GIANG Phuong-Thu, Cécile (3530406)
LENOIR Romain (3670199)
IAMSI - Groupe 2

Compte Rendu du TME 4
Programmation SAT - Générateur de championnat

2021-03-07

1 Exercice 1 - Prise en main glucose et DIMACS

NOTE IMPORTANTE : Nous n'avons pas pu installer Glucose ni aucun autre solveur SAT. Pour le reste du TME, nous nous sommes servis de la librairie python `pycosat'`

Les deux théories de l'exercice 4 de la feuille de TD sont écrites dans les fichiers `ex1_1.cnf` et `ex1_2.cnf`.

Une interprétation satisfiant la première théorie est:
$\{\neg x_1, x_2, \neg x_3, \neg x_4, \neg x_5, x_6 \neg x_7, x_8\}$.

Une interprétation satisfiant la deuxième théorie est:
$\{\neg x_1, x_2, \neg x_3, x_4, x_5, \neg x_6, \neg x_7, \neg x_8, \neg x_9, \neg x_{10}, \neg x_{11}, x_{12}, \neg x_{13}, \neg x_{14}, \neg x_{15}, \neg x_{16}\}$.

La théorie F de l'exercice 2 est écrite dans le fichier `ex1_3.cnf`. Il n'existe aucune interprétation la satisfaisant.

2 Modélisation d'un championnat

Chaque équipe correspond à un numéro entre 0 et $n_e 1$. De même, chaque jour de match correspond à un numéro entre 0 et $n_j 1$.
On considère que le championnat commence un mercredi. Les jours pairs (0 compris), correspondent donc à des mercredis, tandis que les jours impairs correspondent à des dimanches.

On représente par des variables propositionnelles $m_{j,x,y}$ le fait qu'il y ait (ou non) un match entre l'équipe x, jouant à domicile, et l'équipe y au jour j.

Au format DIMACS toutes les variables doivent être numérotées. On propose de coder la variable $m_{j,x,y}$ par v_k, où $k = j \times n_e^2 + x \times ne + y + 1$.

Question 1: *Exprimer en fonction de n_e et n_j le nombre de variables propositionnelles utilisées.*

Les jours j étant à valeur dans $\{0, ..., n_j - 1\}$, et les équipes x et y étant à valeurs dans $\{0, ..., n_e - 1\}$, il existe $n_j \times ne \times (n_e - 1)$ combinaisons de (j, x, y), soit autant de variables propositionelles.

Questions 2 et 3

Nous implémentons deux fonctions dans le fichier `championnat.py`:

- la fonction `codage` qui prend en argument n_e, n_j , j, x, y, et qui renvoie k
- la fonction `decodage` qui retrouve j, x et y à partir de k et de n_e.

3 Génération d'un planning de match

Question 1: Contraintes de cardinalité

Nous implémentons deux fonctions dans le fichier `championnat.py`:

- la fonction `au_moins_un` qui prend en argument une liste de variables `vars`, et qui renvoie la clause au format DIMACS traduisant la contrainte *äu moins une de ces variables est vraie"*
- la fonction `au_plus_un` qui prend en argument une liste de variables `vars`, et qui renvoie les clauses au format DIMACS traduisant la contrainte *äu plus une de ces variables est vraie"* (avec un encodage par paires)

Question 2: Traduction du problème

On s'attache maintenant à traduire le problème en une formule logique propositionnelle, en passant éventuellement par l'intermédiaire de contraintes de cardinalité.

Contrainte C1: "Chaque équipe ne peut jouer plus d'un match par jour "

En utilisant les contraintes de cardinalité, cette contrainte peut se réécrire:

$$\forall x \in \{0, ..., n_e - 1\}, \forall j \in \{0, ..., n_j - 1\} \sum_{y \in \{0,...,n_e-1\}} m_{j,x,y} + m_{j,y,x} \leq 1$$

Nous implémentons une fonction `encoderC1` qui génère ces contraintes pour n_e et n_j données. En exécutant notre fonction sur 3 équipes et 4 jours ($n_e = 3$, $n_j = 4$), nous nous retrouvons avec les 72 contraintes suivantes:

```
-2 -4 0 -2 -3 0 -2 -7 0 -4 -3 0 -4 -7 0 -3 -7 0 -11 -13 0 -11 -12 0 -11 -16 0 -13 -12 0
-13 -16 0 -12 -16 0 -20 -22 0 -20 -21 0 -20 -25 0 -22 -21 0 -22 -25 0 -21 -25 0 -29 -31 0
-29 -30 0 -29 -34 0 -31 -30 0 -31 -34 0 -30 -34 0 -4 -2 0 -4 -6 0 -4 -8 0 -2 -6 0 -2 -8 0
-6 -8 0 -13 -11 0 -13 -15 0 -13 -17 0 -11 -15 0 -11 -17 0 -15 -17 0 -22 -20 0 -22 -24 0
-22 -26 0 -20 -24 0 -20 -26 0 -24 -26 0 -31 -29 0 -31 -33 0 -31 -35 0 -29 -33 0 -29 -35 0
-33 -35 0 -7 -3 0 -7 -8 0 -7 -6 0 -3 -8 0 -3 -6 0 -8 -6 0 -16 -12 0 -16 -17 0 -16 -15 0
-12 -17 0 -12 -15 0 -17 -15 0 -25 -21 0 -25 -26 0 -25 -24 0 -21 -26 0 -21 -24 0 -26 -24 0
-34 -30 0 -34 -35 0 -34 -33 0 -30 -35 0 -30 -33 0 -35 -33 0
```

