Programmation par contraintes
Social Golfer Problem

CALLICO Adrien SAUNIER Nicolas POIRIER Achille

Université de Nantes

14 décembre 2022

Présentation du problème

Social Golfer Problem

Données :

- q : nombre de golfeurs
- w : nombre de semaines
- g : nombre de groupes (par semaine)
- p : nombre de golfeurs par groupe

(note : $q = g.p$)

[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]

Sommaire

Sommaire

Variables

Variables :

- $P = \{p_1, ..., p_q\}$: ensemble des joueurs
- $G = \{G_{1,1}, ..., G_{w,g}\}$: ensemble des groupes de joueurs

$G_{i,j}$ représente le j^{eme} groupe de la semaine i.
On a donc $1 \leq i \leq w$ et $1 \leq j \leq g$.

Contraintes

$$
\begin{aligned}
|G_{ij}| &= p & \forall i \in [\![1, w]\!], \forall j \in [\![1, g]\!] \quad (1)\\
G_{ij} \cap G_{ij'} &= \emptyset & \forall i \in [\![1, w]\!],\ \forall j < j' \in [\![1, g]\!] \quad (2)\\
|G_{ij} \cap G_{i'j'}| &\leq 1 & \forall i, i' \in [\![1, w]\!], i \neq i', \forall j, j' \in [\![1, g]\!] \quad (3)
\end{aligned}
$$

Symétrie

Pour ce modèle nous avons identifié les symétries suivantes :

1. On peut interchanger les groupes au sein d'une semaine
2. On peut interchanger les semaines
3. On peut renuméroter les joueurs

Ainsi le nombre de symétries est : $(g!)^w w!(q!)$

Symétrie

Pour la renumérotation des joueurs, il suffit de fixer la numérotation : au lieu de considérer $P = \{p_1, \ldots, p_q\}$, on considère $P = \{1, \ldots, q\}$

Symétrie

On fixe la première semaine dans l'ordre :

Première semaine : [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]

$$((i-1)*p)+j) \in G_{1,i} \qquad \forall i \in [\![1,g]\!], \forall j \in [\![1,p]\!]$$

Symétrie

On fixe le premier groupe de la deuxième semaine :

Première semaine : [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
Premier groupe de la deuxième semaine : [1,4,7]

$$(1 + p * (j - 1)) \in G_{2,1} \qquad \forall j \in [\![1, p]\!]$$

Symétrie

Pour la deuxième semaine, on impose l'appartenance des p joueurs du dernier groupe de la première semaine aux p derniers groupes de la deuxième semaine :

Première semaine : [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
Deuxième semaine : [[?, ?, ?],[?, ?,10][?, ?,11],[?, ?,12]]

$$(g * (p-1) + j) \in G_{2,j} \qquad \forall j \in [\![(g-p+1), g]\!]$$

Symétrie

À partir de la deuxième semaine, on assigne les p premiers joueurs dans les p premiers groupes :

Première semaine : [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
Deuxième semaine : [[1, ?, ?],[2, ?, ?],[3, ?, ?],[?, ?, ?]]

$$j \in G_{i,j} \qquad \forall i \in [\![2, w]\!],\ \forall j \in [\![1, p]\!]$$

Résultats

p	g	w	sans brisage de symétries	avec brisage de symétries
3	4	2	0.335	0.329
3	4	3	0.361	0.343
3	4	4	0.358	0.353
3	5	2	0.337	0.345
3	5	3	0.356	0.343
4	5	2	0.367	0.362
4	5	3	0.423	0.390
4	6	2	0.485	0.366
5	5	2	0.335	0.337
5	5	3	0.311	0.309
5	5	4	18.466	0.327
5	5	5	22.592	0.352
5	5	6	33.118	0.408

Sommaire

Variables

$$x_{ijkl} = \begin{cases} 1, \text{si le joueur } i \text{ joue en position } j \text{ dans le groupe } k \text{ la semaine } l \\ 0, \text{sinon} \end{cases}$$

Modèle (naïf)

$$\bigwedge_{i=1}^{q} \bigwedge_{l=1}^{w} \bigvee_{j=1}^{p} \bigvee_{k=1}^{g} x_{ijkl} \tag{1}$$

Modèle (naïf)

$$\bigwedge_{i=1}^{q} \bigwedge_{l=1}^{w} \bigwedge_{j=1}^{p-1} \bigwedge_{k=1}^{g} \bigwedge_{m=j+1}^{p} \neg x_{ijkl} \vee \neg x_{imkl} \qquad (2)$$

