Planification des courses de plat au galop

Antoine Houdayer

le cnam

École doctorale Informatique, Télécommunications et Électronique (Paris)

Centre d'Étude et de Recherche en Informatique et en Communications

THÈSE DE DOCTORAT

présentée par : **Antoine HOUDAYER**
soutenue le : **10 juin 2021**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline : **Informatique**

Spécialité : **Informatique**

Planification des courses de plat au galop

THÈSE dirigée par

Mme. Costa Marie-Christine	*Professeur émérite, CNAM*

et co-encadrée par

M. Bonnet Laurent	*Responsable DSI, France Galop*
Mme Plateau Agnès	*Maître de Conférences, CNAM*
M. Soutil Éric	*Maître de Conférences, CNAM*

RAPPORTEURS

Mme Quadri Dominique	*Professeur des universités, Faculté des sciences d'Orsay*
M. Rossi André	*Professeur des universités, Université Paris Dauphine - PSL*

PRÉSIDENT DU JURY

M. Hudry Olivier	*Professeur, Télécom Paris*

EXAMINATEURS

M. Thome Nicolas	*Professeur des universités, CNAM*

Résumé

La problématique centrale de la thèse est la conception et la réalisation d'un logiciel permettant l'automatisation de certains aspects de la planification manuelle des courses hippiques de plat premium. L'outil de planification doit être capable de produire des programmes de courses respectant des contraintes, mais aussi d'évaluer leur qualité. Tout d'abord, un travail important a été réalisé autour de la formalisation du problème auprès de différents experts de France Galop chargés de la planification manuelle. Dans ce cadre, une formulation compacte intégrant les contraintes les plus significatives a été proposée sous la forme d'un programme linéaire en variables 0-1. Le problème d'optimisation est prouvé NP-difficile via une réduction depuis le problème des k-chemins à sommets disjoints dans un graphe orienté acyclique. Les coefficients de la fonction objectif de ce modèle découlent d'une estimation statistique du nombre de partants. Il est à souligner que toutes les contraintes implicites liées à la planification et à l'évaluation n'ont pu être prises en compte dans ce modèle. Ce modèle a néanmoins été testé sur des instances réelles et a montré des faiblesses notamment d'un point de vue temps de calcul mais également relativement aux structures des solutions de planification proposées. Une alternative approchée à cette approche exacte du problème a été développée via la mise en œuvre d'une méthode de planification basée sur des méta-heuristiques, notamment un recuit simulé, guidée par une fonction d'évaluation « boite noire ». L'utilisation de cette fonction d'évaluation non linéaire a permis de mieux contrôler la nature des solutions produites. Ces solutions ont ensuite été comparées aux solutions produites par une heuristique imitant le processus manuel de planification.

La mise au point de cette fonction d'évaluation efficace d'un programme de courses en contexte incertain a représenté un enjeu important de la thèse. Une partie de cette évaluation a reposé sur le respect de certaines contraintes de planification qualifiées de « souples », même si, en général la qualité d'un programme de courses se mesure uniquement en termes de nombre de partants aux courses, métrique indisponible a priori.Des travaux ont donc été effectués pour mettre à profit les données passées

disponibles par France Galop à l'aide de méthodes statistiques pour tenter de prédire le nombre de partants aux courses d'un nouveau programme. Plusieurs méthodes ont été envisagées et sont présentées avec une discussion sur leurs qualités mais aussi sur les difficultés auxquelles elles se heurtent. Par ailleurs, l'évaluation de la qualité d'un programme avec une méthode d'apprentissage est chronophage, et nous présentons une variation de l'algorithme du recuit simulé permettant de minimiser simultanément une évaluation classique et une évaluation par apprentissage tout en minimisant le nombre d'appels à la partie apprentissage, chronophage, de l'évaluation.

Abstract

The main subject of this thesis is the conception and the implementation of a software that can automate aspects of the planification process for flat horses races. The software must be able to produce race programs obeying planification constraints, and to estimate their quality. First of all, we have worked with the formalization of the problem, with the help of the planification experts from France Galop. To this that end, we have written a 0-1 linear compact formulation of the problem that includes the main planification constraints. The corresponding decision problem is proven to be NP-complete using a reduction from the disjoint vertices k-path problem in an acyclic digraph. The coefficients used in the objective function are deduced from a statistical estimation of the number of enlisted horses for each horse race type. It is important to note however that some implicit planification constraints and some aspects of the evaluation could not be taken into account in a linear formulation. Nonetheless, we tested the model on real instances, which highlighted some weaknesses of the formulation with respect to computation times and most importantly with respect to the structure of the solutions produced. As a consequence, we developped meta-heuristic based approximate methods, amongst which an algorihm based on the simulated annealing, guided by a black-box evaluation function. Thanks to the use of this non linear obejective function, we were able to better control the nature of the solutions produced. These solutions are then compared to solutions produced by an heuristic that mimics the manual planification process.

Designing this efficient evaluation function for an uncertain context has been another important subject of the thesis. Part of this evaluation function was obtained by evaluating the respect of soft planification constraints, event though we generally consider that a horse race program is good when a lot of horses participate to each race, a metric that is only available afterward. Therefore, we worked to make use of past race data through machine learning to try and predict the number of participating horses. We've considered several methods that we implemented and we discuss their qualities as well

as the difficulties that they face. Furthermore, evaluating the quality of a horse race program with a machine learning method can be time consuming, and we introduce and variation of the simulated annealing algorithm allowing the simultaneous minimizing of a classical evaluation function and of a machine learning base evaluation, while also minimizing the number of calls to the computationally expensive machine learning evaluation.

Table des matières

Liste des tableaux

Table des figures

Chapitre 1

Présentation du problème

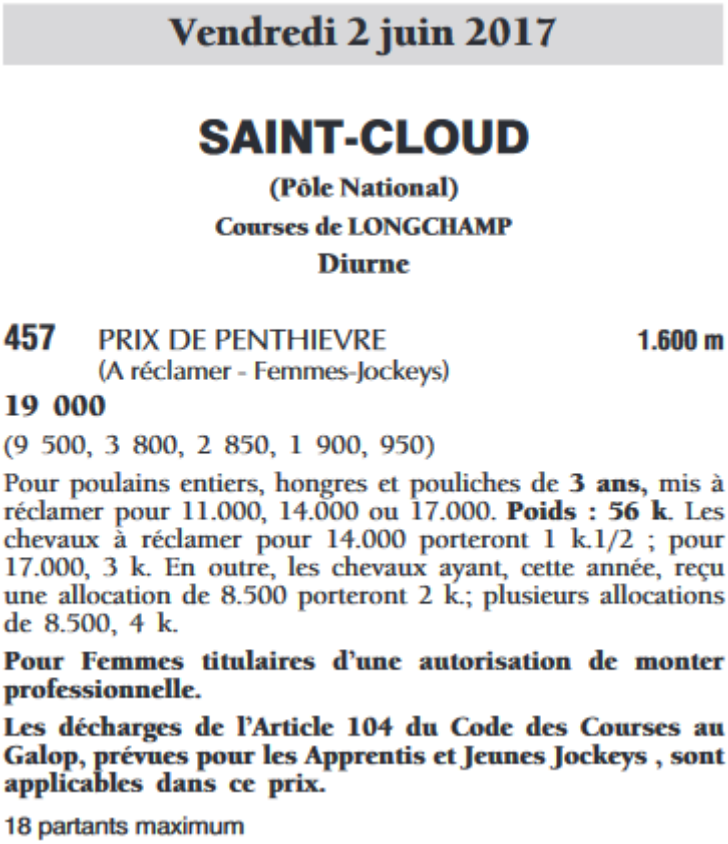
Contenu

1.1 Contexte

France Galop est une association à but non lucratif qui contrôle et organise la filière des courses de galop en France. Chaque année, France Galop élabore un calendrier de plus de 7300 courses de plat et à obstacles sur 151 hippodromes de galop répartis à travers tout le territoire [1]. La création de ce calendrier est une tâche complexe soumise à de nombreuses contraintes. Le calendrier doit entre autres respecter la disponibilité des différents hippodromes, et être compatible avec le calendrier des courses de trot, dont l'association « Le Trot » est responsable. De plus, au sein même de France Galop, il existe différentes catégories de courses, planifiées sur des calendriers différents mais en tenant compte de potentielles corrélations. En effet, les courses de galop sont divisées en deux grandes disciplines : le plat et l'obstacle. Les courses d'obstacle sont des courses dans lesquelles les chevaux doivent courir en franchissant des obstacles présents sur la piste. Il peut s'agir par exemple de simples haies ou de rivières. Les courses de plat sont au contraire des courses dont la piste est exempte d'obstacles. Pour chacune de ces deux disciplines, il existe ensuite des courses dites PMU (Pari Mutuel Urbain) et d'autres dites PMH (Pari Mutuel Hippodrome). Une course est dite PMU lorsqu'elle est ouverte aux paris hors-ligne du PMU. Une course est dite PMH lorsque les paris pour cette course sont uniquement ouverts sur l'hippodrome. Pour chacune de ces familles de courses, un programme des courses est rédigé à la main par l'équipe en charge. Ce programme est actuellement publié trimestre par trimestre : de mars à mai, de juin à août, de septembre à novembre, et de décembre à février.

Un programme de Plat ou d'Obstacles est divisé en réunions, c'est-à-dire un ensemble de courses se déroulant durant la même journée sur un même hippodrome. Le programme des réunions, c'est-à-dire le programme de l'occupation des différents hippodromes, ainsi que le nombre de courses que chaque réunion va accueillir, est décidé au préalable. Le rôle de l'équipe rédigeant le programme des courses est alors de décider quelles courses planifier dans chacune des réunions. Pour chaque course, elle doit notamment décider à quels chevaux sera ouverte la course, suivant principalement leur valeur, leur âge et leur race, quelle distance vont parcourir les chevaux, les allocations de la course, ou encore le type de course. Dans l'exemple de la figure 1.1, la course intitulée, « Prix de Penthievre », est une course de type *à Réclamer*, seules les femmes « titulaires d'une autorisation de monter professionnelle » sont autorisées à monter. Ce prix est réservé aux chevaux de trois ans. La distance à parcourir est de

1. `http://www.france-galop.com/fr/mission`



Vendredi 2 juin 2017

SAINT-CLOUD

(Pôle National)

Courses de LONGCHAMP

Diurne

457 PRIX DE PENTHIEVRE **1.600 m**
(A réclamer - Femmes-Jockeys)

19 000

(9 500, 3 800, 2 850, 1 900, 950)

Pour poulains entiers, hongres et pouliches de **3 ans,** mis à réclamer pour 11.000, 14.000 ou 17.000. **Poids : 56 k**. Les chevaux à réclamer pour 14.000 porteront 1 k.1/2 ; pour 17.000, 3 k. En outre, les chevaux ayant, cette année, reçu une allocation de 8.500 porteront 2 k.; plusieurs allocations de 8.500, 4 k.

