Set Constraint Modelling

Cyril Lepinette – Cyril Grelier – Romain Grelier – Etienne Tignon

Encadrant :
Éric Monfroy

Résumé

Rapport du projet du cours d'approches exactes de résolution de problème avec Éric Monfroy à l'université d'Angers. L'objectif du projet était l'étude de différentes modélisation des problèmes de Sport Tournament Scheduling[5] (STS) et Social Golfer Problem[2] (SGP) en contraintes ensemblistes, SAT et FD et d'implémenter un solveur de contraintes ensemblistes afin d'analyser l'importance des cassages de symétries et autres paramètres.

Nous vous présenterons le solveur ensembliste implémenté puis développerons le travail réalisé sur le STS puis sur le SGP. Pour ces deux modèles, nous détaillerons la modélisation sous différentes formes, l'amélioration de nos modèles et les tests effectués sur ceux-ci.

Table des matières

1 Solveur pour les contraintes ensemblistes

Notre solveur de contrainte ensembliste est développé en Python, le code est disponible à l'adresse `https://github.com/Cyril-Grl/Set_Constraint_Modeling`.

Dans la figure 1 vous trouverez un schéma du fonctionnement du solveur.

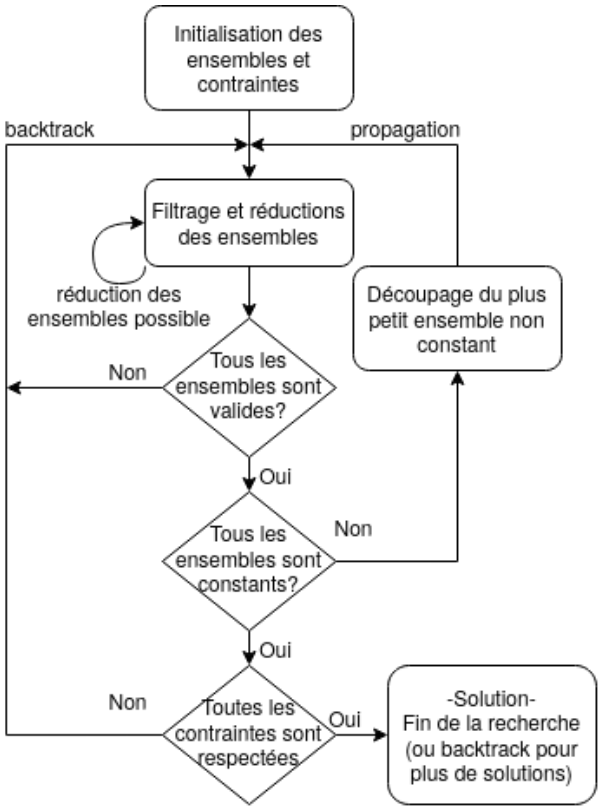

FIGURE 1 – Le fonctionnement du solveur est relativement simple. Les ensembles et contraintes sont tout d'abord initialisés, elles passent ensuite dans une première étape de filtrage. Pendant le filtrage, les ensembles sont réduits autant que possible. Arrive ensuite une étape durant laquelle les ensembles sont triés en fonction de la taille de leurs bornes supérieures et inférieures. Le plus petit ensemble sera donc découpé en sous-ensembles considérés comme "constants". Un ensemble constant ne peut plus être divisé en sous-ensembles. La recherche continue avec tous les sous-ensembles, tout d'abord avec un filtrage. À l'issue du filtrage, soit les ensembles ne sont pas valides, par exemple, la borne inférieure contient plus d'éléments que la supérieure, ou ne respectent pas une contrainte dans ce cas la recherche pour cette branche s'arrête et on backtrack en utilisant l'ensemble suivant. Sinon l'ensemble est valide, dans ce cas nous vérifions si tous les ensembles sont "constants". Si non, on découpe le plus petit et propage, sinon on vérifie les contraintes. Si les contraintes sont valides alors la recherche s'arrête dans le cas où on ne cherchait qu'une solution, dans tous les autres cas, on backtrack sur la valeur suivante du dernier ensemble traité.

En exécutant le fichier *main.py*, vous pouvez lancer la recherche sur le STS en utilisant la commande suivante :

```
python3 main.py -n 8 -s 1
```

Ou (beaucoup plus rapide pour $n >= 8$ mais demande d'installer pypy (`https://pypy.org/`) :

```
pypy3 main.py -n 8 -s 1
```

La recherche sur le STS se lancera avec 8 équipes en cherchant une seule solution ($-s$ 0 pour toutes les solutions).

Ensembles

Les ensembles possèdent un nom, une borne inférieure, une borne supérieure et la possibilité d'être constant ou non. Si un ensemble est constant alors ces bornes ne seront plus modifiés (fin de la recherche sur l'ensemble). Les ensembles se divisent sur l'ensemble de leur treillis durant la recherche (celui-ci n'étant pas généré entièrement mais petit à petit grâce au *yield* de Python). Un ensemble est valide si tous les éléments de sa borne inférieure sont présent dans la borne supérieure.