Modèle (naïf)

$$\bigwedge_{i=1}^{q} \bigwedge_{l=1}^{w} \bigwedge_{j=1}^{p} \bigwedge_{k=1}^{g-1} \bigwedge_{m=k+1}^{g} \bigwedge_{n=1}^{p} \neg x_{ijkl} \vee \neg x_{inml} \tag{3}$$

Modèle (naïf)

$$\bigwedge_{l=1}^{w} \bigwedge_{k=1}^{g} \bigwedge_{j=1}^{p} \bigvee_{i=1}^{q} x_{ijkl} \tag{4}$$

Modèle (naïf)

$$\bigwedge_{l=1}^{w} \bigwedge_{k=1}^{g} \bigwedge_{j=1}^{p} \bigwedge_{i=1}^{q-1} \bigwedge_{m=i+1}^{q} \neg x_{ijkl} \vee \neg x_{mjkl} \tag{5}$$

Modèle : contrainte de sociabilité

$$\bigwedge_{l=1}^{w-1} \bigwedge_{k=1}^{g} \bigwedge_{m=1}^{q-1} \bigwedge_{n=m+1}^{q} \bigwedge_{k'=1}^{g} \bigwedge_{l'=l+1}^{w} (\neg y_{mkl} \vee \neg y_{nkl}) \vee (\neg y_{mk'l'} \vee \neg y_{nk'l'})$$

$$y_{ikl} \iff \bigvee_{j=1}^{p} x_{ijkl}$$

Pour l'obtenir, on reformule l'implication en disjonction ($P \Rightarrow Q \iff P \vee \neg Q$) et on applique les lois de Morgan.

Symétrie

Les symétries du modèles SAT :

1. On peut interchanger les joueurs au sein d'un groupe
2. On peut interchanger les groupes au sein d'une semaine
3. On peut interchanger les semaines
4. On peut interchanger les joueurs

Le nombre de symétries est : $(p!)^{gw}(g!)^{w}w!(q!)$

Symétrie

On ordonne les joueurs dans les groupes :

$$\bigwedge_{i=1}^{q} \bigwedge_{j=1}^{p-1} \bigwedge_{k=1}^{g} \bigwedge_{l=1}^{w} \bigwedge_{m=1}^{i} \neg x_{ijkl} \vee \neg x_{m(j+1)kl}$$

Symétrie

Ensuite, on ordonne les groupes d'une même semaine, par ordre croissant des premiers joueurs de chaque groupe.

$$\bigwedge_{i=1}^{q} \bigwedge_{k=1}^{g-1} \bigwedge_{l=1}^{w} \bigwedge_{m=1}^{i} \neg(x_{i1kl} \vee x_{m1(k+1)l})$$

Symétrie

Et on ordonne les semaines par ordre croissant du deuxième joueur du premier groupe de chaque semaine :

$$\bigwedge_{i=1}^{q} \bigwedge_{l=1}^{w-1} \bigwedge_{m=1}^{i} \neg\left(x_{i21l} \vee x_{m21(l+1)}\right)$$

Symétrie

De même que pour le modèle ensembliste, on enlève des symétries en fixant des joueurs dans certains groupes.
Fixer la première semaine :

$$\bigwedge_{k=1}^{g} \bigwedge_{j=1}^{p} x_{(kp+j+1),j,k,1}$$

Symétrie

Fixer le premier groupe de la deuxième semaine :

$$\bigwedge_{j=2}^{p} x_{(jp)j12}$$

Symétrie

Fixer les derniers joueurs des p derniers groupes de la deuxième semaine :

$$\bigwedge_{j=p-g+1}^{p} x_{(p(g-1)+j)p(g-p+j)2}$$

Symétrie

Fixer les premiers joueurs des p premiers groupes de chaque semaine :

$$\bigwedge_{l=2}^{w} \bigwedge_{k=1}^{p} x_{k1kl}$$

Résultats

p	g	w	sans brisage de symétries	avec brisage de symétries
3	4	2	0.142206907	0.170753479
3	4	3	0.498437404	0.612516880
3	4	4	0.972768783	0.612516880
3	5	2	0.380823850	0.402282476
3	5	3	1.168224573	1.253794193
4	5	2	2.082559823	2.257179021
4	5	3	6.321175336	6.689319610
4	6	2	4.857353210	4.475549221
5	5	2	8.564805746	7.844327688
5	5	3	35.821343898	27.600924253
5	5	4	128.532162427	114.208053588
5	5	5	361.396530151	368.469740390
5	5	6	Crash	Crash

Sommaire

Structures de données

Variable :