Pour Femmes titulaires d'une autorisation de monter professionnelle.

Les décharges de l'Article 104 du Code des Courses au Galop, prévues pour les Apprentis et Jeunes Jockeys , sont applicables dans ce prix.

18 partants maximum

FIGURE 1.1 – Extrait du programme des courses de plat PMU

1600 m. À l'issue de la course, une allocation de 19000 € sera répartie entre les chevaux placés de la première à la cinquième place comme suit : 9500 € (1er), 3800 € (2e), 2850 € (3e), 1900 € (4e), 950 € (5e). Dans toute la suite, la mention « *Le programme* » fera référence au programme des courses et des réunions.

1.2 Problématiques et enjeux de l'étude

Les fonds de France Galop proviennent en grande partie d'un prélèvement sur les 10 milliards d'euros de paris annuels. Une étude précédente a montré une corrélation significative entre le nombre de partants d'une course et les revenus qu'elle génère. En effet, le nombre de partants d'une course conditionne les paris qu'il est possible d'organiser sur cette course. Si le nombre de partants est insuffisant, on ne peut pas organiser de paris complexes. C'est aussi pourquoi le bénéfice n'augmente pas régulièrement avec le nombre de partants, mais plutôt suivant des seuils. Dans toute la suite, on

entendra par nombre de partants, le nombre de chevaux déclarés partants pour la course, c'est-à-dire le nombre de chevaux pour lesquels une intention définitive de participer a été exprimée. Il se peut que le nombre de partants effectif soit inférieur en cas de force majeure, telles que des intempéries empêchant de se rendre sur le lieu de l'hippodrome, une blessure etc... Naturellement, le nombre de partants est influencé par le programme. Une bonne planification des courses est donc primordiale. Pour cela, de nombreuses règles extérieures au code des courses au galop sont appliquées lors de la création du programme pour en assurer sa qualité. D'une certaine manière, il s'agit d'offrir un maximum d'opportunités aux chevaux, ou autrement dit d'éviter qu'un cheval se retrouve sans course adaptée à sa catégorie alors qu'il est disponible. Il peut par exemple s'agir de règles de la forme « Avoir au moins une course de type X toutes les trois semaines ». La plupart de ces règles ne font pas partie explicitement du code des courses au galop [2], mais sont connues et appliquées par les responsables de la programmation. Le processus de rédaction d'un programme est long et fastidieux, il est impossible en l'état actuel des choses, d'en écrire plusieurs pour choisir le meilleur d'entre eux, si tant est que l'on puisse reconnaître le meilleur programme d'un éventail.

Ainsi, les perspectives d'évolution du processus de planification sont multiples. Il s'agit d'une part d'accélérer la procédure, mais aussi d'améliorer si possible la qualité du programme résultant. Ces deux évolutions sont toutefois liées : pouvoir générer rapidement un programme, même grossier, peut permettre l'application de réglages successifs, de changer des contraintes, des paramètres, pour faire évoluer le programme dans une direction satisfaisante, et pour découvrir éventuellement d'autres façons de faire un programme.

La décision prise par France Galop pour adresser cette problématique est la mise au point d'un système informatisé capable d'automatiser une partie de la planification des courses, et les experts ont rédigé un cahier des charges pour cet outil. Nous nous limitons pour le moment aux courses de plat premiums (terme strictement équivalent à course PMU). Essentiellement, on souhaite augmenter la moyenne du nombre de partants par courses, augmenter la part des courses quotidiennes ayant plus de 13 partants, et réduire la part des courses quotidiennes ayant moins de 8 partants.

Ainsi, la problématique centrale de la thèse est la conception et la réalisation à partir de zéro d'un logiciel permettant l'automatisation de certains aspects de la planification manuelle des courses de plat premium. Dans toute la suite, la mention « L'outil » fera référence à la solution logicielle proposée.

2. `http://www.france-galop.com/fr/node/77`

L'objectif est double : l'outil doit être capable de produire des programmes de courses respectant des contraintes, mais aussi d'évaluer leur qualité.

Tout d'abord, un travail important a été réalisé autour de la formalisation du problème. Après avoir clarifié la nomenclature liée aux courses de chevaux, une première difficulté rencontrée a concerné la modélisation des contraintes. De nombreuses règles sont appliquées implicitement par les experts, et il peut s'agir de contraintes dures ou de contraintes souples. Nous avons dialogué avec les experts à de nombreuses reprises pour clarifier les contraintes à modéliser, et la façon de les appliquer. Il est clair également que les contraintes sont amenées à évoluer, pour suivre par exemple l'apparition de nouveaux types de courses ou la disparition d'anciens types de courses. Au fur et à mesure de la production de l'outil, les experts sont consultés pour valider l'implémentation des contraintes, et recueillir leurs suggestions d'évolution. Le problème, c'est-à-dire la nomenclature des courses ainsi que les contraintes exprimées par les experts, est énoncé dans cette première partie.

Dans le deuxième chapitre, nous situons le problème étudié par rapport à la littérature existante. Nous mettons tout d'abord le problème en relation, thématiquement, avec les autres problèmes de planification sportive, qui sont en fait assez éloignés de nos problématiques. Nous répartissons ensuite les références bibliographiques suivant les deux axes principaux de la thèse : l'évaluation de la qualité des solutions d'une part, c'est-à-dire la gestion de l'incertitude (liée à la participation des chevaux aux courses), et d'autre part la planification elle-même. Nous comparons le problème de la planification des courses de galop au(x) problème(s) d'ordonnancement cyclique, qui à notre connaissance est celui qui s'en rapproche le plus dans la mesure où une contrainte centrale du problème est la planification régulière des différents types de courses.

Un travail important dans la thèse a été la mise au point d'une méthode d'évaluation des programmes de courses. Une partie de cette évaluation peut reposer sur le respect des contraintes souples, mais on considère en général qu'un programme de course est de qualité lorsque de nombreux chevaux ont couru. Il s'agit d'une métrique uniquement disponible a posteriori, et qui est donc inutilisable lors de la création d'un nouveau programme de courses. En troisième chapitre, on présentera les travaux effectués pour mettre à profit les données passées à l'aide de méthodes statistiques pour tenter de prédire le nombre de partants aux courses d'un nouveau programme. Plusieurs méthodes ont été envisagées et seront présentées avec une discussion sur leurs qualités mais aussi sur les difficultés auxquelles elles se heurtent.

En quatrième chapitre, nous aborderons les méthodes exactes étudiées pour effectuer la planification, à savoir la programmation par contraintes, et la programmation linéaire. Nous présenterons en particulier un modèle linéaire du problème en variables 0-1, et nous prouverons que le modèle proposé est NP-difficile. Outre la difficulté théorique du problème, les méthodes exactes posent également le problème de la modélisation des contraintes souples, et de l'utilisation des résultats des méthodes statistiques pour guider la planification. Après avoir effectué des tests sur des instances réelles et présenté les limites de l'approche, on présentera en dernière partie les travaux portants sur les méthodes approchées. Une méthode de planification basée sur l'algorithme du recuit simulé est implémentée, et les solutions produites sont comparées aux solutions produites par une heuristique imitant le processus manuel de planification.

1.3 Typologie des courses

Un programme de courses est la liste de toutes les courses planifiées sur un intervalle de temps donné (généralement trois mois). Les courses se déroulant le même jour et sur le même hippodrome sont regroupées dans ce qu'on appelle une *réunion*. Généralement, chaque réunion peut comprendre jusqu'à neuf courses. Les courses sont classées en différents *types* de course, suivant l'âge des chevaux, la distance et la catégorie de la course. Ainsi, dans cette section, nous détaillons les différentes caractéristiques qui définissent le type d'une course. Dans la section suivante, on présentera l'allure typique de la carrière d'un cheval, pour mieux comprendre les interactions entre les différents types de courses.

1.3.1 Conditions d'âge

Pour pouvoir participer à une course donnée, un cheval doit en premier lieu répondre à un critère d'âge [3]. Il existe trois classifications principales : les courses pour chevaux de deux ans, celles pour chevaux de trois ans, et les courses pour chevaux de quatre ans et plus. Des courses spécifiques pour les chevaux de quatre ans ou plus de cinq ans sont également organisées. On peut observer très occasionnellement d'autres conditions, comme des courses dédiées aux chevaux de trois ans et plus.

3. L'âge d'un cheval change au premier janvier. Un cheval né le 12 août 2015 sera considéré comme un cheval de trois ans dès le 1er janvier 2018.

1.3.2 Distances

Il existe plusieurs plages de distances sur lesquelles se déroulent les courses (Fig. 1.2). Les courses les plus courues sont les courses *Flyer*, entre 800 et 1400 mètres. Il y a ensuite les courses *Miler* de 1450 à 1800 mètres, les courses *Intermediate* de 1850 mètres à 2150 mètres, les courses *Classical* de 2200 à 2400 mètres et, pour finir, les courses *Stayer* pour les distances supérieures (de 2450 mètres à 3200 mètres).

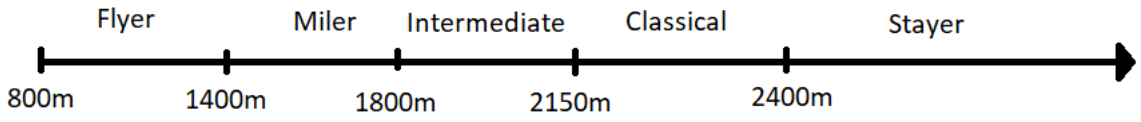


FIGURE 1.2 – Plages de distances

1.3.3 Catégories

Les catégories de courses sont réparties en trois groupes, qui se distinguent par les conditions d'inscription, et les règles de la course.

Courses à Handicap

Dans une course à Handicap, les chevaux partants portent des poids pour courir. Théoriquement, les poids sont choisis de sorte que tous les chevaux aient une probabilité égale de l'emporter, qu'ils soient ainsi sur un pied d'égalité. Le poids que devra porter chaque cheval est décidé par le handicapeur, dont c'est le métier, sur la base des performances récentes du cheval. Le poids à porter est appelé *valeur* du cheval, et varie typiquement entre 15 kilogrammes, pour les chevaux les moins performants, à 50 kilogrammes pour les chevaux les plus performants (les *cracks*). Évidemment, le poids du jockey est pris en compte lors du calcul du poids supplémentaire (chaque jockey se soumet à une pesée avant la course). Par nature, ces courses sont destinées à une population de chevaux assez large puisque un entraîneur peut espérer gagner même avec un cheval moins fort que ses concurrents. Ce sont donc des courses qui ont en général beaucoup de partants. Toutefois, les courses à Handicap ne sont pas toujours ouvertes à tous ; il peut y avoir des conditions d'inscription supplémentaires sur la valeur des chevaux partants. Régulièrement, une course à Handicap est ainsi support d'événement. La récompense accordée au premier est alors bien plus importante que dans un Handicap de même niveau non support

d'événement. On attend alors beaucoup de partants et des paris complexes (comme le quinté) sont organisés.

