Projet de recherche - Planification d'horaires sportifs

 Lien GitHub

Mikaël Gélinas
Bruno Kinder Almentero
Samuel Lévesque

Pour le cours IFT-7020
Optimisation combinatoire

Présenté le 17 avril 2020 au professeur

Claude-Guy Quimper

Département d'informatique et de génie logiciel
Faculté des sciences et de génie
Université Laval

Faculté des sciences et de génie
Université Laval
Hiver 2020

1 Introduction

Les événements des ligues sportives professionnelles tels que les Jeux olympiques, la Coupe du monde de football, le Super Bowl, les grands tournois de golf et de tennis et, plus récemment, les eGames entraînent des légions de fans et de passionnés. Par conséquent, d'énormes sommes d'argent sont associées à ces événements. Clairement, un aspect clé des événements sportifs est la capacité à générer des horaires qui optimisent les problèmes logistiques et qui sont considérés comme équitables pour toutes les équipes impliquées.

Trouver le meilleur calendrier des matchs est une tâche difficile avec de multiples décideurs, contraintes et objectifs impliquant la logistique, l'organisation, les revenus et les questions d'équité. De bons agencements de parties et de lieux sont importants pour maximiser les revenus, assurer l'attractivité des jeux et maintenir l'intérêt des médias et des fans. De bons horaires peuvent avoir des implications financières importantes et interférer dans la performance de chaque équipe participant à un tournoi.

D'un autre côté, les ligues amateurs n'ont généralement pas accès aux mêmes montants, mais le nombre de tournois et de participants peut être très important, nécessitant également un effort considérable de coordination et de logistique.

Nous présentons ici des modèles de programmation linéaire flexibles permettant la planification d'horaires de tournois sportifs de tous genre qui permettent de gérer facilement un grand nombre de contraintes spécifiques aux organisateurs de tournoi.

2 Description du problème

Les tournois *round-robin* doubles (DRRT) et simples (SRRT) où chaque équipe joue contre toutes les autres équipes une fois à domicile et une fois à l'extérieur sont des classes bien connues des horaires des ligues sportives. Le format étant très répandu, de nombreuses recherches ont été menées pour programmer les DRRT de manière efficace et équitable. Des approches de programmation par contraintes (CP) qui décomposent le problème de planification des DRRT ont été proposée par Henz [7] et Schaerf [12].

Cependant, bien que *round-robin* a son importance pour les compétitions professionnelles, bien d'autres préoccupations prévalentes dans les ligues amateurs. Par exemple, les parties pourraient être très rapprochées dans le temps et l'organisateur d'un tournoi pourrait vouloir un temps minimum entre deux parties pour laisser le temps aux joueurs de se reposer. Aussi, plusieurs équipes peuvent avoir le même entraîneur ce qui empêche ces équipes de jouer en même temps dans les lieux différents. Une grande partie de la littérature sur l'organisation des ligues sportives concerne les ligues professionnelles. Knust [8] et Goossens & Spieksma [6] accordent une attention particulière à la planification des ligues de tennis de table non commerciales et des ligues de football amateur. Plusieurs particularités sont prises en compte, telles que l'accès limité aux lieux et la disponibilité limitée des équipes.

Russell et Urban [11] se sont attaqué à un problème similaire au nôtre où le tournoi se déroule sur plusieurs sites qui ne sont associés à aucun des concurrents. Par contre, le patron local-visiteur traité pour eux a moins d'importance dans notre contexte. Certaines caractéristiques considérées pertinentes pour les tournois professionnels ont peu ou aucune importance dans un contexte amateur. Briskorn et Drexl [2] proposent une approche (*branch and bound*) pour trouver des tournois à coût minimum déterminé par une matrice de coûts parmi ceux ayant le nombre minimum de pauses. Cheung [5] présente une solution pour le problème DRRT telle que la distance totale parcourue par toutes les équipes est minimisée. Ces deux approches illustrent bien le genre de problème d'optimisation que nous souhaitons attaquer.

