M1 ALMA
Université de Nantes
2011-2012

Programmation par contraintes : Gestion d'un cursus universitaire

MARGUERITE Alain
RINCE Romain

Université de Nantes
2 rue de la Houssinière, BP92208, F-44322 Nantes cedex 03, FRANCE

Table des matières

1 Brève explication sur le lancement d'un programme

Aucun des programmes ne contient de prompt d'interrogation pour obtenir le fichier de spécification du problème. Il est nécessaire que l'utilisateur ajoute en paramètre l'adresse du fichier lors de l'exécution de la commande bash de la forme `python <NOM_DU_PROG> <FICHIER_DE_SPEC>`. Par exemple :

```
python curicullum1a.py data8
```

Le programme affichera en sortie les différents modules associées à un même semestre sur une seule ligne et au bout de celle-ci la valeur en ECTS du semestre. Chaque ligne représente donc la totalité d'un semestre, ceux ci étant afficher dans l'ordre temporel du premier au dernier. A la suite des semestre s'afficheront des informations complémentaires sur le temps de recherche, le nombre de branches et le nombre d'échec.

2 Présentation du modèle pour la résolution

L'ensemble des questions ont été résolues en utilisant quasiment le même modèle. Nous avons donc choisi de représenter les différentes affectations des modules à un semestre par une matrice de booléens Mat. Les lignes de la matrices sont les différents semestres tandis que chaque colonne représente un module ; ainsi la valeur vrai à la case$[i][j]$ signifie que le module en j$^{\text{ème}}$ position dans la liste des modules est associé au i$^{\text{ème}}$ semestre. Chaque case de la matrice sera donc une variable du problème.

2.1 Réponse au problème 1a : Contraintes globales et réifiées

Pour résoudre le problème les contraintes suivantes ont été ajoutées :

- On vérifie qu'un module ne peut apparaître qu'une seule fois dans l'ensemble des semestres. Pour cela on utilise un `Count` sur chaque colonne de la matrice solution en vérifiant qu'il n'y ait qu'une seule variable à vrai.

$$\forall i : \sum_{j=1}^{nbModules} Mat[i][j] = 1 \tag{1}$$

.

- On vérifie que le nombre de modules dans un semestre est bien comprise entre le nombre maximum et minimum de matière possible dans un semestre grâce à la contrainte globale `Between`.

$$\forall i : minM \leq \sum_{j=1}^{nbModules} Mat[i][j] \geq maxM \tag{2}$$

.

– On réalise la même vérification sur le nombre d'ECTS par semestre en effectuant au préalable un produit scalaire entre chaque ligne de la matrice et la liste qui fournit le nombre d'ECTS pour chaque module.

$$\forall i : minC \leq \sum_{j=1}^{nbModules} Mat[i][j] \times ECTS[j] \geq maxC \tag{3}$$

– Enfin on vérifie que l'on respecte toutes les règles de précédences fournies en récupérant les semestres associés aux modules de la règle puis en comparant leurs valeurs grâce à un `IsGreaterCt`. L'obtention du numéro du semestre n'est pas triviale et consiste à effectuer le produit scalaire entre la colonne du module dont l'on recherche son semestre et un tableau contenant les valeurs, triées de façon croissante, de 1 au nombre de semestre total.

Les paramètres de la recherche sont l'affectation des variables dans l'ordre dans lesquelles elles ont été fournies et l'on leurs assigne la valeur maximale en premier c'est-à-dire vrai [1].

2.2 Réponse au problème 1b : Contraintes redondantes

Pour ce problème, nous avons ajouté des variables sur le nombre d'ECTS par semestre et le nombre de matière par semestre. Les contraintes sur le nombre d'ECTS par semestre (3) et le nombre de matière par semestre (2) n'existe donc plus puisque celle-ci sont implicitement données par le domaines des variables. Ces variables devront évidemment être égales au nombre de matière/ECTS de leur semestre associé.

Les contraintes redondantes ajoutées sont les suivantes :

– On vérifie que la somme des variables sur le nombre de matière et d'ECTS soient égales respectivement au total de matière et au total d'ECTS alloués dans l'ensemble des semestres.