Courses à Réclamer

La deuxième catégorie de course sont les courses à Réclamer. À la fin de la course à réclamer, les partants sont vendus aux enchères. Comme dans la course à Handicap, chaque cheval porte un poids. En revanche, le poids ne dépend pas directement de la valeur du cheval, mais détermine le prix initial de la mise aux enchères du cheval (plus le poids est important, plus la mise à prix est élevée). Un poids minimum est imposé dans les conditions de participation, mais chaque propriétaire peut décider de faire porter un poids plus important à son cheval, pour le vendre plus cher.

Courses à Conditions

Une course à Condition, comme son nom l'indique, est une course pour laquelle la participation est soumise à une (ou des) condition(s). Certaines peuvent, par exemple, dépendre de la valeur du cheval et de ses gains cumulés.

Parmi les courses à Condition, on trouve plusieurs catégories de courses. Tout d'abord, il y a les courses d'*Inédits* et les courses *Maiden*, réservées aux chevaux débutants, pour les accompagner et faciliter leur insertion dans le circuit compétitif. Les courses d'Inédits sont réservées aux chevaux n'ayant jamais couru, les courses Maiden sont réservées aux chevaux n'ayant jamais gagné.

Ensuite, on trouve les courses à Condition qui constituent le circuit compétitif du programme. La participation est soumise à des conditions portant sur la valeur du cheval et sur son palmarès. Les plus accessibles sont les courses à Condition C4, suivies des C3, des C2, et des C1. On trouve ensuite les courses *Listed*, et les courses de *Groupe*, réservées à l'élite. Comme les courses à Condition, et en particulier celle de niveau très élevé, sont plus sélectives, on y attend moins de partants que dans les courses à réclamer ou à Handicap.

N.B. : Il existe encore bien d'autres conditions de participation, liées par exemple à la race du cheval (les courses AQPS pour les chevaux autres que les pur sang) qui n'ont pas été présentées en détail. À titre informatif, nous pouvons citer également des conditions associées aux jockeys comme les courses réservées aux femmes jockeys, et les courses réservées aux jockeys amateurs. Elles sont plutôt d'ordre exceptionnel, et ne changent pas la compréhension du problème. Nous ne les prendrons pas en

compte dans cette étude.

1.4 Contraintes du problème

Dans cette section, nous donnons les contraintes du problème telles qu'énoncées par France Galop. Nous avons d'une part des contraintes dures, matérielles ou émanant du code des courses, et d'autre part des contraintes souples. Ces contraintes souples correspondent à des heuristiques de planification utilisées par les experts pour produire des programmes de qualité. Avant de les présenter, nous donnons quelques détails concernant la carrière typique d'un cheval pour apporter une explication aux contraintes.

Une partie importante de la thèse porte sur la formalisation de ces contraintes, notamment dans la sections 4.5 où l'on proposera un modèle mathématique linéaire, et en section 5.2 ou l'on détaillera l'expression de la fonction objectif utilisée dans les méthodes de résolution approchées.

1.4.1 Carrière d'un cheval

Un des objectifs de la planification est d'attirer le plus de partants possible aux différentes courses. Pour ce faire, il est important de comprendre comment les entraîneurs et propriétaires utilisent le programme pour composer la carrière de leurs chevaux. Dans cette section, sont discutées les carrières type des chevaux pour comprendre l'interaction entre les différentes courses, et donner du sens aux contraintes du problème.

Un cheval peut commencer à courir à partir de ses deux ans, à partir du 25 mars. Puisque les courses d'Inédits sont réservées aux chevaux n'ayant jamais couru, un cheval débute normalement sa carrière par une course d'Inédits. Un cheval qui gagne une course d'Inédits va généralement se diriger vers une course C2, pour entrer dans le circuit compétitif. Si, en revanche, il ne gagne pas, il va participer à une course Maiden, réservée aux chevaux n'ayant jamais gagné. La majorité des courses d'Inédits et des courses Maiden sont donc logiquement réservées aux chevaux de 2 ans, mais on en trouve aussi pour les chevaux de 3ans. Il y a des chevaux qui commencent leur carrière tardivement, mais aussi qui se reconvertissent en coureur de plat, après avoir couru des courses d'obstacles. On observe ensuite deux profils types. Certains chevaux vont avoir un niveau suffisant pour participer aux courses à Condition. S'ils y sont performants, ils pourront prétendre à des courses de plus en plus

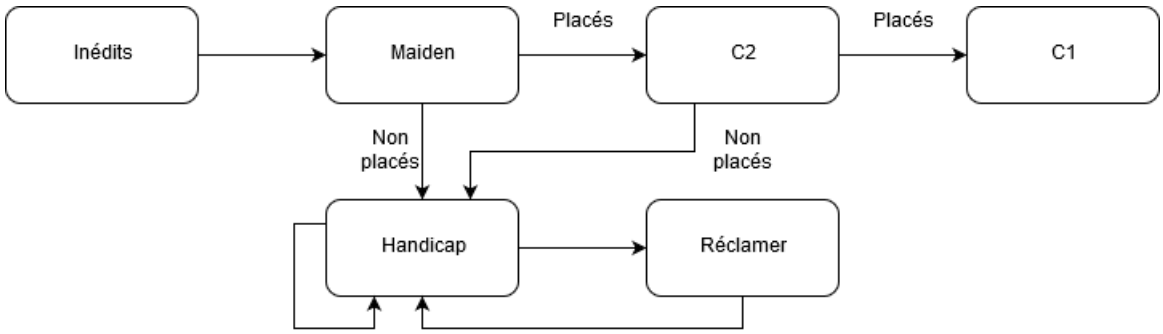


FIGURE 1.3 – Diagramme des évolutions typiques pour un cheval de 2 ans.

sélectives. D'autres chevaux vont plutôt participer aux courses à Handicap et à celles à Réclamer.

La figure 1.3 illustre la progression typique d'un cheval de deux ans. A l'issue d'une course, les chevaux placés, c'est-à-dire finissant parmi les 5 premiers, vont souvent chercher à participer à une course de meilleur niveau. Par exemple un cheval placé à une course C2 pourra vouloir courir une course C1 ensuite. Toutefois, la figure 1.3 est purement indicative et d'autres transitions ont bien sûr lieu.

En ce qui concerne les distances courues, les chevaux jeunes vont chercher leur distance de confort, d'autant plus que les distances importantes ne sont accessibles aux chevaux de 2 ans que tard dans l'année. Ensuite, les entraîneurs inscriront leurs chevaux, en priorité, à des courses sur leur distance de prédilection.

1.4.2 Contraintes dures de la planification

1.4.2.1 Code des courses au galop

Le code des courses au galop [1] réglemente tout ce qui a trait aux courses de galop, de la naissance des chevaux, au versement des gains. On y trouve par exemple le détail du processus d'inscription d'un cheval à une course, des vérifications à effectuer avant une course, ou encore de l'achat des chevaux mis à réclamer. En revanche, il ne contient quasiment aucune règle concernant directement la réalisation d'un programme de courses. Nous prendrons seulement note de la règle suivante, portant sur les distances que peut courir un cheval en fonction de son âge :

— **ART. 59** DISTANCES ET DATES D'ORGANISATION DE CERTAINES CATÉGORIES DE COURSE :

— La distance d'une course ne peut être inférieure à 1000 mètres dans les Handicaps ou à 800 mètres dans les autres courses.

— Les courses ouvertes aux chevaux de deux ans sont soumises aux restrictions suivantes :

a) du jour de l'ouverture des courses plates jusqu'au 30 avril inclus, lesdites courses doivent être réservées aux chevaux de deux ans et d'une distance au plus égale à 1000 mètres. Toutefois, des dérogations à ces dispositions peuvent être accordées par les Commissaires de France Galop.

b) du 1er mai au 31 août, lesdites courses ne peuvent être que des prix réservés aux chevaux de deux ans, d'une distance au plus égale à : 1200 m en mai, 1400 m en juin, 1600 m en juillet. Toutefois, des dérogations aux dispositions des alinéas a) et b) peuvent être accordées par les Commissaires de France Galop.

c) à partir du 1er octobre, les dites courses peuvent être des Handicaps à condition d'être réservées aux chevaux de deux ans.

d) à aucun moment les courses ouvertes aux chevaux de deux ans ne peuvent être disputées sur une distance supérieure à 2000 mètres.

1.4.3 Autres contraintes dures

Des contraintes dures supplémentaires s'appliquent.

— Les courses pour les chevaux de 2 ans ne commencent pas avant le 25 mars.

— Pour les chevaux de 3 ans les courses sur les distances de 2400 m et plus ne peuvent pas commencer avant le 15 avril. Celles sur une distance de 3000 m et plus, ne peuvent débuter avant le 10 juin.

— Tout d'abord, le nombre de courses que peut accueillir une réunion est limité par la durée d'une journée de courses. La décision du nombre de courses dans une réunion ne nous revient pas, mais le respect de cette décision est une contrainte dure.

— Pour chaque hippodrome, seules certaines distances peuvent être organisées, en fonction de la piste, en particulier les courses Flyer devant être courues sur une ligne droite. En effet, un hippodrome ne peut pas accueillir de courses Flyer s'il ne dispose pas d'une ligne droite d'entre 1000 m et 1400 m. À titre d'exemple, du fait de la règle sur les distances émanant du code des courses, il est possible qu'un hippodrome puisse organiser une course Maiden de distance Flyer

pour des chevaux de 2 ans en automne, en juin, mais pas en mars.

1.4.4 Contraintes souples

À ces contraintes dures s'ajoutent des contraintes souples. Ces contraintes nous on été données par les experts, dans un travail de formalisation du processus manuel de planification. Il s'agit donc d'habitudes de planification, basées sur leur expérience.

1.4.4.1 Contrainte d'écart

Entre chaque participation, un cheval a besoin d'un temps de repos. La planification prend en compte ce temps de repos pour faire en sorte qu'un cheval qui ne soit pas au repos ait toujours une course à laquelle il puisse s'inscrire et ainsi progresser dans sa carrière. En pratique, cela se traduit par la règle suivante : pour toute course planifiée, on planifiera toujours une course de même distance (Flyer, Miler, Intermediate, Classical, ou Stayer), de même catgéorie (Handicap, Réclamer, ...) et pour le même âge (2 ans, 3 ans, ou 4 ans et plus) dans un intervalle de temps déterminé après la première course (par exemple, entre 18 et 25 jours plus tard), qui dépend de la distance. Ainsi, tout cheval courant une course aura la possibilité de courir une course similaire après sa période de repos. On demande aussi qu'un tel écart soit respecté entre une course à Inédits et une course Maiden, pour que les chevaux courant une course à Inédits aient la possibilité de courir une course Maiden après leur période de repos.