Dans ce projet, on se concentre sur l'organisation de tournois sportifs inspirés sur des cas réels vécus par l'entreprise de gestion sportive Kreezee. Ces tournois sont généralement organisés pour un grand nombre d'équipes différentes (allant jusqu'à plusieurs centaines) séparées dans des divisions distinctes de petite taille (4 à 6 équipes). Les parties de ces tournois sont jouées dans plusieurs lieux comportant chacune des

contraintes de disponibilité distinctes. Certaines équipes partagent le même entraîneur et ne peuvent donc pas jouer simultanément. Également, chaque équipe a des préférences quant au moment où elle préfère jouer. Finalement, pour laisser un temps de repos aux joueurs il est important de laisser une période de repos entre deux parties de chaque équipe du tournoi. Notre modèle de programmation par contraintes vise donc à concilier ces multiples limitations pour concevoir un horaire sportif plaisant à tout le monde.

Puisqu'il est souvent impossible de respecter toutes les préférences des différents acteurs, nous utilisons une table de coûts permettant de pénaliser les cas où l'horaire généré brime l'une des préférences d'une équipe. Notre modèle vise donc à minimiser ce coût plutôt qu'à chercher une solution parfaite qui souvent est impossible.

Par exemple, prenons un cas simple avec seulement 2 divisions de 3 et 2 équipes respectivement à planifier dans une grille horaire de 5 périodes et 2 lieux. Les parties à planifier se trouvent dans la table 1.

Table 1 – Parties à planifier pour l'exemple de tournoi simplifié

Numéro de partie	Division	Équipe 1	Équipe 2
1	1	A	B
2	1	A	C
3	1	B	C
4	2	D	E

Également, la table 2 présente les coûts associés à la planification de chaque équipe à chaque période.

Table 2 – Table de coûts pour l'exemple du tournoi simplifié

	Période				
Équipe	**1**	**2**	**3**	**4**	**5**
1	0	4	0	1	0
2	1	0	3	0	5
3	5	0	3	0	1
4	3	0	0	0	0
5	0	0	0	4	0

Dans le cas où le lieu 1 n'est pas disponible aux périodes 1, 3 et 5 et que le lieu 2 n'est pas disponible à la période 4, un horaire optimal de coût 8 est donné dans la table 3, où les chiffres dans la table sont les numéros de partie de la table 1 :

Table 3 – Horaire optimal pour l'exemple de tournoi simplifié

	Période				
Lieu	**1**	**2**	**3**	**4**	**5**
1	0	4	0	0	0
2	1	0	3	0	2

Évidemment, ce problème devient vite très complexe à mesure qu'un augmente la taille des tournois et on comprend vite pourquoi cette tâche colossale de planification peut prendre plusieurs jours aux organisateurs de tournoi.

3 Approches proposées

Pour résoudre toutes les contraintes de notre problème et obtenir la solution convenant à un maximum d'équipes, nous avons décidé d'utiliser un modèle de programmation par contraintes. Pour certaines contraintes du problème, nous avons testé plusieurs formulations mathématiques pour les implémenter afin de voir leur impact sur les temps de calcul et sur le nombre de noeuds développés par notre modèle.

3.1 Définition des variables

Afin de modéliser les nombreuses composantes de notre problématique, nous définissons les variables suivantes. N_p : nombre de périodes du tournoi, N_v : nombre de lieux disponibles (*venue*), N_t : nombre d'équipes jouant dans le tournoi (*team*), N_g : nombre de parties à jouer dans le tournoi (*game*), N_c : nombre de coachs du tournoi. On définit également les ensembles de valeurs suivants :

$\mathcal{V} = \{1, ..., N_v\}$: Ensemble des lieux disponibles
$\mathcal{P} = \{1, ..., N_p\}$: Ensemble des périodes disponibles
$\mathcal{G} = \{1, ..., N_g\}$: Ensemble des parties à jouer
$\mathcal{T} = \{1, ..., N_t\}$: Numéro des équipes du tournoi
$\mathcal{T}_n$: Énumération des noms d'équipes du tournoi
$\mathcal{C}_n$: Énumération des noms de coaches du tournoi

Également, plusieurs tables permettant d'encoder les différentes contraintes du problème sont données en entrée de notre modèle. Ces matrices sont les suivantes :

