Université de Nantes

UFR Sciences et Techniques

Master informatique parcours
"optimisation en recherche opérationnelle (ORO)"
Année académique 2019-2020

Projet n 1
-
Set Constraint Modeling

Auteur :
Arthur Gontier
Adrien Cassaigne
Thaddeus Leonard
Oryan Rampon
Jonathan Fontaine
Corentin Pelhâtre
Mathis Ocquident

Référent :
Eric Monfroy

19 décembre 2019

Table des matières

1 Modélisation :

1.1 The Social Golfer Problem (SGP) :

Le SGP est un problème bien connu de la PPC. L'objectif est de planifier g*p golfeurs dans g groupes de p joueurs pour w semaines tel que :

— Il n'y a jamais deux golfeurs qui jouent dans le même groupe plus d'une fois.

— Tous les golfeurs jouent une fois par semaine.

— Le nombre de groupes et le nombre de joueurs par groupe reste inchangé au cours des w semaines.

Une instance du SGP est définie par le triplet (g,p,w) :

— g : Nombre de groupes

— p : Nombre de joueurs par groupe

— w : Nombre de semaines

Prenons l'exemple d'une petite instance avec le triplet (3,3,3). Une solution possible pour le problème est alors de la forme :

Semaine 1			
Groupe 1	1	4	7
Groupe 2	2	5	8
Groupe 3	3	6	9

Semaine 2			
Groupe 1	1	2	3
Groupe 2	4	5	6
Groupe 3	7	8	9

Semaine 3			
Groupe 1	1	5	9
Groupe 2	2	6	7
Groupe 3	3	4	8

1.1.1 Modèle ensembliste :

Dans cette partie, nous nous intéresserons à la méthode que nous avons utilisé pour modéliser le problème de SGP de manière ensembliste.

1.1.1.1 Ensembles utilisés :

Voici, ci dessous, les ensembles utilisés pour cet exercice :

— P : l'ensemble des joueurs

— S : l'ensemble des semaines

— $s \in S$: l'ensemble des groupes d'une semaine

— $g \in s \in S, g \subset P$: l'ensemble des joueurs d'un groupe d'une semaine

Les ensembles utilisés ont les cardinalités suivantes :

— $|S| = w$

— $(\forall s \in S)|s| = g$

— $(\forall g \in s \in S)|g| = p$

1.1.1.2 Présentation du modèle :

$$g_1 \cap g_2 = \varnothing \qquad (\forall s \in S)(\forall g_1, g_2 \in s | g_1 \neq g_2) \tag{1}$$

$$|g_1 \cap g_2| \leq 1 \qquad (\forall s_1, s_2 \in S | s_1 \neq s_2)(\forall g_1 \in s_1, \forall g_2 \in s_2) \tag{2}$$

1 Chaque joueur ne joue que dans un groupe chaque semaine.

2 Chaque joueur rencontre des nouveaux joueurs chaque semaine, ie ne rencontre jamais deux fois le même joueur.

1.1.2 Modèle FD entier :

Cette partie sera quant à elle consacrée à la modélisation en FD entier.

1.1.2.1 Variables :

$$x_{ijk} = \begin{cases} 1 : \text{Si le joueur k joue dans le groupe j la semaine i.} \\ 0 : \text{Sinon.} \end{cases}$$

$$z_{ijq_1q_2} = \begin{cases} 1 : \text{si } q_1 \text{ et } q_2 \text{ jouent ensemble dans le groupe j la semaine i.} \\ 0 : \text{Sinon.} \end{cases}$$

1.1.2.2 Contraintes :

$$\sum_{j \in g} x_{ijk} = 1 \qquad \forall i \in w, \forall k \in p \tag{3}$$

$$\sum_{k \in p} x_{ijk} = p \qquad \forall i \in w, \forall j \in g \tag{4}$$

$$z_{ijq_1q_2} \geq x_{ijq_1} + x_{ijq_2} - 1 \qquad \forall i \in w, \forall j \in g, \forall q_1, q_2 \in p, q_1 \neq q_2 \tag{5}$$

$$\sum_{i \in w, j \in g} z_{ijq_1q_2} \leq 1 \qquad \forall q_1, q_2 \in p, q_1 \neq q_2 \tag{6}$$

3 Le joueur k ne joue qu'une fois par semaine (est dans un seul groupe).

4 Il y a exactement p joueurs dans chaque groupe, chaque semaines.

5 Liaison entre la variable auxiliaire z et les variables x associées.

6 les joueurs q_1 et q_2 jouent au plus une fois dans le même groupe.