Contraintes

Voici la liste des contraintes disponibles dans le fichier $SCM/contraintes.py$:

- Contrainte sur deux ensembles :
  - $Different : var1 \neq var2$
  - $Egal : var1 = var2$
  - $Cardinalite : |var1| = var2$ (où $var2$ est un ensemble représentant un entier)
  - $Inclusion : var1 \subset var2$
  - $Exclusion : var1 \not\subset var2$
- Contraintes sur trois ensembles :
  - $Union : var1 = var2 \cup var3$
  - $Intersection : var1 = var2 \cap var3$
- Contraintes sur quatre ensembles :
  - $Intersection3 : var1 = var2 \cap var3 \cap var4$

Vous pouvez aussi créer de nouvelles contraintes en héritant des différentes sous classes de $Contrainte$.

Pour les initialiser, les contraintes demandent les ensembles utilisés dans la contrainte (liées par leur indice dans le tableau des ensembles, il est possible d'utiliser une fonction pour retrouver l'ensemble grâce au nom de celui-ci) et une priorité utilisée pour trier l'ordre dans lequel les contraintes seront traités.

Création de votre modèle

Vous pouvez bien entendu créer votre propre modèle. Pour cela vous disposez de l'ensemble des contraintes implémentées.

Pour utiliser le solveur, vous devrez donc créer une liste de vos ensembles, une liste de contraintes et une liste vide pour contenir la ou les solutions. En donnant ces listes à la fonction *propagation*, le solveur commencera ça recherche. Si vous ne souhaitez qu'une seule solution, une exception sera envoyé lorsque celle ci sera trouvée.

La possibilité de gérer la parallélisation est disponible mais dans le cas du STS (nécessite la bibliothèque JobLib) est inutile pour les petites instances, le code est présent mais mis en commentaire.

2 STS

2.1 Description du problème

Le Sport Tournament Scheduling (problème 26 du CSPLib) est un problème de planification de rencontres sportives lors d'un tournoi.

Nous considérons les spécifications suivantes :

— Un tournoi regroupe un ensemble n équipes (n pair).

— Chaque équipe rencontre toutes les autres exactement une fois. La notion de matches aller/retour n'est pas traité dans ce problème.

— Le tournoi se déroule sur $n-1$ semaines. Chaque équipe joue une et une seule fois chaque semaine.

— La programmation d'une rencontre se déroule sur un terrain, à une certaine date. Nous emploierons le terme générique de période, au nombre de $n/2$. A chaque période de chaque semaine correspond un match.

— Aucune équipe ne peut jouer plus de deux fois sur une période.

Le problème est de déterminer une programmation qui respecte l'ensemble de ces contraintes.

	Période 0	Période 1	Période 2	Période 3
Semaine 0	0 vs 1	2 vs 3	4 vs 5	6 vs 7
Semaine 1	0 vs 2	1 vs 7	3 vs 5	4 vs 6
Semaine 2	4 vs 7	0 vs 3	1 vs 6	2 vs 5
Semaine 3	3 vs 6	5 vs 7	0 vs 4	1 vs 2
Semaine 4	3 vs 7	1 vs 4	2 vs 6	0 vs 5
Semaine 5	1 vs 5	0 vs 6	2 vs 7	3 vs 4
Semaine 6	2 vs 4	5 vs 6	0 vs 7	1 vs 3

FIGURE 2 – Exemple de planification vérifiant l'ensemble des contraintes pour $n = 8$ équipes numérotées de 0 à 7 (il y a donc 7 semaines et 4 périodes).

Comme dans le tableau précédent, une configuration peut être représentée sous la forme d'un tableau à deux dimensions avec les semaines en lignes et les périodes en colonnes.

2.2 Modélisation

2.2.1 Contraintes ensemblistes

Les données du problème sont :

— n : le nombre d'équipes ($n\%2 = 0$)

— s : le nombre de semaines ($s = n - 1$)

— p : le nombre de périodes ($p = n/2$)

— *equipe* : l'ensemble des équipes

Variables :

— $planning_{i,j}$: planning dont chaque entrée est un ensemble contenant des éléments de *equipe*. $i \in [1, s], j \in [1, p]$.

Contraintes (Les contraintes implémentés ont été addaptées au solveur) :

1. Deux équipes par rencontre :

$$\forall s1 \in [1, s], \forall p1 \in [1, p] \quad |planning_{s1,p1}| = 2$$

2. Toutes les rencontres différentes :

$$\forall s1, s2 \in [1, s]^2, \forall p1, p2 \in [1, p]^2, (s1, p1) \neq (s2, p2) \quad planning_{s1,p1} \neq pl_{s2,p2}$$

3. Un match par équipe par semaine :

$$\forall s1 \in [1, s], \forall p1, p2 \in [1, p]^2, p1 \neq p2 \quad planning_{s1,p1} \cap planning_{s1,p2} = \emptyset$$

4. Pas plus de deux matchs par équipe par période :

$$\forall s1, s2, s3 \in [1, s]^3, \forall p1 \in [1, p], s1 \neq s2 \neq s3 \quad planning_{s1,p1} \cap planning_{s2,p1} \cap planning_{s3,p1} = \emptyset$$