$\mathbf{V}_{N_v \times N_p}$: Disponibilité des lieux par période
$\mathbf{V}[v, p] \in \{Vrai, Faux\} \forall v \in \mathcal{V}, p \in \mathcal{P}$
$\mathbf{C}_{N_t}$: Entraîneur de chaque équipe du tournoi
$\mathbf{C}[t] \in \mathcal{C}_n, \forall t \in \mathcal{T}$
$\mathbf{T}_{N_t \times N_p}$: Coût associé à chaque période par équipe
$\mathbf{T}[t, p] \in \{0, 1, 2, 3, 4, 5\}, \forall t \in \mathcal{T}_n, p \in \mathcal{P}$
$\mathbf{G}_{N_g \times 2}$: Parties à planifier dans le tournoi
$\mathbf{G}[g, i] \in \mathcal{T}_n, \forall g \in \mathcal{G}, i \in \{1, 2\}$

Finalement, les variables libres de notre problème sont les suivantes :

$\mathbf{S}_{N_v \times N_p}$: Horaire final du tournoi
$\mathbf{S}[v, p] \in \{0\} \cup \mathcal{G}, \forall v \in \mathcal{V}, p \in \mathcal{P}$
$\mathbf{A}_{N_t \times N_v \times N_p}$: Matrice indiquant si une équipe joue à un lieu/période donné
$\mathbf{A}[t, v, p] \in \{Vrai, Faux\}, \forall t \in \mathcal{T}_n, v \in \mathcal{V}, p \in \mathcal{P}$
$\mathbf{B}_{N_c \times N_p}$: Matrice indiquant le nombre de parties de chaque coach par période
$\mathbf{B}[c, p] \in \mathbb{N}_0, \forall c \in \mathcal{C}_n, p \in \mathcal{P}$
$\mathbf{X}_{N_t \times N_p}$: Coût associé à la planification de chaque équipe/période
$\mathbf{X}[t, p] \in \mathbb{N}_0, \forall t \in \mathcal{T}_n, p \in \mathcal{P}$
$\mathcal{X}$: Coût de planification total pour l'horaire du tournoi
$\mathcal{X} \in \mathbb{N}_0$

Le modèle a $O(N_v \times N_p + N_t \times N_p + 2N_g)$ paramètres d'entrée. Le nombre de parties à jouer N_g est directement lié au nombre d'équipes N_t et au nombre de divisions. Pour ce rapport, le nombre de parties d'une division est défini selon un modèle *round-robin* et le nombre total de parties à jouer est la somme de ces parties. Toutefois, le modèle permet d'entrer n'importe quelle combinaison de matchs à planifier par division. Le modèle a $O(N_t \times N_v \times N_p)$ variables libres.

3.2 Contraintes partagées

Nous présentons ici l'ensemble d'équations que nous utilisons pour imposer les contraintes de notre problème. Cette section détaille toutes les contraintes qui sont communes aux 4 modèles étudiés.

$$\neg \mathbf{V}[v,p] \implies \mathbf{S}[v,p] = 0, \forall v \in \mathcal{V}, \forall p \in \mathcal{P} \tag{1}$$

$$\mathbf{A}[t,v,p] \iff \mathbf{S}[v,p] > 0 \wedge (\mathbf{G}[\mathbf{S}[v,p],1] = t \vee \mathbf{G}[\mathbf{S}[v,p],2] = t), \forall t \in \mathcal{T}_n \forall v \in \mathcal{V}, \forall p \in \mathcal{P} \tag{2}$$

$$\sum_{A \in \mathcal{V}} \mathbf{T}[t,v,p] <= 1, \forall t \in \mathcal{T}_n, \forall p \in \mathcal{P} \tag{3}$$

$$\mathbf{X}[t,p] = \sum_{v \in \mathcal{V}} \mathbf{A}[t,v,p] * \mathbf{T}[t,p], \forall t \in \mathcal{T}_n, \forall p \in \mathcal{P} \tag{4}$$

$$\mathbf{C}[p] = \sum_{v \in \mathcal{V}, t \in \mathcal{T}_n} \mathbf{A}[t,v,p], \forall p \in \mathcal{P} \tag{5}$$

$$\mathbf{C}[p] \leq 1, \forall p \in \mathcal{P} \tag{6}$$

L'équation (1) permet de s'assurer qu'aucune partie n'est planifiée à un moment où il n'y a pas de disponibilité. La contrainte (2) permet de remplir la matrice de variables $\mathbf{A}$. La contrainte (3) impose que chaque équipe ne peut jouer au maximum qu'un match à chaque période. La sommation (4) permet de remplir la matrice de coûts $\mathbf{X}$. La contrainte (5) permet de remplir la matrice $\mathbf{B}$. La contrainte (6) assure qu'aucun coach ne doive se séparer entre deux de ses équipes. Le modèle a $O(N_t \times N_v \times N_p)$ contraintes.