1.1.3 Modèle SAT :

Dans cette partie nous nous intéresserons au modèle SAT du problème SGP. Afin de créer ce modèle, nous nous sommes fortement inspirés du modèle FD entier allant jusqu'à reprendre une à une les contraintes de ce dernier afin de les transformer en SAT.

1.1.3.1 Méthode de transformation :

Voici d'abord un algorithme qui énumère les arrangements de k parmis n. Cet algoritme sera utilisé lors de la transformation du modèle FD au modèle SAT.

Algorithme 1 : enumar(f,ii1,jj,k,n,sign)

```
1  ar = collect(1 :k)
2  writear(f,ii1,jj,1,k,ar)
3  pour i in 2 :binomial(n,k) faire
4      ii = 0
5      tant que ii<k faire
6          Si ar[end-ii] + 1≤ n − ii alors
7              ar[end-ii] = ar[end-ii] + 1
8              pour iii in k-ii+1 :k faire
9                  ar[iii] = ar[iii-1]+1
10             fin
11             ii = k
12         Sinon
13             ii = ii + 1
14         fin
15     fin
16     Si sign alors
17         writear(f,ii1,jj,i,k,ar)
18     Sinon
19         writearneg(f,ii1,jj,i,k,ar)
20     fin
21 fin
```

Exemple d'exécution sur 3 parmi 5 : 1 2 3 - 1 2 4 - 1 2 5 - 1 3 4 - 1 3 5 - 1 4 5 - 2 3 4 - 2 4 5 - 3 4 5

$$\sum_{a \in A} x_{ab} = c \qquad \forall b \in B \tag{7}$$

$$\Leftrightarrow \sum_{a \in A} x_{ab} \geq c \quad \& \quad \sum_{a \in A} x_{ab} \leq c \qquad \forall b \in B \tag{8}$$

Tout d'abord, a partir des propriétés basiques des problèmes FD entiers, on peut transformer l'égalité 7 en deux inégalités 8.

Ensuite, on peut transformer les contraintes 8 en SAT aisément tel que :

$$\sum_{a \in A} x_{ab} \geq c \quad \forall b \in B \qquad \Rightarrow \bigwedge_{b \in B} \bigvee_{a \in \binom{q}{q-c+1}} x_{ab} \tag{9}$$

$$\sum_{a \in A} x_{ab} \leq c \quad \forall b \in B \qquad \Rightarrow \bigwedge_{b \in B} \Big(\bigvee_{\forall a \in \binom{q}{c+1}} \neg x_{ab} \Big) \tag{10}$$

Ainsi, dans le paragraphe suivant, nous préciserons de quelle contrainte du modèle FD entier est tiré chaque ensemble de clause du modèle SAT.

1.1.3.2 Écriture du modèle :

$$\bigwedge_{i,k} \bigvee_{j} x_{ijk} \tag{11}$$

$$\bigwedge_{i,j,j',k;j\neq j'} (\neg x_{ijk} \vee \neg x_{ij'k}) \tag{12}$$

$$\bigwedge_{i,j} \bigvee_{k\in\binom{q}{q-p+1}} x_{ijk} \tag{13}$$

$$\bigwedge_{i,j} \bigvee_{k\in\binom{q}{p+1}} \neg x_{ijk} \tag{14}$$

$$\bigwedge_{(i,j)\neq(i',j'),k\neq k'} (\neg x_{ijk} \vee \neg x_{ijk'} \vee \neg x_{i'j'k} \vee \neg x_{i'j'k'}) \tag{15}$$

11 le joueur k est dans au moins un groupe par semaine (à partir de 3)

12 si le joueur k, est dans le groupe j, il n'est dans aucun autre (à partir de 3)

13 et 14 on veut p joueurs dans un groupe (à partir de 4)

15 les joueurs k et k' ne jouent pas plus d'une fois dans le même groupe (à partir de 5 et 6)

1.2 Sports Tournament Scheduling (STS) :

Le STS est un problème tout aussi connu de la PPC visant a créer un tournoi entre n équipes réparties sur n/2 terrains en n-1 semaines.

— Chaque équipe doit jouer contre les n-1 autres équipes dans le temps imparti

— Chaque equipe ne peut jouer qu'au maximum deux fois sur un terrain.