2.2.2 FD

```
int: nbr_equipe = 6;
int: nbr_semaine = nbr_equipe-1;
int: nbr_periode = nbr_equipe div 2;

set of int: equipes = 1..nbr_equipe;
set of int: semaines = 1..nbr_semaine;
set of int: periodes = 1..nbr_periode;

array[semaines,periodes,0..1] of var equipes: planning;

%Une équipe ne peut pas jouer contre elle-même
constraint forall(s in semaines)( forall(p in periodes)(planning[s,p,0]!=planning[s,p,1]));

%Un match n'a lieu qu'une fois
constraint forall(s1 in 1..nbr_semaine-1)(
  forall(s2 in s1+1..nbr_semaine)(
    forall(p1 in 1..nbr_periode-1)(
      forall(p2 in p1+1..nbr_periode)(
        (planning[s1,p1,0]=planning[s2,p1,0] -> planning[s1,p2,1]!=planning[s2,p2,1]) /\
        (planning[s1,p1,0]=planning[s2,p1,1] -> planning[s1,p2,1]!=planning[s2,p2,0]) /\
        (planning[s1,p1,1]=planning[s2,p1,0] -> planning[s1,p2,0]!=planning[s2,p2,1]) /\
        (planning[s1,p1,1]=planning[s2,p1,1] -> planning[s1,p2,0]!=planning[s2,p2,0])
))));
constraint forall(s1 in 1..nbr_semaine-1)(
  forall(s2 in s1+1..nbr_semaine)(
    forall(p in 1..nbr_periode)(
      (planning[s1,p,0]=planning[s2,p,0] -> planning[s1,p,1]!=planning[s2,p,1]) /\
      (planning[s1,p,0]=planning[s2,p,1] -> planning[s1,p,1]!=planning[s2,p,0]) /\
      (planning[s1,p,1]=planning[s2,p,0] -> planning[s1,p,0]!=planning[s2,p,1]) /\
      (planning[s1,p,1]=planning[s2,p,1] -> planning[s1,p,0]!=planning[s2,p,0])
)));

%Une équipe ne joue qu'une fois par semaine
constraint forall(s in semaines)(
  forall(p1 in 1..nbr_periode-1)(
    forall(p2 in p1+1..nbr_periode)(
      planning[s,p1,0]!=planning[s,p2,0] /\ planning[s,p1,1]!=planning[s,p2,0] /\
      planning[s,p1,0]!=planning[s,p2,1] /\ planning[s,p1,1]!=planning[s,p2,1]
)));

%Une équipe ne joue que deux fois par période maximum
constraint forall(s1 in 1..nbr_semaine-2)(
  forall(s2 in s1+1..nbr_semaine-1)(
    forall(s3 in s2+1..nbr_semaine)(
      forall(p in periodes)(
        (planning[s1,p,0]=planning[s2,p,0])
            ->((planning[s1,p,0]!=planning[s3,p,0])/\(planning[s1,p,0]!=planning[s3,p,1])) /\
        (planning[s1,p,0]=planning[s2,p,1])
            ->((planning[s1,p,0]!=planning[s3,p,0])/\(planning[s1,p,0]!=planning[s3,p,1])) /\
        (planning[s1,p,1]=planning[s2,p,0])
            ->((planning[s1,p,1]!=planning[s3,p,0])/\(planning[s1,p,1]!=planning[s3,p,1])) /\
        (planning[s1,p,1]=planning[s2,p,1])
            ->((planning[s1,p,1]!=planning[s3,p,0])/\(planning[s1,p,1]!=planning[s3,p,1]))
))));
```

2.2.3 SAT

Il existe peu (ou des modèles incomplets) de littérature au sujet de la modélisation du problème STS en SAT. Nous avons donc tenté de modélisé le problème "from scratch". Deux représentations sont possibles pour ce problème : l'une en considérant un tableau de booléen par équipe (1000 vs 0001 = équipe 1 contre 4), l'autre en un seul tableau : (1001 = équipe 1 contre 4).

Table 1 – Modélisation intuitive avec des entiers

		Période		
		1	2	3
Semaine	1	2-4	1-5	0-3
	2	0-5	3-4	1-2
	3	3-5	0-2	1-4
	4	1-3	0-4	2-5
	5	0-1	2-3	4-5

Table 2 – Modélisation avec un tableau de booléens

		Période		
		1	2	3
Semaine	1	001010	010001	100100
	2	100001	000110	011000
	3	000101	101000	010010
	4	010100	100010	001001
	5	110000	001100	000011

Table 3 – Modélisation avec deux tableaux de booléens

		Période					
		1		2		3	
Semaine	1	001000	000010	010000	000001	100000	000100
	2	100000	000001	000100	000010	010000	001000
	3	000100	000001	100000	001000	010000	000010
	4	010000	000100	100000	000010	001000	000001
	5	100000	010000	001000	000100	000010	000001

Modélisation 1 avec deux tableaux de booléens

Une rencontre (match) entre deux équipes est modélisée de la sorte :

$$M_{wpsn}$$

Avec w une semaine, p une période, s un Slot (équipe 1, équipe 2) et n le numéro de l'équipe.

Pour des raisons de lisibilité dans certains cas nous n'écrirons pas la forme normale conjonctive. Elle est disponible dans les fichiers source du projet.