1.4.4.2 Contrainte HR

On veut que chaque réunion soit composée d'un mélange de course compétitives et de courses plus accessibles. Ainsi, chaque réunion doit contenir de 3 à 5 courses à Handicap ou à Réclamer (en privilégiant les courses à Handicap), dans la mesure du possible, compte tenu de la capacité de la réunion.

1.4.4.3 Contrainte de Paires

Certains types de courses doivent être planifiés par paire, c'est-à-dire que le même type de course doit être planifié deux fois dans toute réunion où il est planifié. C'est le cas des courses à Handicap pour

les chevaux de 4 ans et plus, car elles accueillent beaucoup de partants. C'est également le cas pour les courses Maiden, pour lesquelles on planifiera une course pour les mâles et une pour les femelles.

Chapitre 2

État de l'art

Contenu

2.1 Introduction

En première partie, nous avons présenté le problème de planification, ses problématiques, ainsi que les différentes approches utilisées pour les traiter. Dans cette partie, consacrée à l'état de l'art, nous situons le problème étudié vis-à-vis de la littérature existante, selon deux axes : l'incertitude liée à l'évaluation d'une solution réalisable, et les problèmes de planification connexes à nos problématiques, notamment à la contrainte de périodicité liée aux courses. Le problème émane d'un domaine industriel inhabituel pour ce qui concerne l'utilisation d'outils d'optimisation, et cela s'accompagne de contraintes, de difficultés, et de problématiques tout à fait singulières. En particulier, il n'existe pas à notre connaissance de travaux effectués portant sur la planification des courses de chevaux, même dans d'autres disciplines telles que les courses de trot ou d'obstacle. Le but de cette revue de littérature est donc de montrer comment le problème se rapproche de travaux existants suivant certains aspects, mais aussi de mettre en valeur comment il s'en distingue suivant d'autres.

2.2 Planification d'évènements sportifs

Comme indiqué en introduction, il n'existe pas à notre connaissance d'étude portant sur la planification des courses de chevaux. Il est intéressant de noter par ailleurs que la planification des courses de chevaux peut être très différente suivant les pays. Aux États-Unis, la problématique géographique est très important du fait de la taille du pays. Au Royaume-Uni, les paris passent par des bookmakers, ce qui impacte bien sûr la stratégie de planification.

On peut tout de même, thématiquement, rapprocher le problème étudié des problèmes de planification d'évènements sportifs. Ce type de problème est relativement répandu en Recherche Opérationelle, comme en témoigne la revue de littérature portant sur ce thème par Kendall et al. [2].

La planification des rencontres de football et de baseball en particulier ont de nombreuses applications. La planification des rencontres de la IFL (Italian Football League) a par exemple été étudiée par Della Croce et al. [3]. Une des contraintes récurrentes dans ce type de problème est l'assurance d'une bonne alternance entre les rencontres à domicile et les rencontres à l'extérieur pour les équipes. Dans cet article, l'affectation des droits de diffusions des rencontres aux différentes chaînes de télévision fait également partie de la planification. Les auteurs proposent alors une méthode heuristique basée sur la résolution successive de plusieurs programmes linéaires en variables entières (PLNE). Un

premier PLNE détermine des motifs possibles pour l'alternance rencontre à domicile/à l'extérieur. Un deuxième PLNE constitue un calendrier en combinant ces motifs. Un dernier PLNE affecte les équipes aux motifs.

Un exemple dans un autre sport, l'étude de Taeho Kim [4] sur la ligue de baseball sud-coréenne (sport le plus regardé en Corée du Sud). Ici, on cherche à minimiser la distance parcourue pour les équipes. L'alternance entre rencontres à l'extérieur et à domicile est toujours soumise à des contraintes, et on trouve également les contraintes relatives au format du tournoi. Le problème est modélisé par un programme linéaire à variables mixte-entières, et une heuristique est ici aussi proposée.

Le problème de planification des courses de galop a finalement assez peu de points en commun avec les autres problèmes de planification sportive sur le plan théorique. Une différence majeure est la présence d'incertitude. Il ne semble pas que la robustesse et le traitement de l'incertitude soient des problématiques qui apparaissent habituellement dans les problèmes de planification sportive. Dans notre problème, on ne sait pas quels chevaux participeront aux courses organisées, puisque chaque entraîneur décide librement des courses auxquelles il inscrit ses chevaux. On a donc naturellement des problématiques très différentes des problèmes traditionnels de planification sportive, où le but est généralement de gérer l'enchaînement précis des rencontres. On veut proposer à chaque entraîneur une offre de courses qui soit satisfaisante quels que soient ses objectifs et le profil de ses chevaux. Cette vision de la planification est donc bien opposée à l'objectif typique d'imposer à des équipes un calendrier. Par ailleurs, on planifie plusieurs centaines de courses par trimestre, pour plusieurs milliers de coureurs, alors qu'une ligue professionnelle compte typiquement quelques dizaines d'équipes tout au plus. Enfin, même si nous planifions avec un horizon de planification fixé, il faut assurer que les différents programmes trimestriels s'enchaînent avec souplesse, par opposition à la planification d'un tournoi prenant place sur une plage de temps fixée, et avec un nombre de rencontres fixé.

2.3 Gestion de l'incertitude

La gestion de l'incertitude portant sur la participation est l'un des deux axes principaux de la thèse, avec la production de programmes respectant les contraintes dures et souples. Étant donné un programme de courses, le nombre de partants que va générer le programme n'est pas connu à l'avance. La source d'incertitude est double : d'une part, les nombres de partants aux courses dépendent de

facteurs extérieurs à la planification, et peuvent donc, pour un programme donné, être considérés pour chacune des courses comme des variables aléatoires (qui ne sont a priori pas indépendantes) ; d'autre part, les lois que suivent ces variables aléatoires sont inconnues. Par ailleurs, l'incertitude se situe uniquement dans la fonction objectif, et non dans les contraintes du problème à l'inverse, par exemple, des problèmes de la littérature pour lesquels l'incertitude émane souvent d'une demande incertaine.

La prise en compte d'une incertitude portant sur la fonction objectif apparaît notamment dans le problème de sélection de routes, un sous-problème du problème de gestion des flux du trafic aérien. Le but du problème est de sélectionner parmi un éventail de routages pour les avions, celui qui engendrera le moins de coûts dus à des retards au départ et à l'arrivée (liés à la surconsommation de carburant). Le problème est étudié entre autres par Tian et al. [5] pour l'espace aérien CSC (Central and Southern China). La méthode utilise trois composantes principales : un outil de simulation de transport aérien, un modèle d'estimation des surcoûts en fonction des retards, et un algorithme de sélection des routages. L'outil de simulation permet, étant donné un routage, de générer des scénarios relatifs à ces routages. L'évaluation des coûts de ces scénarios peut alors permettre d'estimer le coût moyen des différents routages. Le but de l'algorithme de sélection est d'allouer dynamiquement aux différents scénarios une quantité de simulations, pour prendre la meilleure décision possible avec un nombre de simulations fixé (une simulation pour un scénario prend environ 25 minutes d'après les auteurs, une bonne allocation du temps de calcul est donc primordiale). L'algorithme utilisé est une implémentation de l'algorithme OCBA [6] (Optimal Computing Budget Allocation). Une des difficultés du problème est bien sûr la mise au point de l'algorithme de simulation. Les auteurs ont utilisé l'outil de simulation discrète Arena, développé par Rockwell Automation, pour modéliser les vols.

Le problème de l'évaluation des programmes de courses de galop est assez similaire dans sa présentation. On aimerait, parmi un éventail de programmes, pouvoir déterminer celui qui attirera le plus de partants de façon robuste. Pourvu que l'on arrive a déterminer un bon modèle, une méthode basée sur la simulation peut tout à fait être une option pour estimer l'attractivité d'un programme donné. À l'inverse du problème de sélection de routages, en revanche, le mécanisme à l'origine de la participation (les décisions des entraîneurs) est difficile à simuler et il est invraisemblable que l'on puisse obtenir une simulation fidèle à la réalité. En ce qui concerne l'algorithme de choix, notre contexte est différent puisque qu'une simulation de la participation aux courses d'une population de chevaux serait a priori rapide, et que le temps d'exécution de l'algorithme de planification n'est pas critique, la planification

étant effectuée une fois tous les trois mois. Nous serions plus intéressés par une façon d'intégrer l'évaluation par simulation des solutions au processus de recherche effectué par une méta-heuristique, pour le guider, ce qui sera étudié en section 5.7.

Les travaux d'apprentissage statistique présentés au chapitre 3 s'inscrivent dans cette démarche de gestion de l'incertitude, et le problème de sélection de routes est celui qui, à notre connaissance, se rapproche le plus du nôtre dans sa problématique d'estimation de la qualité des solutions.

2.4 Planification de tâches avec périodicité

La contrainte d'écart est, comme expliquée en section 1.4.4.1, la contrainte la plus centrale et structurante de la planification des courses de galop. Cette contrainte impose à chaque type de course (que l'on peut considérer comme des tâches) d'être planifié plusieurs fois par programme, plus ou moins régulièrement. Le programme étant trimestriel, la planification est effectuée quatre fois par an. Ce processus de planification est donc réitéré chaque trimestre, tous les ans. Ainsi, notre problème pourrait s'apparenter à un problème de planification cyclique, c'est-à-dire avec un graphe des tâches infini et périodique.

2.4.1 Problème d'Ordonnancement Cyclique de Base

Un tel problème est le Problème d'Ordonnancement Cyclique de Base (BOCP). On se donne un ensemble $T = T_1, T_2, ..., T_q$ de tâches non préemptives, et on note $T(i,n)$ la n-ième exécution de la tâche T_i. On note p_i la durée de la tâche T_i.

Les contraintes de précédence se formulent alors sous la forme d'un ensemble P de triplets (T_i, T_j, k), avec k un entier naturel. Cet ensemble P définit l'ordre

$$(T_i, T_j, k) \in P \Rightarrow (\forall n \geq 1) : T(i,n) < T(j, n+k)$$

c'est-à-dire que la $(n+k)$-ième tâche T_j doit être planifiée après la n-ième tâche T_i. Une solution au BOCP est alors un ensemble des dates de départ $s(i,n)$ pour les tâches $T(i,n)$ qui respecte

$$(T_i, T_j, k) \in P \Rightarrow (\forall n \geq 1) : s(i,n) + p_i \leq s(j, n+k)$$

Dans [7], Philippe Chrétienne s'intéresse aux valeurs maximales que peuvent prendre les $s(i,n)$, lorsque l'on impose des deadlines, c'est-à-dire des bornes supérieures pour les $s(i,n) + p_i$.