L'objectif de cette partie est donc de fournir un emplois du temps des n/2 terrains sur n-1 semaines avec pour chaque élément de ce dernier, les deux équipes qui se rencontrent. On a donc un unique paramètre n pour ce problème. Voici une solution pour $n = 6$:

Semaine 1	
Terrain 1	1 2
Terrain 2	3 4
Terrain 3	5 6

Semaine 2	
Terrain 1	1 5
Terrain 2	3 6
Terrain 3	2 4

Semaine 3	
Terrain 1	1 2
Terrain 2	3 4
Terrain 3	5 6

Semaine 4	
Terrain 1	1 2
Terrain 2	3 4
Terrain 3	5 6

Semaine 5	
Terrain 1	1 2
Terrain 2	3 4
Terrain 3	5 6

1.2.1 Modèle ensembliste :

1.2.1.1 Ensembles utilisés :

— E : l'ensemble des équipes

— $M_{ij} \subset E$: Terrain j semaine i, un ensemble de deux équipes

Cardinalitées associées :

— |E| = n

— $|M_{ij}|$=2 $\forall$ j,i

1.2.1.2 Présentation du modèle :

$$M_{i,j} \cap M_{i,j'} = \varnothing \qquad \forall i, j, j' \; avec \; j \neq j' \tag{16}$$

$$|M_{i_1 j} \cap M_{i_2 j} \cap M_{i_3 j}| = 0 \qquad \forall i_1 \neq i_2 \neq i_3 \;\; \forall j \tag{17}$$

$$|\bigcup_{ij} M_{ij}| = \frac{(n-1)n}{2} \qquad \Leftrightarrow allDifferent(M_{ij} \forall i, j) \tag{18}$$

16 Chaque équipe ne joue qu'une fois par semaine

17 Chaque équipe joue 2 fois max sur le terrain j. Pour chaque sous-ensemble de trois semaines, on regarde si une équipe joue sur les trois (intersection des combinaisons de trois semaines)

18 Cardinal de l'union de toutes les semaines = nombre de match possibles = $(n-1)n/2$

1.2.2 Modèle FD entier :

1.2.2.1 Variables :

$$x_{ijk} = \begin{cases} 1 : \text{Si l'équipe k joue sur le terrain j la semaine i.} \\ 0 : \text{Sinon.} \end{cases}$$

$$z_{ijk_1k_2} = \begin{cases} 1 : \text{si } k_1 \text{ et } k_2 \text{ jouent ensemble sur le terrain j la semaine i.} \\ 0 : \text{Sinon.} \end{cases}$$

1.2.2.2 Contraintes :

$$\sum_k x_{ijk} = 2 \qquad \forall i, j \tag{19}$$

$$\sum_j x_{ijk} = 1 \qquad \forall i \forall k \tag{20}$$

$$\sum_i x_{ijk} \leq 2 \qquad \forall j, k \tag{21}$$

$$z_{ijk_1k_2} \leq \frac{x_{ijk_1} + x_{ijk_2}}{2} \qquad \forall i, j, k_1, k_2 \quad k_1 \neq k_2 \tag{22}$$

$$\sum_{i,j} z_{ijk_1k_2} \geq 1 \qquad \forall k_1, k_2 \quad k_1 \neq k_2 \tag{23}$$

19 A chaque semaine, sur chaque terrain, il y a deux équipes.

20 Chaque équipe joue une fois chaque semaine.

21 Chaque équipe joue au plus deux fois sur le même terrain.

22 Contrainte qui fixe la valeur de la variable auxiliaire $z_{ijk_1k_2}$.

23 Chaque équipe rencontre au moins une fois chaque autre.

1.2.3 Modèle SAT :

Dans cette partie, comme pour le SAT du SGP, nous allons reprendre les contraintes du modèle en FD entier afin de les retranscrire en SAT. Nous utiliserons d'ailleurs les mêmes variables (x et z) :

$$\bigwedge_{i,j} \bigvee_{k \in \binom{q}{q-1}} x_{ijk} \tag{24}$$

$$\bigwedge_{i,j,k,k';k \neq k' \neq k''} (\neg x_{ijk} \vee \neg x_{ijk'} \vee \neg x_{ijk''}) \tag{25}$$

$$\bigwedge_{i,k} \bigvee_{j} x_{ijk} \tag{26}$$

$$\bigwedge_{i,j,j',k;j \neq j'} (\neg x_{ijk} \vee \neg x_{ij'k}) \tag{27}$$

$$\bigwedge_{i,i',i'',j,k;i \neq i' \neq i''} (\neg x_{ijk} \vee \neg x_{i'jk} \vee \neg x_{i''jk}) \tag{28}$$

$$\bigwedge_{i,j,k,k';k \neq k'} (z_{ijkk'} \vee \neg x_{ijk} \vee \neg x_{ijk'}) \tag{29}$$

$$\bigwedge_{i,j,k,k';k \neq k'} (\neg z_{ijkk'} \vee x_{ijk}) \tag{30}$$

$$\bigwedge_{i,j,k,k';k \neq k'} (\neg z_{ijkk'} \vee x_{ijk'}) \tag{31}$$

$$\bigwedge_{k,k',k \neq k'} \bigvee_{i,j} z_{ijkk'} \tag{32}$$

24 et 25 A chaque semaine, sur chaque terrain, il y a deux équipes. (à partir de 19)