Au moins une équipe par slot :

$$\bigwedge_{w}\bigwedge_{p}\bigwedge_{s=1}^{2}\bigvee_{n} M_{wpsn}$$

Au plus une équipe par slot :

$$\bigwedge_{w}\bigwedge_{p}\bigwedge_{s=1}^{2}\bigwedge_{n1<n2} \neg M_{wpsn1} \vee \neg M_{wpsn2}$$

Une équipe joue contre une autre équipe :

$$\bigwedge_{w}\bigwedge_{p}\bigwedge_{s1<s2}\bigwedge_{n} \neg M_{wps1n} \vee \neg M_{wps2n}$$

Pas deux fois le même match dans la même période :

$$\bigwedge_{w1<w2}\bigwedge_{p}\bigwedge_{s1<s2}\bigwedge_{n1<n2} (M_{w1ps1n1} \wedge M_{w1ps2n2}) \rightarrow [(\neg M_{w2ps1n1} \vee \neg M_{w2ps2n2}) \wedge (\neg M_{w2ps2n1} \vee \neg M_{w2ps1n2})]$$

Pas deux fois le même match dans la même semaine :

$$\bigwedge_{w}\bigwedge_{p1<p2}\bigwedge_{s1<s2}\bigwedge_{n1<n2} (M_{wp1s1n1} \wedge M_{wp1s2n2}) \rightarrow [(\neg M_{wp2s1n1} \vee \neg M_{wp2s2n2}) \wedge (\neg M_{wp2s2n1} \vee \neg M_{wp2s1n2})]$$

Pas deux fois le même match dans une semaine et une période différente.

$$\bigwedge_{w1<w2}\bigwedge_{p1<p2}\bigwedge_{s1<s2}\bigwedge_{n1<n2} (M_{w1p1s1n1} \wedge M_{w1p1s2n2}) \rightarrow [(\neg M_{w2p2s1n1} \vee \neg M_{w2p2s2n2}) \wedge (\neg M_{w2p2s2n1} \vee \neg M_{w2p2s1n2})]$$

$$\bigwedge_{w1<w2}\bigwedge_{p1<p2}\bigwedge_{s1<s2}\bigwedge_{n1<n2} (M_{w2p1s1n1} \wedge M_{w2p1s2n2}) \rightarrow [(\neg M_{w1p2s1n1} \vee \neg M_{w1p2s2n2}) \wedge (\neg M_{w1p2s2n1} \vee \neg M_{w1p2s1n2})]$$

Une équipe ne joue qu'une seule fois par semaine :

$$\bigwedge_{w}\bigwedge_{p1<p2}\bigwedge_{s1<s2}\bigwedge_{n1} M_{wp1s1n1} \rightarrow (\neg M_{wp1s2n1} \wedge \neg M_{wp2s1n1} \wedge \neg M_{wp2s2n1})$$

$$\bigwedge_{w}\bigwedge_{p1<p2}\bigwedge_{s1<s2}\bigwedge_{n1} M_{wp1s2n1} \rightarrow \neg M_{wp2s1n1}$$

$$\bigwedge_{w}\bigwedge_{p1<p2}\bigwedge_{s1<s2}\bigwedge_{n1} M_{wp1s2n1} \rightarrow \neg M_{wp2s2n1}$$

Une équipe joue au maximum deux fois par période :

$$\bigwedge_{w1<w2<w3}\bigwedge_{p}\bigwedge_{s1<s2}\bigwedge_{n} [(M_{w1p1s1n} \vee M_{w1p1s2n}) \wedge (M_{w2p1s1n} \vee M_{w2p1s2n})] \rightarrow \neg (M_{w3p1s1n} \vee M_{w3p1s2n})$$

Symétries

Une symétrie de relation d'ordre a été cassée : les équipes dans le slot 1 ont un numéro inférieur à celles dans le slot 2.

Dans un premier temps, et pour tester de casser une symétrie, nous avons interdit la dernière équipe de jouer dans le slot 1 :

nde = dernière équipe
s1 = slot 1

$$\bigwedge_{w}\bigwedge_{p} \neg M_{wps1nde}$$

Si une équipe joue dans le slot 1, alors les équipes ayant un numéro inférieures ou égal ne peuvent pas jouer dans le slot 2

s1 = slot 1
s2 = slot 2
t = (nombre d'équipes -1)

$$\bigwedge_{w}\bigwedge_{p}\bigwedge_{i=1}^{t}\bigwedge_{j=1}^{i+1} M_{wps1i} \rightarrow \neg M_{wps2j}$$

Modélisation 2 avec 1 tableau de booléens

Une rencontre (match) entre deux équipes est modélisée de la sorte :

$$M_{wpt}$$

Avec w une semaine, p une période et t le numéro de l'équipe.
Le modèle 2 est très complexe à modéliser en CNF et nous l'avons modélisé en DNF. Un problème rencontré avec cette modélisation est le temps de conversion, qui est réalisé dans un temps exponentiel. De ce fait nous n'avons pas pu récupérer les résultats du modèle 2 pour plus de 6 équipes. Nous avons tenté, d'après le papier [1] de modéliser une contrainte "Au plus K" (pas plus de deux 1 dans le même tableau), mais nous n'avons pas réussi à l'implémenter dans le temps imparti. Cependant il serait très intéressant d'étudier les résultats avec plus de 6 équipes pour ce modèle. Il n'y a pas de symétries de relation d'ordre dans ce modèle car deux matchs dans le même tableau ne peuvent être modélisés que d'une seule manière (match 1 vs 5 == match 5 vs 1 == 100010).

