifferentConstraint())

# Récupération d'une solution
s = pb.getSolution()

# Récupération de l'ensemble des solutions possibles
s = pb.getSolutions()

# Affichage
print("Nombre de solutions = ", len(s))
print(s)
```

Exemple 2 : Problème du carré magique

```
from constraint import *

# Dimension du problème
n = 3

# Création du problème
pb = Problem()

# Création d'une liste python x représentant les n^2 nombres à placer dans la grille
x = range(1, n**2 + 1)

# Ajout de x dans le problème. Chaque élément de x a pour domaine {1, ..., n**2 + 1}
pb.addVariables(x, x)

# Ajout de la contrainte AllDiff
pb.addConstraint(AllDifferentConstraint())

# Variable contenant la somme de chaque ligne/colonne/diagonale
s = n * (n**2 + 1) / 2

# Ajout des contraintes du carré magique
for k in range(n):
    # ligne k
    pb.addConstraint(ExactSumConstraint(s), [x[k*n+i] for i in range(n)])
    # colonne k
    pb.addConstraint(ExactSumConstraint(s), [x[k+n*i] for i in range(n)])
    # première diagonale
    pb.addConstraint(ExactSumConstraint(s), [x[n*i+i] for i in range(n)])
    # deuxième diagonale
    pb.addConstraint(ExactSumConstraint(s), [x[(n-1)*i] for i in range(1, n+1)])

solutions = pb.getSolutions()
print("Nombre de solutions = ", len(solutions))
print (solutions[0])
```

Exercice 1 : Cryptogramme

$$\text{MOI} + \text{TOI} + \text{LUI} + \text{ELLE} = \text{NOUS}$$

Ce cryptogramme utilise neuf lettres différentes (M,T,L,E et N ne prennent pas la valeur 0). Il possède exactement 160 solutions.

Mais quelle est la plus grande valeur possible de NOUS ?

Remarque : vous devez utiliser une modélisation utilisant les retenues.

Exercice 2 : La grande sœur énervante

Sophie aime les grands nombres et adore effectuer les multiplications. Elle a multiplé 173 par un nombre à 5 chiffres et elle a obtenu un résultat à 7 chiffres. Mais Adelaïde, la grande sœur énervante, a effacé sept chiffres du calcul de sa sœur.

$$173 \times \_\_\_\_\_ = 2020\_2\_$$

Par quel nombre 173 est-il multiplié ?

Exercice 3 : Les chocolats

Vous souhaitez emballer des chocolats pour votre ami. Heureusement, vous travaillez dans une chocolaterie qui a beaucoup de restes de chocolat. Vous avez différents types de chocolat à votre disposition.
Votre objectif est de lui apporter les chocolats maximisant la quantité totale de sucre, que vous pouvez mettre dans votre sac, qui passent le contrôle de sécurité, et qui ne dépasseraient pas une certaine valeur nette pour laquelle vous iriez en prison si vous étiez pris.
Vous n'aurez pas de problème avec la sécurité si vous apportez moins de 3 kilos. Vous pouvez mettre 1 dm^3 de chocolats dans votre sac. Et vous n'irez pas en prison si vous prenez moins de 300 euros de chocolats.
Les données relatives aux 4 types de chocolats sont données ci-dessous :

Type	Poids (g)	Dimensions (cm)	Quantité de sucre (g)	Valeur (euro)
Chocolat A	100	$8 \times 2.5 \times 0.5$	20	8
Chocolat B	45	$7 \times 2 \times 0.5$	16	6.8
Chocolat C	10	$3 \times 2 \times 0.5$	9	4
Chocolat D	25	$3 \times 3 \times 0.5$	7	3

Déterminer le nombre optimal de chocolats de chaque type à mettre dans votre sac.

Exercice 4 : Solveur de Sudoku

Dans cet exercice, vous allez programmer un solveur de Sudoku (version classique 9×9). Vous allez lire le puzzle à partir d'un fichier JSON et trouver toutes les solutions pour ce puzzle particulier (en supposant que le puzzle a une solution).

Si vous avez oublié les règles de résolution du sudoku, les voici :

- Les cellules peuvent avoir des valeurs de 1 à 9
- Toutes les cellules d'une même ligne doivent avoir des valeurs différentes
- Toutes les cellules d'une même colonne doivent avoir des valeurs différentes
- Toutes les cellules d'un carré 3x3 (neuf au total) doivent avoir des valeurs différentes

Vous pourrez utiliser le fichier JSON suivant (également disponible sur Moodle) :

```
[[0, 9, 0, 7, 0, 0, 8, 6, 0],
 [0, 3, 1, 0, 0, 5, 0, 2, 0],
 [8, 0, 6, 0, 0, 0, 0, 0, 0],
 [0, 0, 7, 0, 5, 0, 0, 0, 6],
 [0, 0, 0, 3, 0, 7, 0, 0, 0],
 [5, 0, 0, 0, 1, 0, 7, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 0, 9],
 [0, 2, 0, 6, 0, 0, 0, 5, 0],
 [0, 5, 4, 0, 0, 8, 0, 7, 0]]
```