26 et 27 Chaque équipe joue une fois chaque semaine. (à partir de 20)

28 Chaque équipe joue au plus deux fois sur le même terrain (à partir de 21)

29, 30 et 31 variables qui fixent la valeur de z (à partir de 22)

32 Chaque équipe rencontre au moins une fois chaque autre (à partir de 23)

2 Tunning :

2.1 Tunning global :

2.1.1 Modèle booleen :

Lors des grands opérateurs de la formulation booléenne, on a souvent un $\bigwedge_{k,k',k\neq k'}$ Il est alors possible de réduire le nombre de clauses en remplaçant simplement le $k \neq k'$ par $k < k'$. Ainsi en prenant par exemple $k \in \{1, 2, 3, 4\}$ on a douze couples (k,k') possibles alors qu'avec cette nouvelle formulation $k < k'$, on a plus que six couples possibles tout en conservant tous les formats de solutions possibles.

$k \neq k'$	1,2	1,3	1,4	2,1	2,3	2,4	3,1	3,2	3,4	4,1	4,2	4,3
$k < k'$	1,2	1,3	1,4	2,3	2,4	3,4						

Avec cette méthode nous arrivons donc grandement à réduire la complexité tout en ne perdant aucune information.

2.1.2 Algorithme simple pour la première semaine

Dans l'objectif d'améliorer notre temps d'exécution du solveur classique, nous avons pensé lui donner en "présolve" un maximum d'informations afin de réduire au plus l'espace de recherche.

En effet, il est souvent possible de calculer le premier groupe de la première semaine avec un algorithme polynomial. Une fois ce ou ces premiers groupes fixés, le solveur démarre la résolution bien plus rapidement.

Tout d'abord, dans le cas du SGP, cet ajout s'est avéré d'une grande utilité :

	Sans	Avec
SGP(4,4,4) FD	30,37s	8,13s
SGP(4,4,4) SAT	2,26s	0,07s
SGP(5,5,5) B&B	>5min	1,15s

Par ailleurs, sur le STS, nous avons remarqué une étonnante dégradation des temps d'exécution pour le SAT en utilisant l'algorithme pour la première semaine :

	Sans	Avec
STS(6) B&B	0.3s	1s
STS(8) B&B	85s	>5min
STS(8) FD	4s	3,8s
STS(10) SAT	0,63s	0,04s

2.2 The Social Golfer Problem (SGP) :

2.2.1 Modèle ensembliste :

On remarque qu'il existe de nombreuses symétries dans ce problème, on a donc rajouté des contraintes pour casser certaines de ces symétries. Pour cela on fixe les éléments des groupes de la première semaine, puis ce du premier groupe de la deuxième semaine et finalement les premiers éléments des groupes de chaque semaine.

	Sans	Avec
SGP(4,4,4)	0,103s	0,092s
SGP(5,5,5)	1min 27s	0,139s

2.2.2 Modèle SAT :

Dans la contrainte (i, j) $\neq$ (i', j'), on remarque que pour notre problème, i $\neq$ i' car on a une partition des joueurs sur une semaine, il est donc inutile de comparer ces variables. En effet la contrainte 15 contraint à ce que deux couples de (semaine,groupe) soient différent. De ce fait il faut que soit la semaine soit différente ou alors que les groupes soient différents. Or on contraint avec 11,12 à ce que les joueurs jouent qu'une seule fois par semaine, on ne peut donc pas avoir k appartenant à deux groupes d'une même semaine. Ainsi i et i' sont obligatoirement différent.

2.3 Sports Tournament Scheduling (STS) :

Nous avons appliqué tous les tunning présentés dans la partie 2.1 avec plus ou moins de bénéfice sur les temps d'exécution de cette partie. Nous n'avons pas réalisé de tunning spécifique sur ce problème.

3 Solveur artisanal :

Nous avons implémenté un solveur générique, qui consiste en un branch and bound, qui commence par une phase de réduction des domaines des variables à l'aide d'algorithme de filtrage donné en entrée (définit par les contraintes du problème), puis on effectue un branchement, qui consiste à ajouter un élément de l'ensemble min d'une variable non close.