Au moins deux équipes :

$$\bigwedge_{w}\bigwedge_{p}\bigvee_{t1<t2} M_{wpt1} \wedge M_{wpt2}$$

Au maximum deux équipes :

$$\bigwedge_{w}\bigwedge_{p}\bigwedge_{t1<t2<t3} (M_{wpt1} \wedge M_{wpt2}) \rightarrow \neg M_{wpt3}$$

Une équipe joue 1 fois par semaine :

$$\bigwedge_{w}\bigwedge_{p1<p2}\bigwedge_{t} \neg M_{wp1t} \vee \neg M_{wp1t}$$

Pas deux fois la même équipe par période :

$$\bigwedge_{w1<w2<w3}\bigwedge_{p}\bigwedge_{t} \neg M_{w1pt} \vee \neg M_{w2pt} \vee \neg M_{w3pt}$$

Pas deux fois le même match dans une même période :

$$\bigwedge_{w1<w2}\bigwedge_{p}\bigwedge_{t1<t2} (M_{wpt} \wedge M_{wpt2}) \rightarrow \neg (M_{w2pt} \wedge M_{w2pt2})$$

Pas deux fois le même match dans une semaine et une période différente :

$$\bigwedge_{w1<w2}\bigwedge_{p1<p2}\bigwedge_{t1\neq t2} (M_{wp1t} \wedge M_{wp1t2}) \rightarrow \neg (M_{w2p2t} \wedge M_{w2p2t2})$$

2.3 Amélioration du modèle

Pour améliorer les modèles de STS, nous pouvons définir la première semaine du planning. Cela permettrais d'ignorer toutes les réponses qui ne diffèrent que part une inversion des équipes. En faisant cela, le filtrage sera fortement accéléré. En effet, dès le début du filtrage, il sera possible de fortement diminuer le domaine de la plupart des variables. Et même pour une simple résolution en brute-force, cela économisera le calcul d'une très grande part des solutions possibles. Le temps de calcul économisé pourrais s'accélérer d'un facteur proche de n factoriel (avec n le nombre d'équipe).

Si les équipes sont représentées par des nombres, il est possible de trier les équipes aux cours des semaines. L'idée étant de mettre dans les premières semaines les équipes les plus "petites" dans les premières périodes.

Pour le modèle ensembliste cela donnerais :

$$\forall i1, i2 \in [1, s], i1 < 12$$

$$\sum_{j=1}^{p} sum(planning_{i1,j})^{2nj} \leq \sum_{j=1}^{p} sum(planning_{i2,j})^{2nj}$$

sum(ensemble) est une fonction qui rend la somme des éléments d'un ensemble. Si l'ensemble est vide, sum() rend 0. Néanmoins ce calcul semble assez lourd. Cela peut contrebalancer le temps de calcul gagné. De plus, ce genre de fonction n'est pas implémenté dans notre solveur. Ce genre de contrainte ne semble pas efficace pour accélérer le filtrage.

Pour le SAT, une symétrie de relation d'ordre a été cassée : Les équipes dans le slot 1 on un numéro inférieur à celle dans le slot 2.

Dans un premier temps, et pour tester de casser une symétrie, nous avons interdit la dernière équipe de jouer dans le slot 1 :

nde = dernière équipe
s1 = slot 1

$$\bigwedge_{w} \bigwedge_{p} \neg M_{wps1nde}$$

Si une équipe joue dans le slot 1, alors les équipes ayant un numéro inférieures ne peuvent pas jouer dans le slot 2

s1 = slot 1
s2 = slot 2
t = (nombre d'équipes -1)

$$\bigwedge_{w} \bigwedge_{p} \bigwedge_{i=1}^{t} \bigwedge_{j=1}^{i+1} M_{wps1i} \rightarrow \neg M_{wps2j}$$

2.4 Tests et résultats

2.4.1 SAT

	SAT Modèle 2 tableaux						SAT 1 tableau			FD
	symétries cassées			avec symétries			symétries cassées			avec symétries
	Variables	Clauses	Time	Variables	Clauses	Time	Variables	Clauses	Time	Time
6	180	8100	0.01	180	10020	0.02	90	493440	0.2	0.133
8	448	51590	0.06	448	60872	0.32				0.192
10	900	210255	0.3	900	240690	2.67				0.388
12	1580	651588	2.6	1580	732336	17.424				1.106

Comme nous pouvons le constater, le nombre de clauses du second modèle est très largement supérieur au premier (8100 pour le premier modèle contre 493440 pour le deuxième), bien qu'il n'y ait pas de symétrie d'ordre. Cependant le modèle est plus simple à modéliser. Il faudrait essayer d'enlever les redondances. Pour le moment, le modèle FD semble plus efficace pour un nombre d'équipe croissant.

2.4.2 Contraintes ensemblistes

Nous avons testé l'importance de l'ordre des contraintes pour le temps d'exécution, nous n'avons pas de valeur pour la recherche sans filtrage celle-ci étant trop longue.

Les tests ont été réalisés avec 8 équipes l'objectif étant de trouver une seule solution. Nous disposons de 5 contraintes différentes pour le STS :

— $Cardinalite$, pour la cardinalité de 2 pour chaque case du planning.
— $Different$, équivalent du $alldif$ pour chaque case du tableau.
— $Intersection$, pour les contraintes sur les semaines.
— $Intersection3$, une intersection de 3 variables pour les contraintes sur les périodes.
— $Egal$, pour le cassage de symétrie.

