# Mini-Projet IBM

## Programmation par contraintes

Guillaume Crognier, Séverine Bonnechère

Mars 2019

MPRO

# Plan

1. Allocation de fréquences

2. Tournoi d'échecs

# Allocation de fréquences

# Tournoi d’échecs

# Plusieurs modélisations possibles

Le problème prend seulement une entrée : $N$, le nombre de joueurs.

On décide soi-même de ce qui va être une donnée ou une variable pour la modélisation parmi les matchs (joueur 1, joueur 2) ou les créneaux (jour, période).

Nous avons testé trois approches :

- **Modèle 1** : les variables sont le numéro du créneau sur lequel est joué un match
- **Modèle 2** : les variables sont la période et le jour d'un match
- **Modèle 3** : les variables sont les matchs à affecter au planning

# Modèle 1 - Variables : créneau d'un match

## Données :

- Liste des matchs à jouer
- Nombre de créneaux dans le planning : $\frac{N(N-1)}{2}$

## Variables :

- Numéro de créneau à affecter à chaque match
  $\rightarrow$ $\frac{N(N-1)}{2}$ variables de domaine $[\![1, \frac{N(N-1)}{2}]\!]$

## Contraintes à satisfaire :

- ***allDifferent***(créneaux)
- Chaque joueur joue une fois par jour
  $\rightarrow$ utilisation du **quotient** créneau / nombre de jour
- Chaque joueur joue au plus deux fois par période
  $\rightarrow$ utilisation du **reste de la division** créneau / nombre de jour

# Modèle 1 - Variables : créneau d'un match

Avantages

$\rightarrow$ Nombre minimal de variables
$\rightarrow$ Domaine des variables minimal
$\rightarrow$ Variables qui permettent d'utiliser ***allDifferent***

Inconvénients

$\rightarrow$ Les divisions euclidiennes ne sont pas compatibles avec les contraintes ***count*** du solveur
$\rightarrow$ Nécessité de passer par une formulation en intention avec des sommes
$\rightarrow$ Perte de performance
$N = 16$ **atteint en** 31**s**

# Modèle 2 - Variables : jour et période d'un match

Idée : exploiter les contraintes de type ***count*** du solveur

Variables :

- Jour de chaque match
  $\rightarrow \frac{N(N-1)}{2} \times (N-1)$ variables de domaine $[\![1, N-1]\!]$
- Période de chaque match
  $\rightarrow \frac{N(N-1)}{2} \times \frac{N}{2}$ variables de domaine $[\![1, \frac{N}{2}]\!]$

## Modèle 2 - Variables : jour et période d'un match

Idée : exploiter les contraintes de type ***count*** du solveur

Variables :

- Jour de chaque match
  $\rightarrow \frac{N(N-1)}{2} \times (N-1)$ variables de domaine $[\![1, N-1]\!]$
- Période de chaque match
  $\rightarrow \frac{N(N-1)}{2} \times \frac{N}{2}$ variables de domaine $[\![1, \frac{N}{2}]\!]$
- Ajout de **variables artificielles** : numéro de créneaux de chaque match ($\frac{N(N-1)}{2}$ variables de domaine $[\![1, \frac{N(N-1)}{2}]\!]$)

Contraintes à satisfaire :

- ***allDifferent***(numéro de créneau)
- Numéro de créneau en bijection avec *(jour, période)* d'un match
- Chaque joueur joue une fois par jour
  $\rightarrow$ ***count*** sur les variables de journée
- Chaque joueur joue au plus deux fois par période
  $\rightarrow$ ***count*** sur les variables de période

# Modèle 2 - Variables : jour et période d'un match

## Avantages

$\rightarrow$ Exploitation des contraintes "leviers" du solveur ***allDifferent*** et ***count***

$\rightarrow$ Temps de résolution amélioré : $N = 16$ **atteint en** 14**s**

## Inconvénients

$\rightarrow$ Plus de variables que dans le modèle 1

$\rightarrow$ Écrire le *allDifferent* **en intention** sur les variables de jours et périodes est **peu performant**

$\rightarrow$ Nécessité de créer des variables artificielles sur lesquelles on applique le ***allDifferent*** et les mettre en bijection avec les variables de décision

# Modèle 3 - Variables : matchs à affecter au planning

Idée : Prendre le problème dans l'autre sens : les créneaux deviennent les données, et les matchs deviennent les variables de décision

# Modèle 3 - Variables : matchs à affecter au planning

Idée : Prendre le problème dans l'autre sens : les créneaux deviennent les données, et les matchs deviennent les variables de décision

Variables :

- Joueur 1 de chaque créneau : $\frac{N(N-1)}{2}$ variables de domaine $[\![1, N-1]\!]$
- Joueur 2 de chaque créneau : $\frac{N(N-1)}{2}$ variables de domaine $[\![2, N]\!]$
- Ajout de **variables artificielles** : numéro de match ($\frac{N(N-1)}{2}$ variables de domaine $[\![1, \frac{N(N-1)}{2}]\!]$)

Contraintes à satisfaire :

- ***allDifferent***(numéro de match)
- Numéro de match en bijection avec *(J1, J2)* d'un créneau
- Chaque joueur joue une fois par jour
  $\rightarrow$ ***count*** sur les joueurs d'une journée
- Chaque joueur joue au plus deux fois par période
  $\rightarrow$ ***count*** sur les joueurs d'une période

# Modèle 3 - Variables : matchs à affecter au planning

**Avantages**

$\rightarrow$ Exploitation des contraintes "leviers" du solveur ***allDifferent*** et ***count***
$\rightarrow$ Journée factice plus facilement implémentable que dans le modèle 2
$\rightarrow$ $N = 16$ **atteint en** 13**s**

**Inconvénients**

Nécessité de créer des variables artificielles sur lesquelles on applique le ***allDifferent*** et les mettre en bijection avec les variables de décision
$\rightarrow$ Taille de l'arbre d'exploration plus grande que dans le modèle 1

# Ajout d’une journée factice

- Gain de performance pour les modélisations 1 et 3
- Pas d’amélioration notable pour la modélisation 2 : l’ajout de la journée factice nécessite de créer un autre ensemble de matchs et de variables, ce qui contre-balance le gain apporté par la contrainte d’égalité

# Paramétrage des stratégies de recherche

- Plus on utilise des contraintes-clés du solveur, mieux on peut paramétrer la recherche
- *extended* sur les niveaux d'inférence de *allDifferent* et *count* permet d'améliorer le temps de résolution sur tous les modèles. L'amélioration est d'autant plus notable lorsque la modélisation utilise au maximum les contraintes-clés du solveur (modèles 2 et 3).
  $\rightarrow$ *extended* traduit peut-être une fermeture arc-consistance plus fréquente ou plus complète.
  L'amélioration de la rapidité de résolution signifie que le problème est très contraint, et passer en mode *extended* permet de fermer plus vite une branche dans le backtrack.

# Conclusion

$\rightarrow$ **En théorie** : minimiser le nombre de variables et leur domaine devrait améliorer la vitesse de résolution (moins de variables sur lesquelles brancher dans le backtrack, plus petite taille de l'arbre d'exploration)
$\rightarrow$ **En pratique** : utiliser au maximum les contraintes-clés du solveur, quitte à ajouter des variables artificielles, permet de gagner en performance (dans une certaine mesure)