– La seconde contrainte ajoutée est un peu plus complexe puisqu'il s'agit d'effectuer des vérifications en plus pour les règles de précédences. Ainsi pour deux modules positionnés respectivement à l'indice a et b dans la liste des modules, le module a précédant b, on récupère les colonnes associées dans la matrice solution et l'on vérifie la propriété suivante :

$$\forall i \in [1; nbModules] : Mat[i][a] \leq \sum_{j=i+1}^{nbModules} Mat[j][b] \tag{4}$$

Cette contrainte à notamment l'avantage de ne pas avoir toujours besoin de savoir précisément le semestre associé à un module pour savoir si la contrainte de précédence n'est pas respectée ; il suffit parfois de

1. Les valeurs étant 0 pour faux et 1 pour vrai

connaître un certain nombre de semestre dans lesquels la matière n'est pas présente. On peut d'ailleurs noté différents *symétriques* de cette contrainte qui pourrait la compléter tel que :

$$\forall i \in [1; nbModules] : Mat[i][b] \geq \sum_{j=i+1}^{nbModules} Mat[j][a]$$

Il aurait aussi été possible de faire la somme de ces contraintes en partant de l'indice 1 jusqu'au $i^{\text{ème}} - 1$.

Les paramètres de recherche de solution sont les mêmes que pour le problème 1a (section 2.1)

2.3 Réponse au problème 2a : Contraintes linéaires

La transformation du problème 1a en utilisant uniquement les contraintes linéaires est relativement simple puisqu'il suffit de remplacer chacune des contraintes par un équivalent linéaire. Ainsi la contrainte globale `Count` est remplacée par une simple somme et `Between` par deux contraintes linéaires, l'une vérifiant la supériorité et l'autre l'infériorité.

Il faut cependant noter que la recherche de solution n'est plus la même puisque les valeurs attribuées sont en premier les plus faible (soit 0)

2.4 Réponse au problème 2b : GLPK

Le problème 2b n'a malheureusement pas été traité par manque de temps. Le modèle serait cependant resté similaire à la version 2a mais avec la nécessité de calculer le produit scalaire soi-même, GLPK ne fournissant pas la méthode. Il aurait aussi été nécessaire d'ajouter des cast entier, GLPK ne renvoyant que des doubles.

2.5 Réponse au problème 1c : Objectif

Le problème consistant à minimiser le maximum d'ECTS par semestre est relativement simple à résoudre puisqu'il suffit d'ajouter comme objectif la minimisation du nombre d'ECTS pour le semestre ayant le plus. Pour cela nous avons la plus grande des variables sur le nombre d'ECTS dans un semestre définies dans la section 2.2.

3 Étude sur les résultats obtenus

3.1 Résultats

Tout au long de la réalisation de ce projet, nous avons eu de nombreux résultats souvent très différents et discordants avec nos suppositions. Les

modèles représentés dans les sections précédentes sont les modèles choisis parmi bien d'autres mais étant ceux qui offrait les meilleurs résultats.

On pouvait donc observer dans le problème 1a un temps de recherche proche des deux minutes et plus d'un million de branches pour trouver le premier résultat.

Le problème 1b n'arrivait jamais à trouver la première solution. Des modèles très proches du 1a ont été testés en ajoutant seulement une des contraintes redondantes mais aucun résultat n'était retourné après un quart d'heure.

Le problème 2a est le plus rapide puisqu'il se résout en moins d'une seconde avec environ trois cent branches.

Le problème 1c est lui aussi assez rapide et se résout en quelques secondes.

3.2 Analyse

Il est plutôt étrange de constater que l'ajout de contraintes redondantes nuit, de façon non négligeable, à la recherche et ce malgré tous nos efforts.

L'efficacité de la recherche linéaire peut sans doute s'expliquer au niveau de l'implémentation des contraintes globales mais au vu des résultats des autres méthodes, nous ne saurions dire s'ils sont pertinents. De plus nous avons constater que malgré les similarités entre le programme 1a et 2a, la résolution linéaire est plus efficace si l'on affecte en priorité la valeur minimale tandis que pour la méthode 1a c'est l'affectation des valeurs maximales en priorité qui est la plus efficace [2].

Au vu des résultats et de l'extraordinaire panel d'avis et de suppositions des différents binômes, je pense que l'on peut conclure sans rougir que nous ne maîtrisons pas totalement les différents aspects de la programmation par contraintes.

2. L'attribution des valeurs minimales en priorité ne nous retournant aucun résultat après une heure de recherche