3.3 Contraintes spécifiques

Cette section présente les spécificités des 4 modèles considérés. Ces 4 modèles sont pour la plus grande partie identiques, mais ont des formulations différentes pour deux conditions clés du problème, soit la planification de tous les matchs et l'obligation d'avoir une pause entre deux matchs pour toutes les équipes.

3.3.1 Planification de tous les matchs

Deux jeux d'équations ont été testés pour imposer le fait que tous les matchs doivent être mis à l'horaire une seule fois. Une première formulation est composée des équations (7) et (8) alors que la deuxième formulation est déterminée par l'équation (9).

$$\text{AllDifferent\_Except\_0}(\mathbf{S}) \tag{7}$$

$$\sum_{v \in \mathcal{V}, p \in \mathcal{P}} \mathbb{1}_{\{\mathbf{S}[v,p] > 0\}} = N_g \tag{8}$$

$$\sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} \mathbb{1}_{\mathbf{S}[v,p] = g} = 1, \forall g \in \mathcal{G} \tag{9}$$

3.3.2 Obligation d'avoir une pause entre deux parties

La première formulation mathématique qui permet d'imposer une pause entre deux parties pour chaque équipe utilise les automates à états finis. En effet, on utilise la contrainte `Regular` de Minizinc avec un automate défini pour le nombre de périodes de pauses nécessaire. Plus formellement, on doit respecter l'équation (10). Pour imposer des pauses de 1 période, on utilise l'automate présenté dans la figure 1. Pour une pause de deux périodes, on utilise plutôt l'automate de la figure 2. Au besoin, il est facile de construire par induction un automate pour un plus grand nombre de périodes de repos.

$$\text{Regular}\left(\left[\sum_{v\in\mathcal{V}}\mathbf{T}[t,v,p]\mid p\in\mathcal{P}\right],\right),\forall t\in\mathcal{T}_n \tag{10}$$

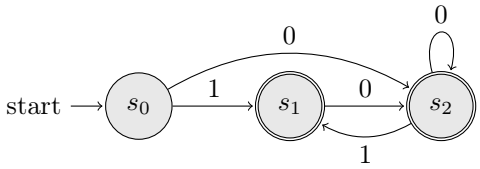


Figure 1 – Automate qui impose au minimum une pause entre matchs

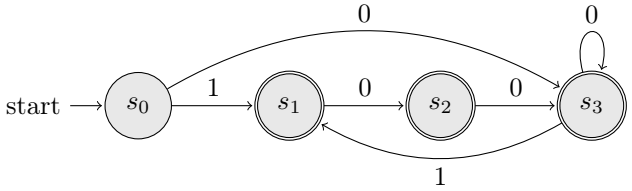


Figure 2 – Automate qui impose au minimum deux pause entre matchs

Une deuxième formulation de ce problème consiste à utiliser directement la matrice $\mathbf{A}$. Pour imposer une pause minimale de i périodes entre deux parties, on peut ajouter la contrainte (11) au modèle

$$\sum_{v\in\mathcal{V}}\mathbf{A}[t,v,p]>0\implies\sum_{t\in\{t+1,\ldots,t+i\}}\sum_{v\in\mathcal{V}}\mathbf{A}[t,v,p]=0,\forall t\in\mathcal{T}_n,p\in\{1,\ldots,N_p-i\} \tag{11}$$

4 Protocole d'expérimentation

Pour comparer nos 4 modèles, nous avons conçu un générateur de scénarios aléatoires qui nous permet de simuler des tournois fictifs de taille variable. Nous souhaitons tester chacun de nos modèles sur des scénarios de taille croissante afin d'observer l'évolution du temps de calcul pour chacun d'eux et voir si un des modèles se démarque en résolvant des problèmes de taille supérieure dans un même intervalle de temps.