3.1 Structures :

On définit premièrement une variable comme suit :

min : l'ensemble minimum de la variable

max : l'ensemble maximum de la variable

card_min : le cardinal minimal de la variable

card_max : le cardinal maximal de la variable

univers : l'univers de la variable

Nos variables seront dans une collection ordonnée. Nos contraintes utilisent leur indice pour modifier les variables présentes dans la liste, utile dans le branch and bound, afin de faire une copie profonde d'une liste, pour ne pas avoir à restaurer une variable lors d'une remontée dans l'arbre.

On pose la fonction *valide*(*variable*) retournant un booléen, *vrai* si la variable ne présente aucune contradiction (l'ensemble min qui a des éléments hors de l'ensemble max, card min plus grand que taille(max), etc).

Une contrainte se présente comme ceci :

— indices_argument : la liste des indices des variables

— filtrage ! : l'algorithme de filtrage à appliquer au variables

Posons *variable*(*contrainte*, *variables*) la fonction qui retourne les variables filtrées par la contrainte présente dans la liste variables.

Pour filtrer une contrainte, on suit l'algorithme suivant :

Algorithme 2 : filtrer !(contrainte, variables)

```
1  n ← taille(contrainte.indices_argument)
   // Récupération des variables
2  arguments ← TableauVariable[n]
3  pour i de 1 à n faire
4  |  j = contrainte.indices_argument[i]
5  |  arguments[i] = variables[j] // copie de pointeurs ! pas de copie profonde
6  fin
   // algorithme de filtrage de la contrainte
7  contrainte.filtrage !(arguments)
   // effectuer un filtrage individuel sur chaque variable
8  pour var ∈ arguments faire
9  |  filtrage_individuel !(var, ctr.univers)
10 fin
```

Le filtrage individuel consiste en de simples tests triviaux mais nécessaires sur la variable, par exemple réduire le cardinal max d'une variable quand la taille de l'ensemble max est plus petit.

3.2 Phase de réduction :

Algorithme 3 : solver_generique !(variables, contraintes)

```
1  contraintesVariables ← Tableau qui associe, à la case i, la liste des indices des contraintes dans
     lesquelles est présente la variable à l'indice i dans le tableau variables
2  pileF ← Pile contenant tous les indices des contraintes. // contrainte à filtrer
3  infaisable ← Faux
4  tant que nonVide(pileF) et non infaisable faire
5  |  indiceCtr = depiler(pileF)
6  |  ctr ← contraintes[indiceCtr]
7  |  filtrer !(ctr, variables)
8  |  pour var ∈ variable(ctr, variables) faire
9  |  |  Si var à changer alors
10 |  |  |  Si valide(var) alors
11 |  |  |  |  pour indCtr ∈ contraintesVariables[var] faire
12 |  |  |  |  |  Si indCtr ∉ pileF alors
13 |  |  |  |  |  |  empiler(pileF, indCtr)
14 |  |  |  |  |  fin
15 |  |  |  |  fin
16 |  |  |  Sinon
17 |  |  |  |  infaisable ← Vrai
18 |  |  |  fin
19 |  |  fin
20 |  fin
21 fin
22 return non infaisable
```

3.3 Branch and Bound :

Algorithme 4 : branch_and_bound !(variables, contraintes)

```
1  faisable = solver_generique !(variables, contraintes)
2  Si faisable alors
3  |  nonClot ← liste de variables non clot
4  |  Si non est_vide(nonClot) alors
5  |  |  trier nonClot dans l'ordre croissant des variables les plus proches d'être réalisées // plus petit
   |  |     différence entre la taille de l'ensemble min et le cardinal max
6  |  |  varBranch ← nonClot[1]
7  |  |  candidat ← Liste(varBranch.max \ varBranch.min)
8  |  |  ptr ← candidat.premier
9  |  |  faisableTemp ← false
10 |  |  tant que ptr ≠ ∅ and non faisableTemp faire
11 |  |  |  varCopie ← copie(variables)
12 |  |  |  inserer(varCopie[varBranch].min, ptr.val)
13 |  |  |  faisableTemp ← branch_and_bound !(varCopie, contraintes)
14 |  |  |  Si non faisableTemp alors
15 |  |  |  |  varBranch.max ← varBranch.max \ ptr.val
16 |  |  |  fin
17 |  |  |  valeur ← ptr.suiv
18 |  |  fin
19 |  |  Si faisableTemp alors
20 |  |  |  variables ← varCopie
21 |  |  Sinon
22 |  |  |  faisable ← solver_generique !(variables, contraintes)
23 |  |  fin
24 |  fin
25 fin
26 return faisable
```

Notre Branch and Bound fait une recherche en profondeur et retourne la première solution trouvée.