Dans [8], l'auteur s'intéresse à la borne de Graham du problème, c'est-à-dire aux performances que l'on peut obtenir avec une heuristique de planification basée sur une liste de priorités. Une généralisation du problème avec un nombre limité de machines (de tâches simultanées) est étudiée.

D'autres généralisations du problème existent. Par exemple, dans [9], Alix Munier Kordon considère que les contraintes de précédence sont des quadruplets (T_i, T_j, k_e, p_e), où k_e et p_e sont des entiers relatifs et

$$(T_i, T_j, k_e, p_e) \in P \Rightarrow (\forall n \geq 1) : s(i, n) + p_e \leq s(j, n + k_e)$$

Notamment, le temps à respecter entre deux tâches n'est plus une durée d'exécution et peut varier suivant la contrainte. Elle établit la periodicité de la solution « au plus tôt » dans le cadre de cette généralisation.

Le BOCP peut être utilisé pour modéliser une chaîne de production, dans laquelle la production d'un produit nécessiterait d'effectuer les tâches de T. Le but est donc de planifier chaque tâche de T exactement n fois pour produire n produits.

Dans le problème de planification des courses hippiques, on pourrait tout à fait considérer que les différents types de courses à planifier sont les tâches de T, quitte à inclure dans T plusieurs fois les mêmes types de courses, pour les types de courses devant être le plus représentés dans un programme. Cependant, un des objectifs est de remplir au maximum le calendrier des réunions, il est alors impossible de fixer le nombre de courses de chaque type devant être présent dans chaque programme. En ce qui concerne la contrainte d'écart, elle énonce qu'après la planification d'une course du type T_i, on doit trouver une autre course de type T_i environ trois semaines plus tard. Elle n'interdit pas, en revanche, qu'il y ait d'autres courses de type T_i planifiées entre temps. Il est donc impossible d'énoncer cette contrainte sous la forme $T(i, n) < T(i, n + k)$, puisque la course située trois semaines (environ) après la n-ième course de type T_i n'est pas la $(n+k)$-ième pour k fixé. D'une manière générale, la planification des courses de chevaux se distingue de beaucoup de problèmes d'ordonnancement du fait de la variabilité de la liste des tâches à planifier dans l'horizon de planification, et de l'impossibilité de modéliser la contrainte d'écart à l'aide d'un graphe de précédence figé.

2.4.2 Recurrent Job Shop

Le BOCP peut aussi être généralisé pour donner le Recurrent Job Shop (RJS). L'énoncé du problème est le même que précédemment, sauf que chaque tâche T_i est associée à une unique machine M_i parmi la liste $M = \{1, \ldots, m\}$ des machines, capables d'exécuter au plus une tâche à la fois. Dans [10], l'auteur étudie le problème consistant à déterminer la période minimale d'une solution périodique au problème. Dans le cas où l'on relâche les contraintes portant sur les machines, on se ramène au BOCP, et le problème est polynomial. Dans le cas général, le problème est bien sûr NP-difficile, puisqu'il contient le job shop comme sous-problème. L'auteur démontre que le problème reste NP-complet lorsque le graphe conjonctif des contraintes de précédence (i.e. dans lequel les tâches $T(i, n)$ sont réduites à un seul point t_i) est un circuit.

En raison de la difficulté du problème, des méthodes méta-heuristiques sont souvent utilisées pour trouver des solutions heuristiques de qualité au problème. Dans [11], un recuit simulé est utilisé pour produire des ordonnancements pour plusieurs itérations sur des instances de job-shop d'OR-Library. Les durées des solutions produites pour 1, 2, ou 4 itérations sont comparées avec une répétition des meilleures solutions connues pour une itération. Sur toutes les instances présentées, le recuit produit une meilleure solution que la simple répétition de la meilleure solution connue pour une itération, et ce de manière plus marquée sur 4 itérations que sur 2. Cela montre que le recuit simulé est capable de tirer parti de la périodicité du problème pour produire de meilleures solutions.

Bien sûr, comme précédemment et pour les même raisons, le RJS ne permet pas de modéliser notre problème de planification. En revanche, on peut espérer qu'un recuit simulé pourra dans notre problème également tirer parti de la nature périodique de notre problème, et rester efficace sur un horizon de planification important.

Chapitre 3

Apprentissage et évaluation d'un programme de courses

Contenu

3.1 Introduction

Dans cette partie, on étudie l'évaluation de la qualité d'un programme de courses donné. Cette évaluation présente plusieurs difficultés : d'une part celles liées à la satisfaction des contraintes souples du problème qui peuvent s'apparenter à des critères empiriques pour jauger la qualité des solutions et d'autre part celles associées au souhait des décideurs d'atteindre un nombre de partants suffisant pour organiser des paris lucratifs. Le nombre de partants des courses n'est pas connu à l'avance et dépend non seulement de la solution de planification proposée mais également de nombreux autres facteurs imprévisibles. Anticiper le nombre de partants des courses est donc difficile. Pour estimer le nombre de partants attendus pour une solution donnée, nous allons avoir recours à des techniques d'apprentissage et mettre ainsi à profit les données de participation dont dispose France Galop. Nous montrerons dans ce chapitre comment l'apprentissage sera utilisé pour produire un estimateur du nombre de partants. Cet estimateur pourra ensuite être utilisé comme fonction objectif « boîte noire » pour guider le processus de planification. Tout d'abord on présente en section 3.2 les données dont nous disposons, et celles que nous utiliserons. Nous présenterons ensuite les travaux réalisés en début de thèse en section 3.3, portant sur l'utilisation de techniques génériques de classification et de régression. Après avoir détaillé la sélection des données pertinentes et le prétraitement effectué, on expliquera le principe de la validation croisée, méthode utilisée pour évaluer la qualité des prédictions obtenues. Chaque algorithme est alors présenté, et on commentera les résultats numériques. Finalement, en partie 3.4, sera présentée une méthode de prédiction du nombre de partants aux courses basée sur la simulation de la participation d'une population de chevaux dans un programme de courses.

L'utilisation du respect des contraintes comme mesure de la qualité d'un programme sera abordée au chapitre 5.

3.2 Données d'apprentissage

France Galop a mis à notre disposition une base de données complète des courses passées. Cette base contient les caractéristiques de chaque course, comme présentées en section 1.3, mais aussi les dotations distribuées, les chevaux participants et leur classement. Cette base fournit également des informations sur les chevaux, notamment leur âge et l'historique de leur valeur.

Les données disponibles remontent très loin dans le temps, mais les données datant de plus de

quelques années sont difficilement exploitables. Tout d'abord, la population de chevaux à l'entraînement évolue. Il y a sensiblement moins de chevaux à l'entraînement aujourd'hui qu'il y a 10 ans. D'autre part, les modalités de planification ont également été modifiées. Enfin, un changement de nomenclature s'opère depuis 2017. En effet, un premier moyen simple et efficace pour augmenter le nombre de partants aux courses a consisté à réduire le nombre de courses planifiées en regroupant des catégories de courses et la nomenclature des courses a donc changé. Ce processus de changement de nomenclature a débuté en 2017 pour les courses dédiées aux chevaux de 2 ans et se poursuit actuellement pour d'autres catégories d'âge. Il est donc assez difficile et probablement peu pertinent d'utiliser des données antérieures à 2017. De plus, la pandémie ayant fortement impacté les courses de 2020, les données de 2020 sont inutilisables.

Par ailleurs, la planification d'hiver est très différente de celle des autres mois puisque l'essentiel des courses de décembre à février se déroule dans le sud de la France, à Pau notamment dans ce qu'on appelle des *meetings*, pour aller chercher la chaleur. Les données d'hiver sont donc peu utiles pour l'apprentissage portant sur les autres mois, et on ne s'occupera de toute façon pas de la planification d'hiver. L'outil n'a pas, pour l'instant, vocation à planifier les meetings.

En début de thèse, nous avons exploité les données associées aux trimestres non hivernaux des années 2016 et 2017 via des méthodes d'apprentissage basées sur la classification et la régression (présentées en section 3.3). Les résultats n'ayant pas été concluants, ces pistes ont été abandonnées. Nos travaux de recherche se sont poursuivis par une exploitation plus large des années disponibles en fin de thèse dans le cadre d'une approche par simulation, présentée en section 3.4. Pour la simulation, nous utiliserons plutôt les données de 2017 à 2019. Cela représente 9736 courses. Cependant, puisque la simulation, on le verra, attribue un score à un programme et non à une course, il peut être pertinent de considérer les programmes de trois mois comme des points de données, plutôt que les courses individuelles. Si l'on exclut le trimestre d'hiver, on est alors réduits à 3 trimestres sur 3 ans, ce qui est bien sûr exceptionnellement peu.

Pour exploiter les données disponibles, nous avons aussi étudié les données avec de simples outils de visualisation ou statistiques. À titre d'exemple, la figure 3.1 représente la répartition du nombre de partants par course sur les 9736 courses étudiées. Nous nous sommes intéressés à la répartition des gains suivant le type de course, à la répartition géographique et démographique des chevaux, ou encore à l'évolution de leur niveau au cours d'une année, et à leur fréquence de participation. Par exemple, on

observe logiquement que les courses à support d'événement attirent beaucoup de partants. On observe aussi des disparités entre les fédérations, ou encore que le nombre de courses planifiées varie au cours de l'année.

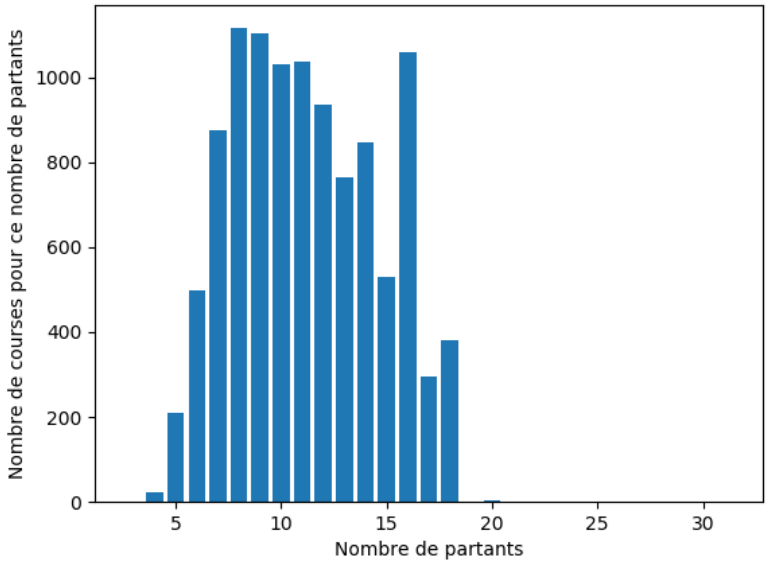


FIGURE 3.1 – Répartition du nombre de partants entre janvier et novembre 2019

Ces statistiques nous ont aidé à cerner les données intéressantes, celles qui peuvent influencer le nombre de partants, et que nous pourrions utiliser pour améliorer les programmes générés par l'outil. En effet, certaines données pourraient être corrélées au nombre de partants, comme par exemple l'ordre des courses au sein d'une réunion, ou la dotation d'une course, mais sur lesquelles l'outil n'a aucun degré de liberté (pour l'instant en tout cas).