Étant donné le grand nombre de cas différents sur lesquels notre modèle pourrait être utilisé, mous ne pouvions évidemment pas tester tous les scénarios de tournois sportifs existants. Nous avons donc choisi des scénarios plus proches de la réalité en nous basant sur des exemples de tournois réels. Les scénarios que nous avons générés ont tous 4 lieux différents et un nombre de périodes égal à deux fois le nombre d'équipes. Les équipes sont toutes séparées dans des divisions de 5 équipes et chaque équipe a son propre entraîneur (aucun entraîneur partagé entre deux équipes ou plus). En suivant ce schéma de construction nous générons des scénarios de façon aléatoire en incrémentant à chaque fois le nombre d'équipe de 5 et résolvons le scénario avec chacun de nos 4 modèles en notant le temps de calcul, le nombre de noeuds développés et le nombre de *nogoods*.

Voici un résumé des quatre modèles étudiés :

— **ADR** : modèle utilisant les contraintes (7) et (8) pour assurer la présence de tous les matchs dans l'horaire et la contrainte (10) pour forcer les pauses entre deux parties.

— **ADI** : modèle utilisant les contraintes (7) et (8) pour assurer la présence de tous les matchs dans l'horaire et la contrainte (11) pour forcer les pauses entre deux parties.

— **SCR** : modèle utilisant la contrainte (9) pour assurer la présence de tous les matchs dans l'horaire et la contrainte (10) pour forcer les pauses entre deux parties.

— **SCI** : modèle utilisant la contrainte (9) pour assurer la présence de tous les matchs dans l'horaire et la contrainte (11) pour forcer les pauses entre deux parties.

Avec le meilleur des 4 modèles, nous faisons également d'autres expérimentations pour tester l'impact des autres dimensions d'un tournoi sportif sur le temps de calcul de notre modèle.

Pour ce faire, nous utilisons un scénario de base avec 50 équipes réparties en 10 divisions, 100 périodes pour chacun de 4 lieux de tournois et 1 coach par équipe. À partir de cette configuration de base, on fait varier une dimension du tournoi à la fois dans le but d'évaluer son impact sur le temps de calcul.

Dans un premier temps, on génère des scénarios aléatoires en faisant diminuer le nombre d'entraîneurs du tournoi à chaque itération. Cela force plusieurs équipes a avoir le même entraîneur et complexifie le problème en ajoutant plus de contraintes. Nous avons mesuré le temps de calcul nécessaire pour trouver une solution optimale pour des cas ayant de 38 à 50 entraîneurs (toujours avec nos 50 équipes).

Ensuite, on évalue l'effet du nombre minimum de pauses requises entre deux matchs d'une même équipe en incrémentant la pause d'une unité à chaque itération en suivant la même méthodologie. Nous avons testé tous les nombres de pauses minimales entre 1 et 10 pour notre même configuration de base.

Après plusieurs expérimentations sur un scénario de base avec 45 équipes, 90 périodes, 45 entraîneurs et 4 lieux, nous avons décidé d'utiliser le solveur Chuffed de Minizinc avec le mode *free search* qui nous offre les meilleures performances. Également, nous forçons un branchement sur les variables de coût en premier en commençant par les plus petites valeurs. En procédant ainsi, le solveur obtient un coût global quasi optimal dès qu'il trouve une solution, ce qui réduit le temps de calcul nécessaire pour obtenir le meilleur horaire possible. Ces configurations de solveur sont utilisées pour toutes les expérimentations décrites ci-haut.

5 Résultats

La figure 3 présente les résultats obtenus pour les tests où nous faisions varier la complexité du modèle pour les quatre modèles étudiés.

La figure 4 présente la performance de notre meilleur modèle selon le nombre de pauses requises entre deux parties.

