Constraint Satisfaction Problem

Résolution de problèmes de satisfaction de contraintes avec Python

Othmane Elhamdouni & Imene Ouahrani

12/11/2020

Sommaire

1. *Introduction*

Nous proposons un solveur pour les problèmes de résolution de contraintes (CSP) sur des domaines finis en Python simple et pur. CSP est une classe de problèmes qui peuvent être représentés en termes de variables (a, b, ...), de domaines (a dans [1, 2, 3], ...) et de contraintes (a <b, ...).

2. *Fonctionnalités utilisées*

Nous avons utilisé les fonctions suivantes :

2.1. **Recherche avec retour sur trace** L'algorithme de recherche avec retour sur trace (en anglais ***backtracking search***) est utilisé pour trouver l'affectation de poids maximum d'un graphe de facteurs. À chaque étape, une variable non assignée est choisie et ses valeurs sont explorées par récursivité

2.2. **Vérification en avant La vérification en avant (*forward checking* en anglais)** est une heuristique d'anticipation à une étape qui enlève des variables voisines les valeurs impossibles de manière préemptive. Cette méthode a les caractéristiques suivantes :

- Après l'affectation d'une variable, les valeurs non consistantes sont éliminées du domaine de tous ses voisins.
- Si l'un de ces domaines devient vide, la recherche locale s'arrête.
- Si l'on enlève l'affectation d'une valeur, on doit restaurer le domaine de ses voisins.

2.3. **Variable la plus contrainte L'heuristique de la variable la plus contrainte (en anglais *most constrained variable* ou *MCV*)** sélectionne la prochaine variable sans affectation ayant le moins de valeurs consistantes. Cette procédure a pour effet de faire échouer les affectations impossibles plus tôt dans la recherche, permettant un élagage plus efficace.

2.4. **Valeur la moins contraignante L'heuristique de la valeur la moins contraignante (en anglais *least constrained value* ou *LCV*)** sélectionne pour une variable donnée la prochaine valeur maximisant le nombre de valeurs consistantes chez les variables voisines. De manière intuitive, on peut dire que cette procédure choisit en premier les valeurs qui sont le plus susceptible de marcher.

3. Bibliothèque utilisées

3.1. Le module re de la bibliothèque standards — Opérations à base d'expressions rationnelles : Code source : Lib/re.py

Le **re.sub ()** remplace les sous-chaînes qui correspondent au modèle de recherche par une chaîne au choix de l'utilisateur afin de parcourir les chaînes de caractère décrites dans le **« input file »** présent dans le dossier **« Test »** ligne par ligne, du début à la fin tout en ajoutant les contraintes présentes dans les **fichiers.con** à la liste des contraintes du code source.

3.2. **sys** — Paramètres et fonctions propres à des systèmes

Ce module fournit un accès à certaines variables utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier. Ce module est toujours disponible.

sys.argv renvoie une liste d'arguments de ligne de commande passés à un script Python. L'élément à l'index 0 dans cette liste est toujours le nom du script. Le reste des arguments est stocké dans les index suivants.

Dans notre script Python **(app.py)** consommant deux arguments de la ligne de commande.

Ce script est exécuté à partir de la ligne de commande comme suit:
capture

3.2.1. Exécution

Arguments de ligne de commande:

1. **Fichier .var** contenant une liste de variables.

2. **Fichier .con** qui contient une liste de contraintes.

3. procédure de mise en cohérence **"aucun"** pour le retour arrière uniquement et **"fc"** pour le retour arrière avec contrôle avant.

Exemples d'arguments :

python3 app.py Tests/ex1.**var** Tests/ex1.**con** Tests/**fc**

python3 app.py Tests/ex2.**var** Tests/ex2.**con aucun**

Comme nous pouvons le voir, **sys.argv [1]** contient le premier paramètre «Tests/ex1.**var**», tandis que **sys.argv [2]** contient le deuxième paramètre «Tests/ex1.**con**». **sys.argv [0]** contient le nom de fichier de script **app.py**.

3.3. copy — Opérations de copie superficielle et récursive : Code source : LIB/COPY.PY

Les instructions d'affectation en Python ne copient pas les objets, elles créent des liens entre la cible et l'objet.

3.4. Operator — Opérateurs standards autant que fonction : Code source: Lib/operator.py

Le module operator fournit un ensemble de fonctions correspondant aux opérateurs natifs de Python. Nous nous intéressons aux fonctions de comparaison s'appliquent à toutes les contraintes afin de choisir la variable la plus contraignante, et leur nom vient des opérateurs de comparaison qu'elles implémentent : **operator.gt, operator.lt, operator.eq, operator.ne**

Chaque variable a un **label (une étiquette)**, appartient à un **domaine** et a un **assignement (une valeur)**.

4. Fonctionnement

1. Le CSP choisit la variable en fonction de l'heuristique de variable la plus contrainte, rompant les liens en utilisant l'heuristique de variable la plus contraignante. L'algorithme peut trouver plusieurs solutions viables.

2. S'il reste plus d'une variable après l'application de ces heuristiques, rompre les liens par ordre alphabétique.

3. Le CSP choisit les valeurs en fonction de l'heuristique de valeur la moins contraignante.

4. Le dossier "test" contient 3 exemples de cas de test.

5. Lien GITHUB :

https://github.com/othmaneha25/Projet-IA-API-CSP

6. Conclusion

Ce projet s'est révélé très enrichissant dans la mesure où il a consisté en une approche concrète de l'intelligence artificielle, il nous a permis de mettre en place des connaissances acquises tout au long des cours et TDs, ainsi que des connaissances de programmation (PYTHON).

L'ensemble des fichiers du projet sont mise à disposition du public sur notre page GITHUB.