Contrainte C2: " Sur la durée du championnat, chaque équipe doit rencontrer l'ensemble des autres équipes une fois à domicile et une fois à l'extérieur, soit exactement 2 matchs par équipe adverse "

En utilisant les contraintes de cardinalité, cette contrainte peut se réécrire:

$$\forall x \in \{0, ..., n_e - 1\}, \forall y \in \{0, ..., n_e - 1\} \sum_{j \in \{0,...,n_j-1\}} m_{j,x,y} = 1 \quad \text{et} \quad \sum_{j \in \{0,...,n_j-1\}} m_{j,y,x} = 1$$

Nous implémentons une fonction `encoderC2` qui génère ces contraintes pour n_e et n_j données. En exécutant notre fonction sur 3 équipes et 4 jours ($n_e = 3$, $n_j = 4$), nous nous retrouvons avec les 42 contraintes suivantes:

```
-2 -11 0 -2 -20 0 -2 -29 0 -11 -20 0 -11 -29 0 -20 -29 0 2 11 20 29 0 -3 -12 0 -3 -21 0
-3 -30 0 -12 -21 0 -12 -30 0 -21 -30 0 3 12 21 30 0 -4 -13 0 -4 -22 0 -4 -31 0 -13 -22 0
-13 -31 0 -22 -31 0 4 13 22 31 0 -6 -15 0 -6 -24 0 -6 -33 0 -15 -24 0 -15 -33 0 -24 -33 0
6 15 24 33 0 -7 -16 0 -7 -25 0 -7 -34 0 -16 -25 0 -16 -34 0 -25 -34 0 7 16 25 34 0 -8 -17
0 -8 -26 0 -8 -35 0 -17 -26 0 -17 -35 0 -26 -35 0 8 17 26 35 0
```

Enfin, nous écrivons une fonction `encoder` qui encode toutes les contraintes C1 et C2 pour n_e et n_j données et génère un fichier `cnf` contenant la théorie: fichier `championnat.cnf`.

Question 3: Utilisation du solveur

Le solveur nous renvoie UNSAT: les contraintes de l'énoncé ne permettent pas de résoudre le problème à ce stade.
En effet, avec trois équipes, chaque équipe devra jouer exactement 4 matchs (2 matchs exactement contre chaque équipe adverse, le nombre d'équipes adverses étant de 2), soit une fois tous les jours $(n_j = 4)$.

Or chaque équipe ne pouvant jouer qu'un match au plus par jour, cela est impossible: au 1er jour, deux équipes s'affrontent, la dernière ne peut affronter personne.

Il nous faut au moins $ne!/(ne-2)! = 6$ jours pour que le problème soit solvable. En générant maintenant le fichier `cnf` correspondant au problème avec $n_e = 3$ et $n_j = 6$, le solveur nous rend bien une interprétation satisfiant la théorie. Pour rappel, **nous n'avons pas réussi à installer un solveur SAT et utilisons une librairie python:** `pycosat`. Ce solveur nous renvoie la solution suivante:

```
-1 -2 3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 17 -18 -19 -20 -21 22 -23 -24 -25
-26 -27 -28 29 -30 -31 -32 -33 -34 -35 -36 -37 -38 -39 -40 -41 -42 43 -44 -45 -46 -47
-48 -49 -50 51 -52 -53 0
```

Nous écrivons ensuite une fonction `genere_cnf` qui prend en arguments n_e, n_j et utilise la fonction `encode` pour générer le fichier `cnf` correspondant à la théorie.

Question 4: Décodage

NOTE IMPORTANTE : Pour rappel, nous n'avons pu installer Glucose. Pour le reste du TME, nous nous sommes servis de la librairie python pycosat

Nous étions censés écrire une fonction `decoder` qui prend pour argument un fichier contenant la sortie de l'appel à glucose (plus éventuellement n_j et n_e) et qui traduit le modèle rendu en une solution du problème de planning des matchs affichée lisiblement.

Comme nous ne pouvons pas utiliser Glucose et utilisons à la place `pycosat`, la fonction `decoder` prendra en arguments les éléments suivants:

- le nom d'un fichier `cnf_file` (format .cnf) la théorie à résoudre : il s'agit d'une solution retournée par `genere_cnf`
- n_e et n_j, qui ont servi à générer `cnf_file`
- éventuellement un fichier extérieur donnant le nom des équipes : une par ligne dans leur ordre de numérotation.

La fonction renvoie un dictionnaire `planning` dont les clés sont les jours de match et les valeurs les listes des matchs pour chaque jour (couples (x, y) où x esy l'équipe qui joue à domicile).

Nous avons également ajouté une fonction `affiche_planning` qui prend en argument le dictionnaire `planning` renvoyé par `decoder` et l'affiche de manière un peu plus jolie.

Question 5 - Assemblage final: *Ecrire un programme ou un script qui étant donné n_e, n_j et un fichier de noms d'équipes, affiche un planning des matchs en utilisant les programmes écrits dans les questions précédentes.*

cf. fichier `run.py`

4 Optimisation du nombre de jours

Nous cherchons à déterminer le nombre n_j de jours minimal pour pouvoir planifier tous les matchs de championnat pour un n_e donné.

(Voir la fonction `optimiser_nj`)