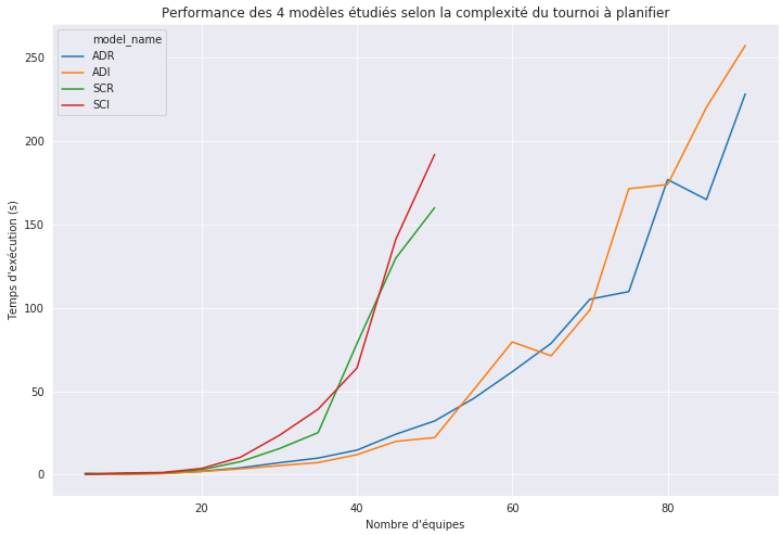
Finalement, la figure 5 présente les résultats obtenus en faisant le nombre d'entraîneurs du tournoi à planifier pour notre modèle le plus performant.

La table 4 montre les temps de calcul pour des scénarios de grande taille où le temps de calcul est très élevé.

Table 4 – Temps de calcul du modèle ADR pour des grandes instances de tournoi planifier

nombre d'équipe	**temps (secondes)**	**temps**
150	4106	1h14
180	10520	2h55
200	13862	3h51
220	31985	8h53
240	60342	16h45
260	90087	25h01

Figure 3 – Performance des 4 modèles étudiés selon la complexité du tournoi à planifier



6 Discussion

Tout d'abord, on analysant les résultants de la figure 3, on remarque une grande différence de performance entre les modèles utilisant la contrainte *AllDifferent_Except_0* et ceux qui utilisent les contraintes de sommes conditionnelles pour s'assurer que toutes les parties soient mises à l'horaire exactement une fois.

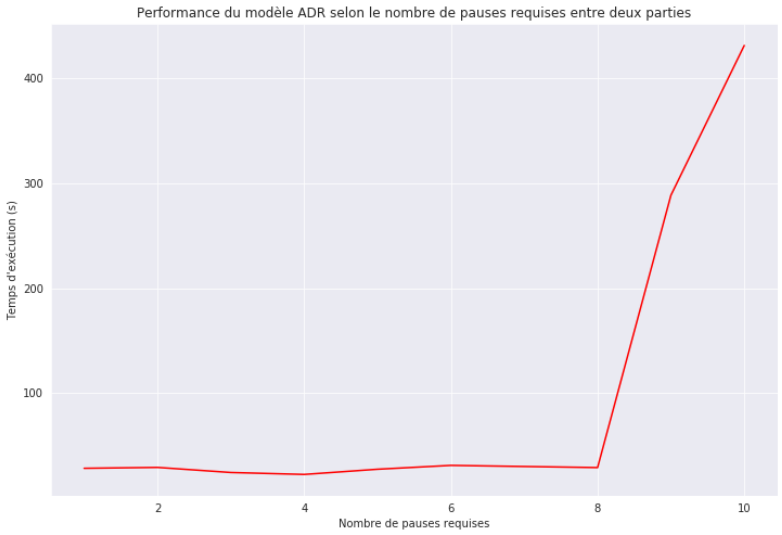
Les deux contraintes étudiées pour gérer le nombre de pauses entre les parties semblent avoir une importance plus limitée, mais la contrainte (10) performe un peu mieux que (11), surtout lorsque le tournoi à planifier est complexe. À la lumière de ces résultats, nous concluons que notre modèle le plus performant est ADR qui obtient de très bonnes performances en trouvant l'horaire optimal pour un tournoi de 85 équipes en moins de 3 minutes.

Ce résultat est très intéressant d'un point de vue pratique puisque la portée de notre modèle est assez grande pour traiter des situations réelles dans des temps plus que raisonnables. Il serait alors possible d'intégrer ce modèle dans une plateforme de gestion de tournois sportifs afin de générer automatiquement des horaires valides pour les organisateurs pour des tournois de moins de 100 équipes. En procédant ainsi, il serait possible en l'espace de quelques minutes de proposer des horaires pour le tournoi en entier, tâche qui peut prendre plusieurs jours, lorsque faits manuellement.