Pour la règle de branchement, on prend la variable la plus près d'être close, c'est à dire avec la plus petite différence entre la taille de l'ensemble min, et le cardinal max.
Par curiosité, on a essayé l'inverse, mais les résultats étaient catastrophiques, 100 fois pire en temps.

3.4 Algorithme de filtrage :

Pour résoudre nos problème on doit mettre en place un algorithme de filtrage pour nos contraintes.

- $(v_1 \cap v_2) = \varnothing$
  $v_1^{\uparrow} \leftarrow v_1^{\uparrow} \backslash v_2^{\downarrow}$
  $v_2^{\uparrow} \leftarrow v_2^{\uparrow} \backslash v_1^{\downarrow}$
- $|v_1 \cap v_2| \leq 1$
  Ne filtrer que si $(v_1 \cap v_2) \neq \varnothing$. Prenons $val \in (v_1^{\downarrow} \cap v_2^{\downarrow})$:
  $v_1^{\uparrow} \leftarrow v_1^{\uparrow} \backslash (v_2^{\downarrow} \backslash val)$
  $v_2^{\uparrow} \leftarrow v_2^{\uparrow} \backslash (v_1^{\downarrow} \backslash val)$
- $(v_1 \cap v_2 \cap v_3) = \varnothing$
  $v_1^{\uparrow} \leftarrow v_1^{\uparrow} \backslash (v_2^{\downarrow} \cap v_3^{\downarrow})$
  $v_2^{\uparrow} \leftarrow v_2^{\uparrow} \backslash (v_1^{\downarrow} \cap v_3^{\downarrow})$
  $v_3^{\uparrow} \leftarrow v_3^{\uparrow} \backslash (v_1^{\downarrow} \cap v_2^{\downarrow})$

Remarquons qu'il n'y a pas d'algorithme de filtrage pour la contrainte alldifferent, car on peut la représenter par une contrainte : $|v_1 \cap v_2| \leq \max_v\{cardmax(v)\} = 1$ dans notre cas. Cette représentation est valide, mais n'est pas optimale en terme de filtrage.

Revenons un peu plus en détail sur les différents algorithmes de filtrage spécifiques à chaque contrainte, à travers quelques exemples.

Le filtrage de la première contrainte $(v_1 \cap v_2) = \varnothing$ permet d'éliminer de l'ensemble maximum d'une variable les éléments contenus dans l'ensemble minimum de l'autre variable, et donc les éléments obligatoirement présents dans l'intersection.
Illustrons avec l'exemple suivant : soit $v_1^{\downarrow} = \{1, 2\}$, $v_1^{\uparrow} = \{1, 2, 3, 4, 5\}$ et $v_2^{\downarrow} = \{2, 3\}$, $v_2^{\uparrow} = \{1, 2, 3, 6, 7\}$
On a donc v_1 qui contient au moins 1 et 2, et comme l'intersection est vide, on peut donc les retirer des valeurs possibles pour v_2. De même pour v_2 qui contient au minimum les éléments 2 et 3. En mettant à jour, on obtient alors :
$v_1^{\downarrow} = \{1, 2\}$, $v_1^{\uparrow} = \{1, 4, 5\}$ et $v_2^{\downarrow} = \{2, 3\}$, $v_2^{\uparrow} = \{3, 6, 7\}$
On remarque ainsi que quelques incohérences ont été crées entre les ensembles minimum et maximum qui seront réparées lors du filtrage individuel de chaque variable.

Concernant la deuxième contrainte et donc le filtrage associé, on s'intéresse plus particulièrement aux valeurs communes des deux variables. On commence par déterminer l'ensemble *Val* qui prend les éléments qui sont au moins compris dans l'intersection des ensembles minimum des deux variables présentes dans la contrainte. En reprenant l'exemple précédent : $v_1^{\downarrow} = \{1, 2\}$, $v_1^{\uparrow} = \{1, 2, 3, 4, 5\}$ et $v_2^{\downarrow} = \{2, 3\}$, $v_2^{\uparrow} = \{1, 2, 3, 6, 7\}$, on obtient $Val = \{2\}$. On sélectionne ensuite un élément aléatoire dans cet ensemble : *val* qu'on va retirer de l'ensemble minimum de la seconde variable. Puis on retire de l'ensemble maximum de la première variable celui nouvellement créé.
Ainsi après filtrage on obtient les nouveaux ensembles suivant : $v_1^{\uparrow} = \{1, 2, 4, 5\}$ et $v_2^{\uparrow} = \{2, 3, 6, 7\}$