Nous avons choisi de présenter ces statistiques au fil de l'eau pour expliquer certaines des décisions prises au cours du manuscrit, plutôt que de les rassembler ici. Quelques statistiques complémentaires sont également fournies en annexe (chapitre 5.7.2).

3.3 Classification et régression

3.3.1 Données utilisées, caractéristiques d'une course

Pour appliquer les méthodes d'apprentissage présentées dans ce chapitre, on décrit chaque course par un vecteur de données numériques appelées caractéristiques (« features » en anglais). Le choix des caractéristiques d'une course est déterminant. Parmi les informations disponibles dans la base de données, les caractéristiques suivantes ont été identifiées comme intéressantes. Attention, comme il s'agit d'un travail effectué avant/pendant le changement de nomenclature de 2017, la terminologie ne correspond pas exactement aux différentes catégories présentées en introduction.

— La condition d'âge sur la course, suivant les 6 catégories suivantes : 2 ans, 3 ans, 3 ans et plus, 4 ans, 4 et plus, 5 ans et plus.
— La condition de sexe : mâles, femelles, ou sans condition.
— Le type de course, parmi : Listed, Condition sup., Condition inf., Handicap sup., Handicap inf., Réclamer sup., Réclamer inf., Inédits et Maiden.
— La distance, parmi : Sprinter, Miler, Classique, Intermédiaire et Stayer.
— Si la course est dédoublée ou non, c'est-à-dire fait partie d'un lot de courses identiques ayant lieu pendant la même réunion.
— La fédération dans laquelle se déroule la course.
— Le jour de la semaine où se déroule la course
— Le mois dans lequel se déroule la course.

Pour mettre ces différentes caractéristiques sous forme numérique, on pourrait par exemple, pour les âges, affecter la valeur 0 aux courses de « 2 ans », la valeur 1 aux courses « 3 ans », etc... Une telle description présume cependant assez arbitrairement qu'une course « 3 ans » (codée 1) est plus proche dans l'espace des caractéristiques d'une course « 2 ans » (codée 0) que d'un course « 4 ans » (codée 3), les autres caractéristiques étant égales par ailleurs. Cela peut impacter négativement les résultats. Nous allons donc plutôt utiliser un prétraitement de type « One Hot ». Au lieu d'avoir une unique caractéristique « âge », avec 6 valeurs numériques possibles, nous allons avoir 6 caractéristiques « 2 ans », « 3 ans », etc... qui prennent pour valeur 1 si on est dans cette catégorie et 0 sinon. L'appellation « One Hot » vient du fait que seule une des ces 6 catégories d'âge peut prendre la valeur 1 pour une course. Ce traitement est appliqué à chacune des caractéristiques ci-dessus.

On peut remarquer qu'aucune des caractéristiques retenues ne dépend de l'enchaînement des courses dans le programme. Une course a les mêmes caractéristiques (et donc la même prédiction) qu'elle soit seule dans le programme ou avec des courses similaires programmées à proximité. Cela est gênant bien sûr, mais aucune des caractéristiques supplémentaires testées utilisant les courses similaires programmées à proximité n'a été bénéfique. Par exemple, nous avons essayé de prendre comme caractéristique le nombre de courses du même type programmées à une distance de moins d'une semaine, mais cela n'a pas été intéressant. Nous reviendrons sur ce point plus tard.

3.3.2 Définition du problème d'apprentissage

La question à laquelle on veut répondre est « Quel sera le nombre de partants de la course c ? ». Il existe plusieurs familles de problèmes d'apprentissage ; et cette question peut se ranger dans deux d'entre elles.

— Dans un problème de type régression, on cherche à prédire une quantité continue, par exemple une température. Une prédiction sur un nombre de partants pourrait alors ressembler à « Il y aura 12,3489 partants pour la course c ». La qualité de la régression est évaluée en considérant les écarts entre la prédiction et la réalité. Cela étant dit, un nombre de partants est une quantité entière. On pourra arrondir le résultat de la prédiction si l'on veut.

— Dans un problème de type classification, on cherche à associer une classe ou une catégorie à chaque point de donnée (pour nous les courses). Il peut s'agir d'un problème de classification binaire, c'est-à-dire avec seulement deux classes. Par exemple : déterminer si une image contient un visage ou non. On peut également avoir un problème dit en classes multiples avec un nombre arbitraire (fini) de classes. Pour notre problème, la réponse se formulerait alors sous la forme « La course c appartiendra à la classe "12 partants" ». Pour évaluer la qualité d'un classifieur (fonction qui effectue la classification), on peut regarder le pourcentage de bonnes prédictions (précision du classifieur). En pratique, le classifieur renvoie une probabilité d'appartenir aux différentes classes. On peut donc aussi mesurer la qualité d'un classifieur avec des méthodes de type entropie.

Dans un premier temps, nous avons testé essentiellement des classifieurs, car ils nous semblaient plus adaptés au problème des courses de chevaux. En effet, ce qui est réellement important n'est pas le nombre exact de partants, mais les paris qui vont être organisables sur chaque course. Par exemple,

les courses avec plus de 13 partants pourront accueillir les paris dits complexes, et on voudra plutôt savoir si une course aura plus de 13 partants ou pas que de savoir si elle aura 14 ou 15 partants. Nous avons ensuite comparé avec la régression pour vérifier cette intuition. Avant de présenter les deux approches, il faut cependant avoir une méthode pour évaluer correctement la qualité des résultats. Nous allons donc présenter la méthode de test utilisée pour éviter les écueils du sur-apprentissage et du sous-apprentissage.

3.3.3 Méthode de test

Nous cherchons à déterminer une fonction qui à une course associe la prédiction d'un nombre de partants. Pour ce faire, nous disposons de données sur le passé. Nous connaissons pour chaque course passée le nombre de chevaux qui ont participé à la course (le nombre de partants). Une bonne fonction de prédiction doit donc, pour ces courses dont on connaît le nombre de partants réel, prédire un nombre de partants proche de la réalité. C'est ce qu'on appelle un problème d'apprentissage supervisé. En revanche, il ne suffit pas qu'une fonction fasse des prédictions fidèles à la réalité sur les données passées pour être intéressante, il faut aussi qu'elle fasse de bonnes prédictions sur les courses à venir.

Illustrons cette problématique sur l'exemple imagé suivant. Imaginons que suite à une étude, je connaisse les coordonnées géographiques d'un certain nombre d'habitations françaises, bien réparties sur le territoire, ainsi qu'un indicateur de la qualité de leur eau courante. Je souhaite prédire la qualité de l'eau courante d'une habitation non référencée dans ma base de données pour laquelle je ne connais que les coordonnées géographiques.

On propose trois fonctions de prédiction.

— La première fonction consiste à regarder quelle est la qualité moyenne de l'eau courante en France sur les maisons de ma base de données et de supposer qu'une nouvelle maison qui n'est pas dans la base aura un indicateur de qualité d'eau similaire.

Cette fonction n'est bien sûr pas très intéressante. Nous n'avons pas utilisé les coordonnées géographiques des maisons de la base de données, il y a fort à parier qu'il est possible de trouver une fonction plus performante en les utilisant.

— La seconde fonction que l'on propose est la suivante. Pour prédire la qualité de l'eau courante d'une nouvelle maison, on regarde dans la base de données quelle est la maison dont on connait la qualité de l'eau qui est la plus proche géographiquement de la nouvelle maison, et on prédit

que la nouvelle maison aura la même qualité d'eau.

Cette fonction peut sembler efficace au premier abord. Si par exemple la nouvelle maison est en fait une maison qui est déjà dans la base de données, alors la prédiction sera parfaite. On pourrait donc avoir l'impression que cette fonction prédit parfaitement la qualité de l'eau. Cependant une maison peut très bien avoir une eau courante de bonne qualité parce qu'elle a installé un purificateur, ou une eau de mauvaise qualité à cause d'un problème de plomberie. Il n'est pas judicieux de supposer que deux maisons voisines ont forcément la même qualité d'eau courante.

— La troisième fonction que l'on propose est hybride entre les deux premières. Cette fois, pour prédire la qualité de l'eau d'une nouvelle maison, nous prenons la moyenne de la qualité des eaux des maisons de la base de donnée qui sont dans les 5 kilomètres à la ronde.

Ainsi, on a une fonction qui exploite correctement les données disponibles, contrairement à la première fonction, mais on s'est aussi prémunis contre la présence de données de mauvaise qualité dans la base de données, contrairement à la seconde fonction.

Dans l'absolu, la seconde fonction est celle qui minimise le mieux l'écart entre les prédictions et les données réelles quand on regarde les maisons de la base de données, et ce n'est pourtant pas la fonction de prédiction la plus intéressante. Si maintenant on s'intéresse seulement aux fonctions de prédiction constantes, qui prédisent la même qualité d'eau pour toutes les maisons, alors la fonction qui minimise le mieux l'écart entre les prédictions et les données réelles est la première fonction. Pour que la minimisation de l'écart entre les prédictions et les données réelles produise la troisième fonction, il faut donc chercher parmi un espace de fonctions suffisamment grand pour qu'il contienne la troisième fonction, mais suffisamment petit pour qu'il ne contienne pas la seconde.

Une méthode d'apprentissage qui produirait la première fonction fait ce qu'on appelle du sous-apprentissage. Une méthode d'apprentissage qui produirait la seconde fait ce qu'on appelle du sur-apprentissage.