Il n'est pas étonnant que la contrainte *AllDifferent_Except_0* performe mieux que l'autre approche proposée même si le nombre de contraintes de ce modèle est plus grand. En effet, l'utilisation de la contrainte *AllDifferent_Except_0* permet d'appliquer la cohérence de domaines sur les variables de la matrice **S** qui est l'élément clé de notre modèle. On peut ainsi très rapidement élaguer notre arbre de recherche et obtenir un horaire optimal.

Aussi, on remarque un comportement étrange lorsqu'on augmente drastiquement le nombre de pauses requises entre deux parties. En effet, la figure 4 montre que le temps requis pour trouver l'horaire optimal est constant jusqu'à ce que l'on atteigne 8 pauses. pour un très grand nombre de pauses, le temps de calcul explose. Ceci est explicable par le fait que les horaires possibles sont largement limités lorsque le nombre de pauses est grandement limité alors le branchement sur les variables de coût mène souvent à des *nogoods*. Heureusement, ces situations ne sont pas pertinentes dans des cas pratiques puisque le nombre de périodes de pauses requises

Figure 4 – Performance du modèle ADR selon le nombre de pauses requises entre deux parties



sera généralement petit. Notre modèle offre alors des performances constantes pour tous les nombres de pauses qui sont susceptibles d'être utilisés en pratique.

De plus, nous avons analysé l'effet du nombre d'entraîneurs dans un tournoi à planifier sur le temps de calcul nécessaire pour trouver un horaire optimal. Les résultats de cette analyse se trouvent dans la figure 5. Pour tous ces scénarios, le nombre d'équipes a fixé à 50. Ainsi, lorsqu'on diminue le nombre d'entraîneurs, on augmente le nombre de cas où un entraîneur est responsable de deux équipes ou plus ce qui complexifie le problème. La forme en dents de scie décroissante de la courbe de la figure 5 confirme cette intuition et nous indique que le temps de calcul augmente à mesure que l'oncomplexifie le problème. La forme en dents de scie du graphique est explicable par le fait que l'attribution des entraîneurs aux différentes équipes se fait de façon aléatoire dans notre générateur de scénarios et que certaines situations avec beaucoup d'entraîneurs peuvent tout de même être plus complexes que certaines situations avec moins d'entraîneurs. C'est notamment ce qu'on remarque pour les scénarios avec 43 et 44 entraîneurs. Toutefois, on remarque que même dans les cas extrêmes, l'augmentation du temps de calcul est beaucoup moins drastique que celle obtenue avec des valeurs extrêmes pour le nombre de pauses.

Finalement, comme dans tout problème d'optimisation combinatoire, le temps de calcul augmente de façon exponentielle. On confirme ceci pour notre situation avec les résultats de la table 4 où la taille du tournoi amène rapidement les temps de calcul à plusieurs heures. Pour s'attaquer à des problèmes de très grande taille comme certains tournois impliquant jusqu'à 350 équipes, il faudrait prévoir de très grandes périodes de calcul ou améliorer le modèle encore plus pour repousser les barrières qui freinent présentement notre modèle.

7 Conclusion

Lors de ce projet, nous nous sommes attaqués à la planification d'horaires sportifs pour les ligues amateurs. Pour ce faire, nous avons construit 4 modèles de programmation linéaire prenant en compte plusieurs contraintes rencontrées lors de l'organisation de tournois sportifs dans des contextes réels. Ces contraintes sont la présence d'entraîneurs partagés entre certaines équipes, la présence de plusieurs lieux, les contraintes

Figure 5 – Performance du modèle ADR selon le nombre d'entraîneurs du touroi à planifier

d'indisponibilités des lieux et de temps pour certaines équipes et un nombre minimal de pauses entre deux parties.

À la lumière de nos expérimentations, notre modèle le plus performant est ADR qui utilise les contraintes globales *AllDifferent_Except_0* et *Regular* qui appliquent la cohérence de domaine sur nos variables d'horaire et permettent de trouver un horaire optimal pour des tournois allant jusqu'à 95 équipes en environ 4 minutes, tâche qui prend normalement des jours, lorsque faits manuellement.

