VIỆN QUỐC TẾ PHÁP NGỮ
Institut Francophone International

RAPPORT DE TP2 PROGRAMMATION PAR CONTRAINTE

SUJET : Résoudre le Problème d'apprentissage académique équilibré avec la librairie Choco

Etudiant :

KAFANDO Rodrique

Professeur:

Dung Pham Quang

Promotion 21

INTRODUCTION

Dans la vie courante, nous sommes constamment soumis ou confronté à des problèmes d'ordre organisationnel en fonction du temps, des moyens à utiliser. De façon générale il s'agit de résoudre des problèmes complexes, des problèmes soumis à une diversité de contraintes. Autrement dit, une contrainte est une relation logique établie entre différentes variables, chacune prenant sa valeur dans un ensemble qui lui est propre, appelé domaine.

Dans cette logique qu'il nous a été proposé dans le cadre du cours intitulé programmation par contrainte de résoudre un problème d'apprentissage académique équilibré étendu.

Dans les lignes qui suivent, nous aborderons un peu la problématique des CSP (Constraint Satisfaction Programming), ensuite nous expliciterons le problème posé, troisièmement nous donnerons le modèle mathématique du problème, quatrièmement nous donnerons le modèle choco, et nous terminerons par la présentation des résultats obtenus.

I) PROBLÉMATIQUE

Le principe du résolution de contraintes par choco peut être résumé en deux grands points essentiels à savoir la définition du modèle **Model** et celle du solveur **solver** qui permet de résoudre le modèle prédéfini.

I-1) Le modèle

Le Model, accompagné du Solver, est l'un des éléments fondamentaux de choco. Il permet de décrire simplement un problème de manière déclarative. Son rôle principal est d'enregistrer les variables et contraintes d'un modèle.

I-1-1) Les variables

Une variable est définie par son type (entier, réel, ensemble), un nom et les valeurs de son domaine. On peut spécifier certaines options pour imposer une représentation du domaine dans le Solver, par exemple un domaine borné ou énuméré pour une variable entière. Un choix judicieux des types des domaines en fonction de leur taille et des contraintes du modèle portant sur leur variable a une influence critique sur l'efficacité du solveur.

I-1-2) Les contraintes

Une contrainte porte sur une ou plusieurs variables et restreint les valeurs qu'elles peuvent prendre simultanément en fonction d'une relation logique. Le scope d'une contrainte simple contient un nombre prédéterminé de variables contrairement à celui des contraintes globales. Une contrainte globale représente un sous-problème pour lequel elle réalise généralement des réductions de domaine supplémentaires par rapport à un modèle décomposé logiquement équivalent.

I-2) Le solveur

Les fonctionnalités principales offertes par l'interface Solver sont la lecture du Model et la configuration d'un algorithme de recherche.

I-2-1) La lecture du modèle

La lecture du modèle se fait une et une seule fois après sa définition complète. Elle peut être décomposée en deux étapes, la lecture des variables et la lecture des contraintes. La création d'un Solver PPC (Programmation Par Contrainte) et la lecture d'un modèle sont réalisées de la manière suivante.

I-2-1-1) La lecture des variables

Le solveur parcourt les variables du modèle et construit les variables et domaines spécifiques du solveur. Le solveur utilise trois types de variables simples, les variables entières (IntVar), ensemblistes (SetVar) et réelles (RealVar). Le type de domaine est déterminé en fonction des options ou automatiquement. Les variables du modèle et du solveur sont des entités distinctes. Les variables du solveur sont une représentation des variables du modèle. Ainsi, la notion de valeur d'une variable est définie uniquement dans le solveur

I-2-1-2) La lecture des contraintes

Après la lecture des variables, le solveur parcourt les contraintes du modèle pour créer les contraintes du solveur qui encapsulent des algorithmes de filtrage réagissant à différents événements sur les variables (modifications des domaines) ou du solveur (propagation). Les contraintes du solveur sont ajoutées à son réseau de contraintes.

I-2-2) Les stratégies de branchement

En choco, la construction de l'arbre de recherche repose sur une séquence d'objets de branchement qui jouent le rôle des '*goals*' intermédiaires en programmation logique. L'utilisateur peut définir la séquence d'objets de branchement utilisée lors de la construction de l'arbre de recherche. Outre son large catalogue de contraintes, choco propose de nombreuses stratégies de branchement et heuristiques de sélection pour tous les types de variables simples (entière, tâche, réelle) et complexes (tâches).
Dans la figure ci-dessous, nous résumons le principe de résolution de des CSP.