Le dernier algorithme de filtrage reprend l'idée de celui établit pour la première contrainte. On cherche à réduire au mieux l'ensemble maximum de chaque variable. Afin d'y arriver on va considérer l'ensemble minimum de l'intersection des deux autres variables pour les retirer de l'ensemble maximum de la première variable et ainsi conserver une intersection vide.
En définissant les ensembles suivant : $v_1^{\downarrow} = \{1, 2\}$, $v_1^{\uparrow} = \{1, 2, 3, 4, 5\}$; $v_2^{\downarrow} = \{2, 3\}$, $v_2^{\uparrow} = \{1, 2, 3, 6, 7\}$ et $v_3^{\downarrow} = \{1, 3\}$, $v_3^{\uparrow} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
En filtrant, on peut réduire les ensembles maximum de chaque variable de la façon suivante :
$v_1^{\uparrow} = \{1, 2, 4, 5\}$, $v_2^{\uparrow} = \{2, 3, 6, 7\}$, $v_3^{\uparrow} = \{1, 3, 4, 5, 6, 7, 8, 9\}$

3.5 Analyse et discussions :

Ce solveur fonctionne correctement et trouve des résultats. Cependant des améliorations sont possibles.

- Dans le solveur générique, on peut détecter les contraintes closes, qui sont toujours vraies, et ne plus les ajouter dans la pile. exemple : $(v_1 \cap v_2) = \varnothing$ avec $(v_1^{\uparrow} \cap v_2^{\uparrow}) = \varnothing$ alors on peut désactiver la contrainte.
- Dans le Branch and Bound, lors d'un nouvel appel du solver générique, on peut ajouter seulement les contraintes ayant la variable modifiée dans le branchement.

4 Résultats :

4.1 Résultats pour SGP 4-4-4 :

Ensembliste 4-4-4		
Semaine 1	Groupe 1	[1,2,3,4]
	Groupe 2	[5,6,7,8]
	Groupe 3	[9,10,11,12]
	Groupe 4	[13,14,15,16]
Semaine 2	Groupe 1	[1,5,9,13]
	Groupe 2	[2,7,10,16]
	Groupe 3	[3,8,11,14]
	Groupe 4	[4,6,12,15]
Semaine 3	Groupe 1	[1,7,11,15]
	Groupe 2	[2,5,12,14]
	Groupe 3	[3,6,9,16]
	Groupe 4	[4,8,10,13]
Semaine 4	Groupe 1	[1,6,10,14]
	Groupe 2	[2,8,9,15]
	Groupe 3	[3,7,12,13]
	Groupe 4	[4,5,11,16]

Ensembliste	0,091 s
FD-Gurobi	0.313877 s
SAT-z3	0.04 s
SAT-Minisat	1.6285 s
Branch And Bound	0.077761 s

4.2 Résultats pour SGP 5-5-5 :

Ensembliste					
Semaine 1	Groupe 1	[1,2,3,4,5]	**Semaine 4**	Groupe 1	[1,8,13,18,23]
	Groupe 2	[6,7,8,9,10]		Groupe 2	[2,6,15,17,24]
	Groupe 3	[11,12,13,14,15]		Groupe 3	[3,10,11,19,22]
	Groupe 4	[16,17,18,19,20]		Groupe 4	[4,9,12,16,25]
	Groupe 5	[21,22,23,24,25]		Groupe 5	[5,7,14,20,21]
Semaine 2	Groupe 1	[1,6,11,16,21]	**Semaine 5**	Groupe 1	[1,7,14,17,22]
	Groupe 2	[2,9,13,20,22]		Groupe 2	[2,10,14,16,23]
	Groupe 3	[3,8,14,17,25]		Groupe 3	[3,9,15,18,21]
	Groupe 4	[4,7,15,19,23]		Groupe 4	[4,8,11,20,24]
	Groupe 5	[5,10,12,18,24]		Groupe 5	[5,6,13,19,25]
Semaine 3	Groupe 1	[1,9,14,19,24]			
	Groupe 2	[2,7,11,18,25]			
	Groupe 3	[3,6,12,20,23]		**Temps**	**0.134 s**
	Groupe 4	[4,10,13,17,21]			
	Groupe 5	[5,8,15,16,22]			

Ensembliste	0.134 s
FD-Gurobi	6.227383 s
SAT-z3	7.00 s
SAT-Minisat	Time Out
Branch And Bound	1.503023 s

4.3 Résultats pour STS 6 :

FD		
Semaine 1	Terrain 1	[1, 2]
	Terrain 2	[3, 4]
	Terrain 3	[5, 6]
Semaine 2	Terrain 1	[1, 5]
	Terrain 2	[3, 6]
	Terrain 3	[2, 4]
Semaine 3	Terrain 1	[4, 6]
	Terrain 2	[2, 5]
	Terrain 3	[1, 3]
Semaine 4	Terrain 1	[3, 5]
	Terrain 2	[1, 4]
	Terrain 3	[2, 6]
Semaine 5	Terrain 1	[2, 3]
	Terrain 2	[1, 6]
	Terrain 3	[4, 5]
Temps	0.032309 s	