Également, nous avons vu que le nombre d'entraîneurs partagés entre les différentes équipes et le nombre minimum de pauses requises entre chaque partie n'a pas d'impact significatif sur le temps de calcul lorsqu'ils se trouvent dans des intervalles de valeurs normales. Cependant, si le nombre d'entraîneurs ou le nombre de pauses requises deviennent extrêmes, on observe une explosion du temps de calcul. Si notre modèle venait à être utilisé dans des cas réels, il serait important de fixer des limites sur ces valeurs pour s'assurer que les temps de calcul se trouvent dans un intervalle de temps acceptable.

Aussi, l'architecture assez générale de notre modèle permet d'encoder plusieurs restrictions à l'aide des matrices de coût par équipe et l'utilisation d'une table de parties à planifier. En effet, il est possible d'utiliser notre modèle pour planifier un tournoi ayant une structure classique de type *round robin*, mais elle permet aussi de gérer des cas spécifiques où les organisateurs souhaiteraient placer un nombre fixe de parties à jouer pour chaque équipe. Également, il est possible de paramétrer la matrice de coût par équipe de sorte à optimiser plusieurs métriques tels le nombre de contraintes d'indisponibilité qui sont brimées ou les revenus générés par le tournoi.

Pour conclure, notre modèle s'intègre bien dans une solution informatique commerciale par l'utilisation de l'API Python de Minizinc qui permet d'appeler notre modèle avec une table de paramètres propre à la situation des utilisateurs de la solution. Notre but de proposer une solution utile à une problématique industrielle complète est donc atteint. L'entièreté du code et des résultats pour ce projet se trouvent sur notre dépôt GitHub.

Références

[1] Dirk Briskorn. Combinatorial properties of strength groups in round robin tournaments. *Eur. J. Oper. Res.*, 192 :744–754, 2006.

[2] Dirk Briskorn and Andreas Drexl. A branching scheme for finding cost-minimal round robin tournaments. *Eur. J. Oper. Res.*, 197 :68–76, 2009. 2

[3] Dirk Briskorn and Sigrid Knust. Constructing fair sports league schedules with regard to strength groups. *Discret. Appl. Math.*, 158 :123–135, 2010.

[4] Mats Carlsson, Mikael Johansson, and Jeffrey Larson. Scheduling double round-robin tournaments with divisional play using constraint programming. *Eur. J. Oper. Res.*, 259 :1180–1190, 2017.

[5] Kevin K. H. Cheung. Solving mirrored traveling tournament problem benchmark instances with eight teams. *Discret. Optim.*, 5 :138–143, 2008. 2

[6] Dries R. Goossens and Frits C. R. Spieksma. Indoor football scheduling. 2014. 2

[7] Martin Henz. Constraint-based round robin tournament planning. In *ICLP*, 1999. 2

[8] Sigrid Knust. Scheduling non-professional table-tennis leagues. *Eur. J. Oper. Res.*, 200 :358–367, 2010. 2

[9] Sigrid Knust and Daniel Lücking. Minimizing costs in round robin tournaments with place constraints. *Comput. Oper. Res.*, 36 :2937–2943, 2009.

[10] Jean-Charles Régin. Minimization of the number of breaks in sports scheduling problems using constraint programming. In *Constraint Programming and Large Scale Discrete Optimization*, 1998.

[11] Robert A. Russell and Timothy L. Urban. A constraint programming approach to the multiple-venue, sport-scheduling problem. *Comput. Oper. Res.*, 33 :1895–1906, 2006. 2

[12] Andrea Schaerf. Scheduling sport tournaments using constraint logic programming. *Constraints*, 4 :43–65, 1996. 2

[13] Túlio A. M. Toffolo, Jan Christiaens, Frits C. R. Spieksma, and Greet Vanden Berghe. The sport teams grouping problem. *Annals of Operations Research*, 275 :223–243, 2019.

[14] Timothy L. Urban and Robert A. Russell. Scheduling sports competitions on multiple venues. *Eur. J. Oper. Res.*, 148 :302–311, 2003.

[15] Mesut Yavuz, Umut H. Inan, and Alpaslan Figlali. Fair referee assignments for professional football leagues. *Comput. Oper. Res.*, 35 :2937–2951, 2008.