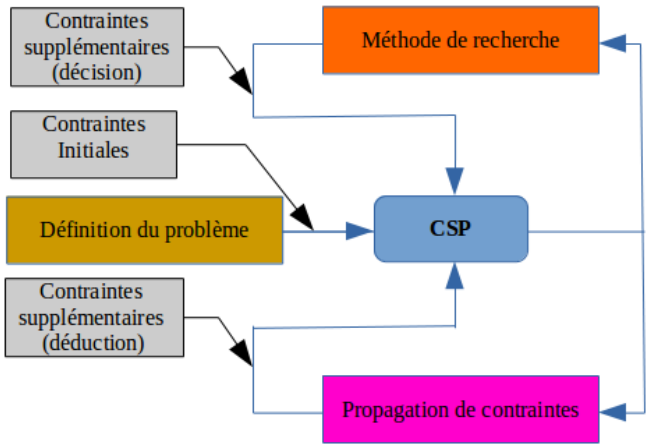

Figure1 : Principe de résolution CSP

II) DÉFINITION DU PROBLÈME

Ce TP s'inscrit dans le cadre du cours de programmation par contrainte, dans le quel nous sommes amené à résoudre un certain nombre de problèmes qui entrent dans la gestion d'un emploi du temps académique d'une université.

En effet, il y a un nombre de cours **n** que l'on doit attribuer dans une séquence de **n** semestres. Chaque cours possédant un nombre fini de crédit **C(i)**.

Le but de ce TP est de résoudre le problème posé ci-dessus tout en tenant compte des contraintes posées ci-dessous.

contrainte 1 : le nombre de crédits de cours de chaque semestre doit être supérieure ou égale à A et inférieure ou égale à B ;

contrainte 2 : le nombre de cours attribués à chaque semestre doit être supérieure ou égale à C et inférieure ou égale à D ;

contrainte 3 : la contrainte sur la condition sine qua non ;

contrainte 4 : chaque enseignant enseigne au plus un cours par semestre ;

contrainte 5 : le nombre de cours assignés à chaque enseignant doit être inférieur ou égal à E.

Pour ce faire, nous allons utiliser la librairie open-source Choco précédemment définie. Elle est une bibliothèque java utilisée pour les problèmes de satisfaction des contraintes (CSP), la programmation des contraintes (CP) et résolution de contraintes basée sur l'explication (e-CP). Il est construit sur un mécanisme de propagation basé sur des événements avec structures reculées.

II-1) Modèle mathématique du problème

Les contraintes définies ci-dessus peuvent être représentes par des formules mathématiques. Tout d'abord, nous définissons quelques variables.

Soit :

nbCreditsParSemestre = nombre de crédits de cours par semestre ;

nbCoursParSemestre = nombre de cours par semestre ;

pCoursI= période du cours i ;

pCoursJ= période du cours j ;

nbCoursParEnseignantParSemestre = le nombre de cours enseigné par chaque enseignant dans un semestre ;

nbCoursTotalParEnseignant = nombre de cours total assignés à chaque enseignant.

Contrainte 1 :

$$minCredits \leq nbCreditsParSemestre \leq maxCredits$$

Contrainte 2 :

$$minCours \leq nbCoursParSemestre \leq minCours$$

Contrainte 3 :

$$pCoursI \leq pCoursJ$$

Contrainte 4 :

$$NbCoursParEnseignantParSemestre \leq 1$$

Contrainte 5 :

$$1 \leq \text{NbCoursTotalParEnseignant} \leq \texttt{nbCoursMaxParProf}$$

II-2) Modèle à CHOCO

Dans les lignes qui suivent nous définissons les modèles mathématiques précédemment définies sous forme de contraintes avec la librairie choco.