Ensembliste	0.032309 s
FD-Gurobi	0.032309 s
SAT-z3	0.00 s
SAT-Minisat	0.007741 s
Branch And Bound	0.364210

4.4 Résultats pour STS 8 :

FD					
Semaine 1	Terrain 1	[1, 2]	**Semaine 5**	Terrain 1	[5, 7]
	Terrain 2	[3, 4]		Terrain 2	[1, 3]
	Terrain 3	[5, 6]		Terrain 3	[4, 8]
	Terrain 4	[7, 8]		Terrain 4	[2, 6]
Semaine 2	Terrain 1	[6, 7]	**Semaine 6**	Terrain 1	[1, 8]
	Terrain 2	[5, 8]		Terrain 2	[1, 8]
	Terrain 3	[2, 3]		Terrain 3	[3, 7]
	Terrain 4	[1, 4]		Terrain 4	[4, 6]
Semaine 3	Terrain 1	[3, 6]	**Semaine 7**	Terrain 1	[2, 4]
	Terrain 2	[4, 7]		Terrain 2	[6, 8]
	Terrain 3	[2, 8]		Terrain 3	[1, 7]
	Terrain 4	[1, 5]		Terrain 4	[3, 5]
Semaine 4	Terrain 1	[3, 8]			
	Terrain 2	[1, 6]			
	Terrain 3	[4, 5]			
	Terrain 4	[2, 7]			
		Temps	3.760724 s		

Ensembliste	0.201 s
FD-Gurobi	3.760724 s
SAT-z3	0.01 s
SAT-Minisat	0.032834 s
Branch And Bound	87.801295 s

4.5 Résultats pour STS 10 :

Ensembliste	1.05 h
FD-Gurobi	Time Out
SAT-z3	0.02 s
SAT-Minisat	0.019324 s
Branch And Bound	Time Out

4.6 Résultats pour STS 12 :

Ensembliste	Time Out
FD-Gurobi	Time Out
SAT-z3	59.70 s
SAT-Minisat	88.2098 s
Branch And Bound	Time Out

4.7 Analyse des Résultats

Analyse :

- SGP
  - Problème plus combinatoire que le STS donc plus faciles à résoudre avec un filtrage
  - On observe que l'on peut casser plus de symétrie que sur le STS
- STS
  - Beaucoup de clauses à deux litéraux donc les solveurs sat sont plus efficaces

Améliorations :

- SGP
  - On pourrait modifier le modèle en générant les groupes possibles pour les utiliser comme valeurs de variables ensembliste (il ne faudrait générer que $\frac{n(n-1)}{k(k-1)}$ avec n le nombre de joueurs et k le nombre de joueurs dans un groupe).

5 Conclusion :

On remarque que :

- Le modèle ensembliste est le meilleur sur le SGP.
- Le modèle SAT est le meilleur sur le STS.
- Z3 est meilleur que Minisat en général
- Notre solveur artisanal talonne minizinc sur le SGP
- Notre solveur artisanal est faible sur le STS a cause de son filtrage

6 Commandes d'exécution :

Pour les modèles Ensembliste, il suffit d'ouvrir les fichiers dans minizinc pour les exécuter.

`SGP.mzn`

Pour les fichier FD, il faut julia 1.2 avec les paquets :

```
using JuMP, Gurobi, MathOptInterface, LinearAlgebra
```

Attention, il faut une licence de Gurobi pour résoudre avec Gurobi. La commande pour résoudre est :

```
include("SGP.jl")
```

Pour résoudre les instances sat, il faut d'abord générer les fichiers cnf avec julia 1.2 et la commande :

```
include("SGP_SAT.jl")
```

ensuite, il faut résoudre le problème avec :

```
z3 satsgp.cnf -st
```

Puis il faut copier la solution du terminal dans le fichier res.out ou on peut le générer avec minisat :

```
minisat satsgp.cnf res.out
```

Il faut ensuite interpréter les résultats avec (dans le même environnement julia) :

```
include("satInterpret.jl")
```

Pour lancer le branch and bound il faut utiliser la commande suivante

```
include("SGP_solve.jl")
```

Pour changer l'instance, il faut modifier les paramètres dans le code.

Pour changer de problème il faut remplacer les SGP par STS.

Les résultats obtenus sont dans les fichiers results des deux dossiers SGP et STS.