- min : l'ensemble minimum de la variable
- max : l'ensemble maximum de la variable
- card_min : le cardinal minimum de la variable
- card_max : le cardinal maximum de la variable
- univers : l'univers de la variable

Les variables sont stockées dans une liste

Structures de données

Contraintes :

- liste_indices_variables : la liste des indices des variables concernées par la contrainte
- filtrage : la fonction de filtrage à appliquer aux (domaines des) variables

Filtrage

- $v_1 \cap v_2 = \emptyset$ (les groupes d'une même semaine doivent être disjoints)
- $|v_1 \cap v_2| \leq 1$ (sociabilité)

Filtrage intersection vide

Pour $v_1 \cap v_2 = \emptyset$, on retire de l'ensemble maximum de chaque variable l'ensemble minimum de l'autre.

$v_2^\uparrow \leftarrow v_2^\uparrow \setminus v_1^\downarrow$

$v_1^\uparrow \leftarrow v_1^\uparrow \setminus v_2^\downarrow$

Filtrage cardinal inférieur à 1

Pour $|v_1 \cap v_2| \leq 1$, on définit *Inter* l'intersection des ensembles minimum des deux variables.
Soit *Inter* est vide, auquel cas il n'y a rien à filtrer, soit il contient un élément v.

- $v_1^{\uparrow} \leftarrow v_1^{\uparrow} \setminus (v_2^{\downarrow} \setminus v)$
- $v_2^{\uparrow} \leftarrow v_2^{\uparrow} \setminus (v_1^{\downarrow} \setminus v)$

Propagation

Algorithm 1: propagation !(variables, contraintes)

```
C ← stocke toutes les contraintes associées à chaque variable
P ← contient les indices des contraintes
feasible ← true
while il reste des contraintes à propager et que le problème est faisable do
    on dépile une contrainte
    on filtre les domaines des variables concernées par la contrainte
    for chaque variable concernée par la contrainte do
        if valide(variable) then
            if la variable a changé then
                for ctr ∈ C[variable] do
                    push!(P,ctr)
        else
            feasible ← false
return feasible
```

Branch and prune

Algorithm 2: branch_and_prune!(variables, contraintes)

```
faisable ← propagation!(variables, contraintes)
if faisable then
    nonCloses ← variables non closes
    if il reste des variables non closes then
        on branche sur une des variables non closes
        while on n'a pas tout testé et le sous-problème est faisable do
            on copie les variables
            on ajoute une des valeurs possibles à l'ensemble min de la variable sur
             laquelle on a branché
            on résout le sous problème (appel récursif)
            if le sous problème est infaisable then
                on retire la valeur candidat de l'ensemble max de var_branch
        if le sous problème est infaisable then
            variables ← var_copie
        else
            faisable ← propagation!(variables, contraintes)
return faisable
```

Brancher sur la variable la plus proche d'être close

p	g	w	sans brisage de symétries	avec brisage de symétries
3	4	2	0.084936458	0.003543939
3	4	3	0.23066264	0.028514438
3	4	4	14.547025263	4.228137
3	5	2	0.436509853	0.007416181
3	5	3	0.869990061	0.177317006
4	5	2	64.327950907	0.018084951
4	5	3	101.899178069	0.072101976
4	6	2	757.236650508	0.020967814
5	5	2	0.035401851	0.010251618
5	5	3	0.1479803	0.066581473
5	5	4	139.414759117	2.981129436
5	5	5	216.903792642	4.369493788
5	5	6	2050.381534559	3.283116111

Brancher sur la variable touchant le plus de contraintes

p	g	w	sans brisage de symétries	avec brisage de symétries
3	4	2	0.07233353	0.002993781
3	4	3	0.21666242	0.033480252
3	4	4	12.65295004	0.437641403
3	5	2	0.443468615	0.006964418
3	5	3	0.855314173	0.117607571
4	5	2	59.552878025	0.017617706
4	5	3	95.642175864	0.064143038
4	6	2	714.595436453	0.015016308
5	5	2	0.041139826	0.011115361
5	5	3	0.1449457	0.061023429
5	5	4	134.46716426	2.753192971
5	5	5	205.018074857	3.749897352
5	5	6	1954.675881979	1.393180103

Analyse

- Solveur fonctionnel
- temps raisonnables sauf sur quelques instances
- Deuxième stratégie légèrement meilleure sur les instances testées

Pistes d'amélioration :

- Filtrage
- D'autres heuristiques de branchement
- Ecarter les contraintes toujours vraies
- Implémenter d'autres contraintes