Résolution de problèmes, génération de mots-croisés

Androide M1 – RP

Thibault Gigant
Laura Greige

Enseignants :

Patrice Perny
Morgan Chopin

2015 – 2016

Table des matières

Introduction

Dans ce projet, on s'intéresse à compléter une grille de taille $M \times N$. Dans un premier temps, on modélisera ce problème comme un problème de satisfaction de contraintes et on développera plusieurs méthodes différentes de recherche d'une solution au problème CSP : arc consistance du graphes des contraintes ($AC3$), forward checking (FC), conflict back-jumping (CBJ). La grille peut être déjà partiellement remplie par l'utilisateur.

Dans un second temps, on modélisera ce problème comme un CSP valué. Chaque mot du dictionnaire sera muni d'un poids positif ou nul $\in [0, 1]$ et on développera un algorithme de type Branch and Bound qui permette de rechercher la solution optimale qui serait la solution de poids maximal dans la grille.

1 Modélisation par un CSP et résolution

1.1 Modélisation

Pour résoudre ce problème, on peut le modéliser comme un problème de satisfaction de contraintes en associant une variable à chaque mot de la grille. Supposons qu'il y ait m mots dans la grille.

Variables : x_i , $\forall i \in \{1, ..., m\}$

Domaines : Soit *dict* un dictionnaire de mots admissibles :

$$D(x_i) = \{X \in dict\}$$

Contraintes : Soit l_i la taille du mot en i :

$$len(x_i) = l_i \tag{1}$$

Pour tous mots x_i et x_j qui se croisent à la q-ième lettre de x_i et à la p-ième lettre de x_j, on a :

$$x_i[q] = x_j[p] \tag{2}$$

Si l'on ajoute la contrainte supplémentaire qu'un même mot ne peut apparaître plus d'une fois dans la grille, il suffit d'ajouter la contrainte *AllDiff* :

$$AllDiff\,(x_1, x_2, ...x_m) \tag{3}$$

1.2 Expérimentation

Application

Dans cette section, nous allons tester les performances de nos algorithmes, d'abord en utilisant RAC avec Forward Checking sans AC3 préalable, ensuite RAC avec Forward Checking et AC3 préalable.

	Grille A	Grille B	Grille C
AC3	t1	t2	t3
FC sans AC3 préalable	t1	t2	t3
FC avec AC3 préalable	t1	t2	t3

Conflict BackJumping

Applications sur des grilles de tailles plus grandes

La méthode ( ) étant la plus efficace, on l'applique sur des instances de grilles de tailles plus grandes.

	Temps d'éxecution	% de solution trouvée
Taille = 10	t1	%
Taille = 100	t2	%

2 Extension au cas pondéré

2.1 Modélisation

Petite intro.

Variables : x_i , $\forall i \in \{1, ..., m\}$

Domaines : Soit *dict* un dictionnaire de mots admissibles :

$$D(x_i) = \{X \in dict\}$$

Contraintes : Soit l_i la taille du mot en i :

$$len(x_i) = l_i \tag{4}$$

Pour tous mots x_i et x_j qui se croisent à la q-ième lettre de x_i et à la p-ième lettre de x_j, on a :

$$x_i[q] = x_j[p] \tag{5}$$

Si l'on ajoute la contrainte supplémentaire qu'un même mot ne peut apparaître plus d'une fois dans la grille, il suffit d'ajouter la contrainte *AllDiff* :

$$AllDiff\,(x_1, x_2, ...x_m) \tag{6}$$

2.2 Expérimentation

2.3 Bonus

Conclusion