Nous avons lancé plusieurs tests en lançant une contrainte avant puis après toutes les autres afin de voir les impacts sur le temps puis avons compilé ces information dans une configuration "idéale".

Contrainte	Avant	Après
$Cardinalite$	0.78s	1.9s
$Different$	1.1s	0.83s
$Intersection$	1.3s	1.16s
$Intersection3$	3.8s	1s
$Egal$	1s	1s

Comme vous pouvez le constater, $Cardinalite$ devrait plutôt être placé en premier, les contraintes $Different$ et $Intersection$ en 2 ou 3ème position, $Egal$ n'a pas d'impact sur le temps, $Intersection3$ à tout intérêt à être placé en dernier.

Nous avons donc choisi l'ordre $Cardinalite$, $Different$, $Intersection$, $Egal$ puis $Intersection3$. Le résultat étant 0.73s dans cet ordre et 3.7s en inversant l'ordre. Sans casser les symétries, il faut 139s pour résoudre le problème.

Les tests sur l'importance de l'ordre les variables à couper n'ont pas montré de réelle différence sur le temps d'exécution, probablement dû au fait que tous les ensembles non constants ont la même taille pendant la recherche.

2.4.3 Comparatif

En terme de temps cpu, le modèle FD semble le plus efficace à la résolution du problème suivis par le modèle SAT 1 dont les symétries ont étés brisées. Il aurait été intéressant d'implémenter les solutions proposées par [1] pour étendre les recherche autour du modèle SAT 2. L'ensembliste lui est plus lent et plus impacté par la taille de l'espace de recherche que les autres, probablement dû au fait qu'il ne dispose pas d'outils équivalents à l'arc consistance.

Nombre d'équipes	SAT Modèle 1		SAT Modèle 2		FD		Ensembliste	
Symétries Cassées	Oui	Non	Oui	Non	Oui	Non	Oui	Non
6	0.01s	0.02s	0.2s	-	0.1s	0.09s	0.20s	0.29s
8	0.06s	0.32s	-	-	0.39s	0.16s	0.78s	141.8s
10	0.3s	0.67s	-	-	1m8	0.4s	1h30	-
12	2.6s	17.424s	-	-	-	1.17s	-	-

3 SGP

3.1 Social Golfer Problem (SGP)

Le Social Golfer Problem (problème 10 du CSPLib) consiste à organiser des rencontres entre p golfeurs divisés en g groupes durant s semaines de telle sorte qu'aucun golfeur ne joue dans le même groupe que n'importe lequel autre golfeur plus d'une fois.

3.2 Modèle initial

Dans cette partie nous vous présenterons les

3.2.1 Contraintes ensemblistes

Les données du problème sont :
— $nb\_semaine$: le nombre de semaines
— nb_groupe : le nombre de groupes
— $nb\_golfeurs\_par\_groupe$: le nombre de golfeurs par groupe
— $nb\_golfeur = nb\_groupe * nb\_golfeurs\_par\_groupe$: le nombre de golfeurs
Variables :
— $semaines = \{1, nb\_semaine\}$
— $groupes = \{1, nb\_groupe\}$
— $golfeurs = \{1, nb\_golfeur\}$
— $Gr_{s,g} \in golfeurs$
Contraintes :
— 1 - Tout les groupes ont la bonne taille :

$$\forall i \in semaines, \forall j \in golfeurs$$

$$|Gr_{ij}| = nb\_golfeurs\_par\_groupe$$

— 2 - Chaque joueur joue chaque semaine :

$$\forall i \in semaines, \forall g \in golfeurs$$

$$g \in \cup_{j \in groupes} Gr_{ij}$$

— 3 - Les joueurs ne jouent pas dans deux groupes dans la même semaine :

$$\forall i \in semaines, \forall j_1, j_2 \in groupes, j_1 \neq j_2$$

$$G_{i,j_1} \cap G_{i,j_2} = \emptyset$$

— 4 - Deux joueurs ne jouent pas deux fois dans le même groupe :

$$\forall w_1, w_2 \in semaines, w_1 \neq w_2, \forall p_1, p_2 \in golfeurs, p_1 \ \neq p_2, \forall g_1, g_2 \in groupes, g_1 \neq g_2$$

$$(p_1 \in Gr_{w_1 g_1} \wedge p_2 \in Gr_{w_1 g_1} \wedge p_1 \in Gr_{w_2 g_2}) \rightarrow p_2 \notin Gr_{w_2 g_2}$$