Contrainte 1 :

Soit **z** une liste contenant la valeur d'un cours dans chaque période donnée (oui=1, non=0).

```
m.addConstraint(Choco.leq(Choco.scalar(z,listCredits),maxCredits);
m.addConstraint(Choco.leq(minCredits,Choco.scalar(z,listCredits));
```

Contrainte 2 :

La somme des cours dans une période doit etre au plus maxCours et au moins minCours.

```
m.addConstraint(Choco.leq(Choco.sum(z), maxCours));
m.addConstraint(Choco.leq(minCours, Choco.sum(z)));
```

Contrainte 3 :

Soient en déffinissant I[l] comme la liste des cours qui doivent etre enseignées premierement et J[l] la liste des cours qui doivent etre dispensées en seconde position.
On a :

```
m.addConstraint(Choco.lt(x[I[l]],x[J[l]]));
```

Contrainte 4 :

Nous verifions tout d'abord la présence d'un éventuel cours dans l'ensemble des périodes à travers la création de la contrainte **c.** Par la suite, nous imposons la contrainte de tel sorte que le meme enseignant n'intervienne qu'au plus une seule fois dans chaque période.

```
Constraint c = Choco.and(Choco.eq(x[i],p), Choco.eq(x[j], p));
m.addConstraint(Choco.implies(c, Choco.neq(x2[i], x2[j])));
```

Contrainte 5 :

Nous considerons que dans chaque période, aumoins un enseignant donne un cours, et nous limitons le nombre maximum de cours dans une période par nbCoursMaxParProf.

```
m.addConstraint(Choco.leq(Choco.sum(z2), nbCoursMaxParProf));
m.addConstraint(Choco.leq(1, Choco.sum(z2)));
```

III-) Tableau présentant les résultats expérimentaux

Dans le but de pouvoir évaluer la pertinence de notre programme, nous avons effectué différents tests sur des données différente et les résultats obtenus sont les suivants. Nous avons les périodes P0, P1, P2, P3. Dans chaque période nous donnons les enseignants (Esgnt), les cours et les crédits des cours concernés.

		ex-bacp.inp temps=1270.0ms				ex-bacp-c20-p5-t5.inp temps=2289.0ms					ex-bacp-c25-p5-t5.inp temps=3197.0ms					ex-bacp-c27-p3-t10.inp temps=3013.0ms							
P0	Esgnt	0	1	2	3	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	5	6	
	Cours	0	3	11	12	0	1	4	5	13	0	1	6	7	14	0	1	6	7	12	13	14	
	Crédit	2	2	3	3	2	2	2	2	2	2	1	2	3	3	2	1	2	3	3	2	3	
P1	Esgnt	0	1	2	3	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	5		
	Cours	1	2	6	9	2	6	8	12	15	2	3	4	12	18	2	3	4	15	18	21		
	Crédit	1	2	2	2	2	2	2	2	2	2	2	1	3	2	2	2	1	3	2	2		
P2	Esgnt	0	1	2		0	1	2			0	1	2	3	4	0	1	2	3	4	5		
	Cours	4	8	10		3	7	19			5	9	11	15	21	5	9	11	16	25	26		
	Crédit	1	3	2		2	2	2			2	2	3	3	2	2	2	3	2	2	3		
P3	Esgnt	0	1			0	1	2	3		0	1	2	3	4	0	1	2	3	4	5	6	7
	Cours	5	7			9	10	14	17		8	10	19	20	22	8	10	17	19	20	22	23	24
	Crédit	2	3			2	2	2	2		3	2	1	1	3	3	2	3	1	1	3	3	2
P4	Esgnt					0	1	2			0	1	2	3	4								
	Cours					11	16	18			13	16	17	23	24								
	Crédit					2	2	2			2	2	3	3	2								

Tableau1 : Récapitulatif des résultats

A l'issu des expérimentation, les quatre fichiers fichiers de données ont donnés de résultats satisfaisants, c-à-d que le programme à fournie des résultats positifs tout en tenant compte des contraintes évoquées ci-dessus dans la limite du temps. Le plus long était de 3013.0ms.

CONCLUSION

La programmation par contraintes est un formidable paradigme déclaratif permettant la modélisation et la résolution de nombreux problèmes de complexités diverses. Un problème peut se modéliser en général de plusieurs façons, et le choix d'une modélisation plutôt qu'une autre peut avoir un impact important en temps de résolution par le solveur. Pour la résolution de notre problème nous avons utilisé la librairie choco qui nous a permis d'avoir des résultats satisfaisants dans l'ensemble des tests que nous avons effectué.