Master Informatique 2018-2019
M1 ANDROIDE – Résolution de Problèmes

TME 1 – Résolution de CSPs

1 Introduction au package constraint

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

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

```
# Dimension du problème
n = 8

# Création du problème
pb = Problem()

# Création d'une variable python de dimension n
cols = range(n)

# Création d'une variable cols dont le domaine est {1, ..., n}
pb.addVariables(cols, range(n))

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

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

Exemple 2 : Problème du carré magique de dimension 3

```
# Dimension du problème
n = 3

# Création du problème
p = Problem()

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

# Création d'une variable x dont le domaine est {1, ..., n**2 + 1}
p.addVariables(x, x)

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

# Variable contenant la somme de chaque ligne/colonne/diagonale
s = n**2 * (n**2 + 1) / 6

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

s = p.getSolution()
```

Exercice : Coloration de carte et CSP valués

On considère le problème P_1 qui consiste à colorer la Figure 1 en attribuant une couleur parmi ["rouge", "vert", "bleu"] à chacune des 4 zones $Z_i, i = 1, \ldots, 4$, de sorte qu'on ne puisse trouver deux zones voisines (c'est-à-dire ayant au moins une frontière en commun) de la même couleur.

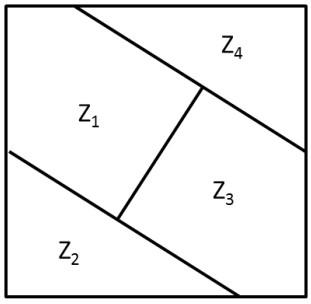

Figure 1: Carte du problème P_1

Question 1 – Dans un premier temps, on recherche une solution particulière telle que Z_1 soit en rouge ou en bleu, Z_2 et Z_3 soient en bleu ou en vert, et Z_4 soit en rouge. Écrire une fonction solve_p1 qui retourne

l'ensemble des solutions de ce problème. Que retourne cette fonction? Que peut on en déduire concernant la solution de P_1?

Question 2 – On considère maintenant le problème P_2 qui consiste à colorer la Figure 2 toujours en attribuant une couleur à chacune des 4 zones $Z_i, i = 1, \ldots, 4$, de sorte qu'on ne puisse trouver deux zones voisines de la même couleur. Cette fois on recherche une solution particulière telle que Z_1 soit en rouge ou en vert, Z_2 et Z_3 soient en bleu ou en vert, et Z_4 soit en vert. Écrire une fonction solve_p2 qui retourne l'ensemble des solutions de ce problème. Que retourne cette fonction? Que peut on en déduire concernant la solution de P_1?

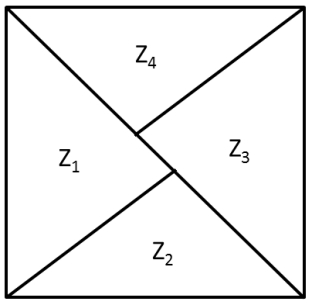

Figure 2: Carte du problème P_2

Question 3 — On considère maintenant le problème P qui consiste à colorer les deux figures 1 et 2 en autorisant les couleurs Bleu, Vert et Rouge pour chacune des zones et en tenant compte des préférences d'un agent sur ces couleurs. Les préférences de l'agent sont représentées par la fonction d'utilité donnée dans le tableau suivant (plus la valeur d'utilité est grande, plus la couleur est jugée adéquate pour la zone considérée):

	Z_1	Z_2	Z_3	Z_4
bleu	1	0.2	1	0.5
vert	0.3	0.6	0.7	1
rouge	0.8	1	0.4	0.9

En notant c_i la couleur attribuée à la zone i pour $i = 1, \ldots, 4$, on définit l'utilité d'une coloration partielle ou complète de la carte par $\min_{i \in I}\{u(c_i)\}$ où $I \subseteq \{1, \ldots, 4\}$ est l'ensemble des indices des zones déjà colorées.

3.1 - Coder un *branch and bound* avec *forward checking* permettant de déterminer une coloration complète d'utilité maximale (toujours sous la contrainte que deux zones voisines ne peuvent pas recevoir la même couleur).

3.2 - Quelle est la solution retournée par l'algorithme?

Question 4 — On définit maintenant l'utilité d'une coloration partielle ou complète de la carte par la fonction $f_T(\{u(c_i), i \in I\})$ définie par $f_T(\emptyset) = 1$ et $f_T(\{u\} \cup U) = T(u, f_T(U))$ où T est une fonction définie sur $[0,1] \times [0,1]$ à valeurs dans $[0,1]$ et vérifiant les 4 propriétés suivantes :

$$T(1,x) = x \quad (1)$$
$$T(x,y) = T(y,x) \quad (2)$$
$$T(x,T(y,z)) = T(T(x,y),z) \quad (3)$$
$$T(x,y) \leq T(z,w) \text{ si } x \leq z \text{ et } y \leq w \quad (4)$$

4.1 - Considérons c une coloration des zones $Z_i, i \in I$, et une extension de cette coloration colorant les zones $Z_i, i \in I'$ avec $I \subset I'$. Montrer que $f_T(\{u(c_i), i \in I\}) \geq f_T(\{u(c_i), i \in I'\})$.

4.2 - Proposer une variante de l'algorithme utilisé à la question précédente pour déterminer une coloration qui maximise f_T avec $T(x,y) = x \times y$. Quelle est la solution retournée?

4.3 Quel est l'intérêt d'utiliser cette fonction $T(x,y) = x \times y$ plutôt que $T(x,y) = \min\{x,y\}$?