3.2.2 FD

```
int: nbr_groupe = 3;
int: nbr_semaine = 3;
int: nbr_golfeur_par_groupe = 3;
int: nbr_golfeur = nbr_groupe*nbr_golfeur_par_groupe;

set of int: groupe = 1..nbr_groupe;
set of int: semaine = 1..nbr_semaine;
set of int: golfeur = 1..nbr_golfeur;

array[semaine, golfeur] of var groupe: groupe_par_semaine;

% Tous les groupes ont la bonne taille.
constraint forall(s in semaine)(
    forall(g in groupe)(
      how_many_in_a_group(s,g)==nbr_golfeur_par_groupe
    )
);

% Si deux golfeurs sont dans le même groupe la même semaine,
% pour toutes les autres semaines, ils sont dans des groupes différents.
constraint forall(g1 in 1..nbr_golfeur-1)(
  forall(g2 in g1+1..nbr_golfeur)(
    forall(s1 in 1..nbr_semaine-1)(
      if groupe_par_semaine[s1,g1]==groupe_par_semaine[s1,g2]
        then forall(s2 in s1+1..nbr_semaine)(
          groupe_par_semaine[s2,g1]!=groupe_par_semaine[s2,g2]
        )
      endif
    )
  )
);

% Ces fonctions servent à cacluler le nombre de golfeurs dans un groupe.
function var int: how_many_in_a_group(int: semaine, int: groupe) =
  how_many(semaine, groupe, 1);

function var int: how_many(int: semaine, int: groupe, int: golfeur) =
  if golfeur > nbr_golfeur then 0
  else
    if groupe_par_semaine[semaine,golfeur]==groupe
      then 1+how_many(semaine, groupe, golfeur+1)
    else
      how_many(semaine, groupe, golfeur+1)
    endif
  endif;

solve satisfy;
```

3.2.3 SAT

Pour modéliser le SGP sous forme SAT, nous utilisons deux modèles qui diffèrent sur la contrainte de socialisation pour faire des observations sur le nombre de clauses générées. Les contraintes sur les joueurs et le groupes de ces deux modèles sont semblables, mais celle de socialisation peut être faite sous la forme d'une seule contrainte pour un modèles avec un encodage direct [3] (DE) et un encodage avec des variables intermédiaires [4] (TME).

En ce qui concerne l'encodage de la solution, il nous suffit de prendre un encodage utilisant des entiers et les

remplacer par les booléens en adaptant les contraintes :

	semaine					
	1		2		3	
1	1	2	1	3	1	4
2	3	4	2	4	2	3

Il est donc possible de reprendre l'encodage avec les entiers avec une dimension supplémentaire pour représenter les joueurs.

		semaine					
joueur	groupe	place		place		place	
1	1	1	0	1	0	1	0
	2	0	0	0	0	0	0
2	1	0	1	0	0	0	0
	2	0	0	1	0	1	0
3	1	0	0	0	1	0	0
	2	1	0	0	0	0	1
4	1	0	0	0	0	0	1
	2	0	1	0	1	0	0

— Chaque joueur joue au moins une fois par semaine quelque soit le groupe ou la position du joueur :

$$\bigwedge_{q'=1}^{q}\bigwedge_{w'=1}^{w}\bigvee_{p'=1}^{p}\bigvee_{g'=1}^{g} G_{q'w'p'g'}$$

— Chaque joueur joue au plus une fois par semaine :

$$\bigwedge_{q'=1}^{q}\bigwedge_{w'=1}^{w}\bigwedge_{p'=1}^{p}\bigwedge_{g'=1}^{g}\bigwedge_{p''=p'+1}^{p} \neg G_{q'p'g'w'} \vee \neg G_{q'p''g'w'}$$

— Un joueur ne joue que dans un seul groupe par semaine :

$$\bigwedge_{q'=1}^{q}\bigwedge_{w'=1}^{w}\bigwedge_{p'=1}^{p}\bigwedge_{g'=1}^{g}\bigwedge_{g''=g'+1}^{g}\bigwedge_{p''=p'+1}^{p} \neg G_{q'p'g'w'} \vee \neg G_{g'p''g''w'}$$

— Au moins un joueur joue dans un groupe à une certaine position :

$$\bigwedge_{w'=1}^{w}\bigwedge_{p'=1}^{p}\bigwedge_{g'=1}^{g}\bigvee_{q'=1}^{q} G_{q'g'p'w'}$$

— Au plus un joueur joue dans un groupe à une certaine position :

$$\bigwedge_{w'=1}^{w}\bigwedge_{p'=1}^{p}\bigwedge_{g'=1}^{g}\bigwedge_{q'=1}^{q}\bigwedge_{q''=q'+1}^{q} \neg G_{q'p'g'w'} \vee \neg G_{q''p'g'w'}$$

— Deux joueurs ne peuvent pas jouer dans le même groupe deux fois, pour un joueur 1 jouant dans un groupe une certaine semaine avec un joueur 2, le joueur 1 jouant dans un groupe une autre semaine implique que le joueur 2 ne peut pas jouer dans le même groupe :

$$\bigwedge_{w'=1}^{w}\bigwedge_{g'=1}^{g}\bigwedge_{w''=w'+1}^{w}\bigwedge_{g''=1}^{g}\bigwedge_{q'=1}^{q}\bigwedge_{p_1=1}^{p}\bigwedge_{p'_1=1}^{p}\bigwedge_{q''=q'+1}^{q}\bigwedge_{p_2=1}^{p}\bigwedge_{p'_2=1}^{p}$$

$$G_{q'p_1g'w'} \wedge G_{q''p_2g'w'} \wedge G_{q'p'_1g''w''} \rightarrow \neg G_{q''p'_2g''w''}$$

3.3 Modèle amélioré

Pour briser les symétries de ce modèle comme l'ordre des joueurs dans un groupe, l'ordre des groupes dans une semaine et l'ordre des semaines.