Dans la suite du chapitre, nous présentons plusieurs méthodes d'apprentissage et nous aimerions nous assurer que nous ne faisons ni du sous-apprentissage, ni du sur-apprentissage. Nous allons pour cela utiliser une méthode de test appelée la validation croisée. Parmi les données disponibles, nous allons seulement en utiliser une partie pour produire la fonction de prédiction, que l'on appelle ensemble d'apprentissage. Les données non utilisées pour l'apprentissage sont l'ensemble de test. Une

fois qu'une fonction est produite par la méthode d'apprentissage en utilisant les données de l'ensemble d'apprentissage, on applique la fonction à l'ensemble de test, et on regarde si les prédictions sont proches de la réalité sur cet ensemble de test. Si la fonction de prédiction est significativement plus performante sur l'ensemble d'apprentissage que sur l'ensemble de test, il est probable que nous ayons fait du sur-apprentissage, c'est un signe que la fonction produite n'est pas efficace sur des données nouvelles. À l'inverse, si la fonction est performante sur l'ensemble de test, la fonction de prédiction se généralise bien sur des nouvelles données, il n'y a pas de sur-apprentissage. Nous allons donc, pour chaque méthode d'apprentissage, présenter des résultats sur l'ensemble d'apprentissage et sur l'ensemble de test, et les comparer, pour justifier de la pertinence de la méthode. Cette idée est souvent utilisée également pour déterminer les hyperparamètres des méthodes d'apprentissage produisant le moins de sur-apprentissage, à l'aide d'un grid-search.

Il reste le problème du sous-apprentissage, et c'est pour cela que nous allons tester plusieurs méthodes pour pouvoir sélectionner la plus performante et la plus adaptée au problème.

Remarquons par ailleurs, en reprenant l'exemple de la qualité de l'eau, que si nous devions prédire la qualité de l'eau dans une maison située à l'étranger, il va de soi que nous n'aurions pas les données nécessaires pour faire une bonne prédiction. Cela suggère que les données connues doivent être bien réparties, parmi toutes les coordonnées d'entrées possibles. Cette remarque prend toute son importance dans le sujet qui nous préoccupe, l'évaluation d'un programme de courses. En effet, toutes les données disponibles proviennent de programmes produits à la main suivant une méthode précise et ne sont a priori pas représentatives de tous les programmes qu'il est possible d'élaborer.

3.3.4 Classification par régression logistique

3.3.4.1 Énoncé du problème de classification

Considérer la problématique de prédiction du nombre de partants comme un problème de classification présente un intérêt tout particulier vis-à-vis du cahier des charges. En effet, il est possible de séparer les courses suivant trois classes :

— La classe 0 : « moins de 8 partants ».

— La classe 1 : « entre 8 et 13 partants inclus ».

— La classe 2 : « plus de 13 partants ».

Ces classes correspondent aux nombres de partants nécessaires pour organiser les différents types de

paris. Avoir un nombre de classes plus réduit permet a priori d'obtenir un classifieur plus efficace. Il est bien sûr possible d'affiner la prédiction et d'avoir 4 classes en séparant par exemple la classe intermédiaire en deux. Cependant on aura très probablement des erreurs de classifications supplémentaires entres ces deux classes. Ce n'est donc pas nécessairement préférable. Essentiellement, la question est de savoir si la différence des bénéfices financiers entre deux classes compense les pertes par erreur de prédiction. Hypothétiquement, si la différence de recettes entre une course avec 15 partants et une avec 16 partants est très faible, il n'est pas forcément intéressant d'essayer de les dissocier. Le choix par défaut a donc été de se référer au cahier des charges pour séparer les courses en trois catégories.

3.3.4.2 Présentation de la méthode

Dans notre problème, la régression logistique peut être utilisée pour prédire la probabilité d'appartenir à chacune des classes. Ces probabilités sont des données numériques continues, il est donc adapté d'utiliser une méthode de régression pour les prédire. On peut appliquer la régression logistique pour faire une classification binaire, avec deux classes, auquel cas on n'a qu'une seule probabilité à prédire, mais on peut aussi l'adapter dans le cas d'une classification à classes multiples, avec plus de classes, trois dans notre problème. Donnons une explication intuitive de la régression logistique appliquée à une classification, les détails du modèle sont présentés en annexe, en section 5.7.2. Intuitivement, on associe à chaque caractéristique un coefficient qui indique dans quelle mesure la caractéristique en question est en faveur de la classe 1. Cependant, imaginons que pour un échantillon, un critère en faveur de la classe 1 soit présent et aucun en faveur de la classe 0. On en déduit qu'il y a des chances raisonnables que l'échantillon appartienne à la classe 1. Maintenant si il y a deux critères en faveur de la classe 1 et aucun contre, on est tenté de penser que la probabilité d'appartenir à la classe 1 est bien plus que deux fois plus élevée que dans le cas précédent. En quelque sorte, la régression logistique fait la supposition, que les différents critères pour ou contre une classe fonctionnent de façon multiplicative plutôt qu'additive. Dans notre problème, on s'attend par exemple à ce que la caractéristique « Handicap » pèse en faveur de la classe « plus de 13 partants » , la classe 2, car on observe que les courses à Handicap attirent souvent plus de 13 partants. Inversement, les courses « C1 » ont souvent peu de partants, la caractéristique « C1 » pèsera donc probablement en faveur de la classe 0.

3.3.4.3 Résultats numériques de la régression logistique

Nous avons appliqué la régression logistique à notre problème de prédiction du nombre de partants. L'implémentation a été faite avec le classifieur LogisticRegressionCV du paquet Python scikit-learn [1]. Une validation croisée 3-fold a été appliquée. Cela signifie qu'on a divisé l'ensemble des courses de l'ensemble d'apprentissage en 3 parts équilibrées, et que chacune de ces parts a assumé à tour de rôle la fonction d'ensemble de test, pour déterminer les hyperparamètres faisant le moins de sur-apprentissage.

Les résultats numériques sont présentés sous la forme d'une matrice de confusion. Une ligne de la matrice est associée à la classe réelle de la course en fonction du nombre de partants, chaque colonne à la classe prédite. Les valeurs sur la diagonale correspondent donc au nombre de bonnes prédictions.

Précisons que nous avons pris toutes les courses de mars à novembre 2016 comme ensemble d'apprentissage. Notons que sur les courses de mars à novembre 2016, dans notre ensemble d'apprentissage, il y a 463 (soit environ 17,1%) courses avec moins de 8 partants, 1416 (soit environ 52,3%) courses avec entre 8 et 13 partants, et 826 (soit environ 30,5%) courses avec plus de 13 partants, soit un total de 2705 courses. Ainsi, un classifieur prédisant tout le temps la classe 1, c'est-à-dire un nombre de partants dans l'intervalle [8 ;13], aurait déjà 52,3% de précision.

On obtient les résultats suivants en prédiction sur les courses de mars à novembre 2016 :

true\pred	<8	[8 ;13]	>13
<8	355	103	5
[8 ;13]	332	821	263
>13	21	206	599

TABLE 3.1 – Régression logistique sur l'ensemble d'apprentissage

Ainsi sur la dernière colonne on lit que parmi les courses pour lesquelles on a prédit qu'il y aurait plus de 13 partants, 599 ont effectivement eu plus de 13 partants, alors que 263 ont eu entre 8 et 13 partants, et que 5 ont eu moins de 8 partants. Sur la première ligne, on lit que parmi les courses ayant eu moins de 8 partants, on en a identifié 355 correctement, 103 comme des courses avec entre 8 et 13 partants et 5 comme des courses avec plus de 13 partants. En regardant le nombre de courses sur la diagonale et en le comparant au nombre total de courses, on peut également lire qu'il y a eu environ

1. `http://scikit-learn.org/stable/`, licence BSD

65,6% de bonnes prédictions, c'est-à-dire de précision.

Testons maintenant les performances du classifieur sur les courses de mars et avril 2017. Dans cet ensemble, il y a 61 (soit environ 11,9%) courses avec moins de 8 partants, 283 (soit environ 55,4%) courses avec entre 8 et 13 partants, et 167 (soit environ 32,7%) courses avec plus de 13 partants, soit un total de 511 courses. On obtient les résultats suivants (environ 62,6% de précision) :

true \pred	<8	[8 ;13]	>13
<8	29	32	0
[8 ;13]	64	178	41
>13	9	45	113

TABLE 3.2 – Régression logistique sur l'ensemble de test

Ces résultats nous mènent aux premières conclusions suivantes :

— La validation croisée semble avoir été plutôt efficace, puisque les précisions sur les ensembles d'apprentissage et de test sont assez proches l'une de l'autre, le classifieur est donc en un sens fiable.

— Les bonnes prédictions sur les moins de 8 partants sont assez peu nombreuses en prédiction sur 2017. Cette différence de résultats entre 2017 et 2016 apparaît uniquement pour cette classe.

L'irrégularité des résultats observée sur la classe des moins de 8 partants peut être mise en lien avec le changement de nomenclature des courses opéré par France Galop entre décembre 2016 et février 2017. De simples statistiques tendent à montrer que ce changement de nomenclature a contribué à une augmentation du nombre de partants pour les courses à conditions pour les chevaux de 2 et 3 ans. On le voit bien dans les données utilisées (11,9% contre 17,1% de courses avec moins de 8 partants). Il serait logique alors qu'un apprentissage sur les courses « pré-changement » ait quelques difficultés à prédire les courses « post-changement ». Nous verrons si cette difficulté de prédiction des courses avec moins de 8 partants se confirme avec les autres méthodes d'apprentissage. Alors, si le changement de nomenclature est effectivement une source de difficultés pour l'apprentissage, on peut tout à fait espérer que la qualité du classifieur puisse être améliorée lorsque l'on disposera de plus de données avec la nouvelle classification.

3.3.4.4 Conclusion sur la régression logistique

La régression logistique suppose que chaque caractéristique a une influence indépendante de la valeur des autres caractéristiques sur la probabilité d'appartenir à telle ou telle classe (voir modèle en annexe). La régression logistique est donc sensible à la façon dont l'on détermine le vecteur des caractéristiques d'une part, mais n'est surtout pas adaptée à tout type de problème. Par exemple, il est possible (hypothétiquement) que les courses de 3 ans soient populaires dans le sud-ouest, et boudées en Bretagne. La caractéristique « 3 ans » aurait alors une influence inverse suivant la valeur des caractéristiques « Bretagne » et « Sud-Ouest ». Il est impossible de rendre compte d'un comportement de ce type avec une régression logistique. Ainsi, l'utilisation d'un modèle inadapté cause un sous-apprentissage.

Cette limite du pouvoir prédictif d'une régression logistique peut être une des raisons pour lesquelles les résultats sont assez peu satisfaisants. Il est donc souhaitable d'essayer un classifieur plus puissant pour évaluer la qualité du régresseur logistique.

Concernant l'examen des valeurs des coefficients de la régression, on observe par exemple que les courses en fédération Paris (fédération 0) marchent bien mieux que les courses en fédération Sud-Ouest (fédération 6), ou encore que les courses supports d'événements marchent beaucoup mieux que les courses à conditions. Ces résultats sont intéressants, mais se contentent de confirmer ce qui est accessible par de simples statistiques.