— ordonner les joueurs dans un ordre croissant :

$$\bigwedge_{q'=1}^{q}\bigwedge_{p'=1}^{p-1}\bigwedge_{g'=1}^{g}\bigwedge_{w'=1}^{w}\bigwedge_{q''=q'+1}^{q'} \neg G_{q'p'g'w'} \vee \neg G_{q''p'(g'+1)w'}$$

— ordonner les groupes avec leur dernier joueur :

$$\bigwedge_{q'=1}^{q}\bigwedge_{g'=1}^{g-1}\bigwedge_{w'=1}^{w}\bigwedge_{q''=1}^{q'-1} \neg G_{q'1g'w'} \vee \neg G_{q''1(g'+1)w'}$$

— ajoute le second joueur du premier groupe pour chaque semaines :

$$\bigwedge_{q'=1}^{q}\bigwedge_{w'=1}^{w}\bigwedge_{q''=1}^{q'} \neg G_{q'21(w'+1)} \vee \neg G_{q''21(w'+1)}$$

3.4 Tests et résultats

Ayant deux modèles disponibles pour le problème du golfeur, il est possible de faire une comparaison, en particulier à la manière dont est implémentée la contrainte de sociabilité, avec ou sans variables intermédiaires.

	DE						TME					
	sym			nosym			sym			nosym		
	v	c	t	v	c	t	v	c	t	v	c	t
5-3-1	225	2280	0.02	225	3694	0.003	300	3480	0	300	5160	0.004
5-3-2	450	217185	0.3	450	220118	0.029	600	9585	0.001	600	13065	0.00
5-3-3	675	644715	1.42	675	649167	1.58	900	18315	0.02	900	23595	0.01
5-3-4	900	1284870	4.24	900	1290481	4.46	1260	29670	0.01	1260	36750	0.02
5-3-5	1125	2137650	9.37	1125	2145140	10.58	1500	43650	0.03	1500	52530	0.07
5-3-6	1350	3203066	9.21	1350	3212664	22.57	1800	60255	0.21	1800	70935	0.26

On observe comme prévu que le nombre de variable du TME est supérieur au DE à cause des variables intermédiaires de Ladder et le que le nombre de clause augmente entre les modèles dont les symétries sont brisés et sans ces mêmes symétries. Une autre chose est concernant le temps de résolution à cause du nombre de clauses générées en particulier pour le DE, son nombre de clauses étant élevés les temps de résolution le sont tout autant, contrairement au TME gardant des temps raisonnable.

Nous pouvons maintenant comparer avec le modèle FD à l'aide du solver Gecode.

	FD
	t(s)
5-3-1	11
5-3-2	0.188
5-3-3	0.197
5-3-4	0.566
5-3-5	23
5-3-6	-

On constate que les temps de résolution sont plus long que le SAT lors de prévision supérieure à 5 semaines.

4 Conclusion

Ce projet nous à permis à chacun de travailler sur différents aspects de la programmation par contrainte. Chaque modèle à ses particularités, les modèles FD et Ensemblistes sont certes plus faciles à comprendre comparé aux modèles SAT mais leur temps de résolution est beaucoup plus long, suivant le modèle. Par exemple, pour le sts, la résolution FD est similaire au SAT, mais pour le SGP le sat est plus rapide. Les modèles SAT étant composé de plus de variables que les modèles FD et Set et suivant les contraintes pouvant mener à une certaine difficulté dans l'expressivité du problème à modéliser. Les modèles Set donne une meilleure abstraction sans avoir à se préoccuper de dimension comme les modèles SAT mais des contraintes plus compliquées à intégrer à un solveur.

Implémenter le solveur nous a permis de mieux comprendre le fonctionnement des modèles ensemblistes et nous a poussé à optimiser le code pour avoir des temps d'exécution raisonnables. Le temps de résolution sur le STS avec 10 équipes étant cependant trop élevé. Pour résoudre ce problème, nous pourrions apporter plusieurs optimisations, entre autre le filtrage sur la cardinalité et lexicographique pour réduire l'espace de recherche plus rapidement. La gestion du treillis pourrait aussi être modifiée pour ajouter au fur et à mesure des éléments de la borne supérieure à la borne inférieure, bien que générer le treillis sous ensemble par sous ensemble nous à permis de beaucoup réduire le temps de calcul (et la consommation de RAM en particulier en parallèle...).

Références

[1] Paul Maximilian Bittners, Thomas Thüm, and Ina Schaefer. Sat encodings of the at-most-k constraint. *Lecture Notes in Computer Science*, 09 2019.

[2] Warwick Harvey. CSPLib problem 010 : Social golfers problem. `http://www.csplib.org/Problems/prob010`.

[3] Frédéric Lardeux, Eric Monfroy, Broderick Crawford, and Ricardo Soto. Set constraint model and automated encoding into sat : Application to the social golfer problem. *Annals of Operations Research*, 235, 06 2014.

[4] Markus Triska and Nysret Musliu. An improved sat formulation for the social golfer problem. *Annals OR*, 194 :427–438, 04 2012.

[5] Toby Walsh. CSPLib problem 026 : Sports tournament scheduling. `http://www.csplib.org/Problems/prob026`.