Puisque le pouvoir prédictif de la régression logistique est limité, et qu'elle n'apporte que peut d'information par rapport à de simple statistiques, nous n'avons pas continué l'étude de cette méthode.

3.3.5 Classification par gradient boosting

3.3.5.1 Présentation de la méthode

Pour obtenir de meilleurs résultats, nous utilisons un classifieur au pouvoir expressif plus important. Il s'agit de la classification par gradient boosting. Par plus expressif, on entend qu'il est possible en théorie d'apprendre des choses plus complexes avec un gradient boosting qu'avec une simple régression linéaire. Ce pouvoir d'expression est dû à l'utilisation d'arbres de décision. Pour comprendre le gradient boosting, commençons par nous intéresser aux arbres de décision.

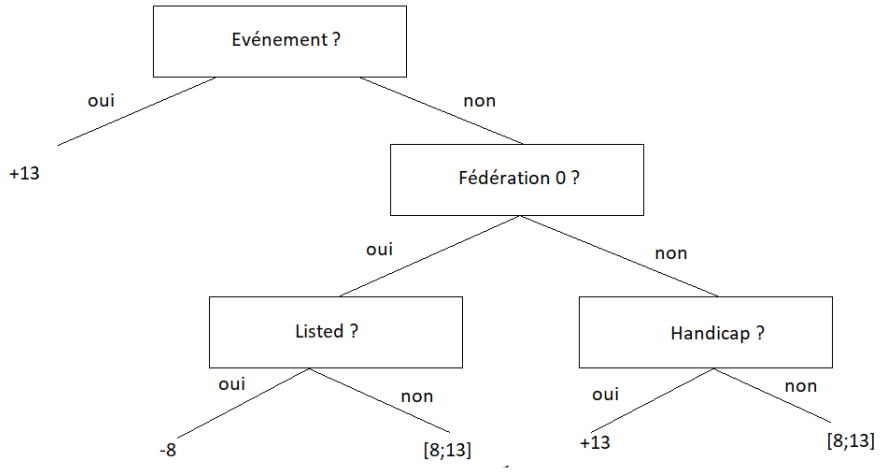


FIGURE 3.2 – Exemple d'arbre de décision appliqué au problème de courses

Arbres de décision Pour illustrer l'intérêt des arbres de décision, considérons le problème suivant. On connaît le groupe sanguin d'un individu et son rhésus, et l'on veut savoir si il peut recevoir du sang B+. Le vecteur des caractéristiques de mon individu est le suivant : (« A », « B », « + »). Par exemple un vecteur (1 ;1 ;0) correspondrait à du sang AB- alors que (0 ;0 ;1) serait du O+. On veut distinguer deux classes : Compatible pour recevoir B+ (1) et pas compatible pour recevoir B+ (0). La démarche est très simple : on regarde tout d'abord si la caractéristique B de l'individu testé est à 1. Si non, on est dans la classe 0, « pas compatible ». Si oui, on regarde dans un deuxième temps la caractéristique « + » : si elle vaut « 1 », l'individu est compatible, si elle vaut « 0 », il ne l'est pas.

Cette simple procédure de décision consistant à examiner successivement les différentes caractéristiques de l'échantillon et à trancher de façon binaire suivant des seuils sur des caractéristiques s'appelle un arbre de décision. La question posée dans l'exemple ci-dessus, bien que simple, est pourtant impossible à traiter avec une régression logistique. Au mieux, il est aussi important d'avoir du B que d'avoir du + pour pouvoir recevoir du sang B+, mais alors un individu qui n'aurait qu'une des deux caractéristiques correcte (de groupe sanguin A+, par exemple) serait considéré comme ayant par exemple 30% de chances d'être compatible. Ce simple exemple montre donc bien l'intérêt et le pouvoir prédictif des arbres de décision.

Dans la figure 3.2, nous donnons un exemple d'arbre de décision de profondeur 3 appliqué à notre

true\pred	<8	[8 ;13]	>13
<8	461	2	0
[8 ;13]	22	1393	1
>13	2	45	779

TABLE 3.3 – Arbre de décision sur l'ensemble d'apprentissage

true\pred	<8	[8 ;13]	>13
<8	26	30	5
[8 ;13]	39	198	46
>13	2	57	108

TABLE 3.4 – Arbre de décision sur l'ensemble de test

problème de planification de courses de chevaux. On se demande en premier lieu si la course est à support d'événements. Si oui, la prédiction est plus de 13 partants. Si non, on passe à la question suivante, « la course se déroule-t-elle en fédération 0 ? ». Le développement de l'arbre continue de cette manière jusqu'à atteindre une feuille de ce dernier.

La puissance prédictive d'un arbre de décision dépend de sa profondeur. Commençons par considérer des arbres de décision de profondeur arbitraire (donc avec une grande puissance prédictive). Nous avons, toujours à l'aide de la bibliothèque scikit-learn, produit un arbre de décision pour notre problème. Les résultats de prédiction produits par l'arbre de décision pour la classification des courses de chevaux sont présentés dans les tableaux 3.3 et 3.4. Dans le tableau 3.3, toujours en apprentissage sur les courses de mars à novembre 2016, on obtient les résultats en prédiction sur les données d'apprentissage, puis dans le tableau 3.4 les résultats en prédiction sur les courses de mars à avril 2017.

On voit très clairement que si les résultats sont excellents sur l'ensemble d'apprentissage, il sont comparables aux résultats de la régression logistique sur l'ensemble de test (65% de précision, c'est-à-dire de bonnes prédictions). Autrement dit, l'arbre de décision a fait un sur-apprentissage très important.

Ainsi, les arbres de décision présentent plusieurs avantages : ils sont simples à appréhender, rapides, demandent peu de calculs pour faire une prédiction, et ont un pouvoir prédictif intéressant. Ils ont cependant deux défauts majeurs. Premièrement, ils sont très prompts au sur-apprentissage. En effet, il est très difficile d'apprendre efficacement sur de nouvelles données. Cela est dû au fait que pour

séparer 2^n individus (avec un vecteur de caractéristiques de taille n), il ne faut que n décisions. Même un petit arbre de décision a un pouvoir d'expression très important. De simples critères tels que limiter la profondeur de l'arbre, ou exiger que chaque feuille de l'arbre représente un nombre minimum de courses de l'ensemble d'apprentissage, peuvent être utilisés pour limiter ce sur-apprentissage. Ainsi, on peut avec de simples réglages, comme limiter la profondeur de l'arbre, atteindre une précision de 68%. Deuxièmement, trouver un arbre de décision d'une profondeur donnée optimal au sens de la qualité de la prédiction est un problème NP-difficile. Il existe de nombreuses heuristiques gloutonnes pour construire des arbres de décision, mais c'est tout de même un obstacle à l'utilisation efficace d'arbres de décision pour faire de l'apprentissage. De plus, l'arbre de décision ne fournit pas des probabilités d'appartenir à telle ou telle classe mais directement une prédiction. Pour chaque course du programme, on applique les décisions de l'arbre de décision, et on lit sur la feuille atteinte la classe prédite.

Heureusement, il existe plusieurs algorithmes pouvant pallier les défauts des arbres de décision tout en gardant leurs qualités, comme le gradient tree boosting [12].

Gradient tree boosting Une méthode classique pour limiter le sur-apprentissage, notamment des arbres de décision, est le *bagging*. Le bagging consiste à construire plusieurs classifieurs (ici des arbres de décision) mais en n'utilisant à chaque fois qu'une partie tirée au hasard de l'ensemble d'apprentissage, avec remise (On tire au hasard parmi la totalité des échantillons à chaque fois ; un échantillon peut donc être tiré plusieurs fois). On agrège ensuite les différents classifieurs en un seul, typiquement à l'aide d'un vote, la classe ayant été prédite le plus grand nombre de fois l'emporte. Le bagging permet donc par exemple de diminuer l'impact des point aberrants, comme une course avec peu de partants à cause de la météo, et donc de réduire le sur-apprentissage.

Une autre méthode consiste à limiter la puissance d'apprentissage de l'arbre. On peut par exemple limiter la profondeur maximale de l'arbre, le nombre de caractéristiques utilisées etc... pour avoir ce qu'on appelle un classifieur faible. Cette deuxième technique est celle utilisée par le gradient boosting. Le gradient boosting appliqué aux arbres de décision, consiste à construire de nombreux arbres de décision faibles, puis de les agréger à l'aide d'un vote comme pour le bagging. Toute la subtilité réside donc dans la manière de générer ces arbres faibles. De façon simpliste, on les construit successivement, les uns à la suite des autres, en essayant d'apprendre ce qui n'a pas été appris correctement par les arbres précédents.

true\pred	<8	[8 ;13]	>13
<8	48	40	0
[8 ;13]	16	255	23
>13	1	56	102

TABLE 3.5 – Gradient boosting sur mars et avril 2016

Le gradient boosting utilise de nombreux paramètres. Il s'agit essentiellement, du nombre d'arbres utilisés, et de la façon de limiter l'apprentissage des arbres individuels. On utilisera de la validation croisée pour choisir les meilleurs paramètres, c'est-à-dire que l'on sélectionnera les paramètres produisant le moins d'erreur sur l'ensemble de test. Les paramètres adaptés dépendent beaucoup par exemple de la caractérisation de la course, on refait donc tout le temps le réglage pour chaque résultat présenté.

3.3.5.2 Résultats du gradient boosting

On fait toujours l'apprentissage sur les courses de mars à novembre 2016. On règle les paramètres en validation croisée sur les courses de mars à avril 2017. On fait tout d'abord une prédiction sur mars et avril 2016, c'est-à-dire sur un sous-échantillon de l'ensemble d'apprentissage.

La précision en prédiction est donc cette fois d'environ 75%. Les paramètres ayant été réglés en validation croisée, et le gradient boosting étant peu sujet au sur-apprentissage pour un choix de paramètres raisonnables, cette valeur est significative, même si la prédiction porte sur les données d'apprentissage. Outre le fait que la performance soit bien meilleure que celle de la régression logistique dans des circonstances similaires, la matrice de confusion est tout à fait typique de ce que l'on peut espérer. On a prédit une seule fois moins de 8 alors que c'était plus de 13, et aucune fois plus de 13 alors que c'était moins de 8. Les erreurs entre les classes 0 et 1 et entre les classes 1 et 2 sont inévitables, une course étant à la limite des deux classes sera difficile à prédire juste plus de la moitié du temps, puisque la variabilité du nombre de partants est importante. Il serait tout à fait satisfaisant de pouvoir obtenir de tels résultats de façon consistante.

Le tableau 3.6 présente les résultats obtenus en prédiction sur les courses de mars à avril 2017.

Les résultats sont très similaires aux précédents, excepté pour les bonnes prédictions sur les courses avec moins de 8 partants. Maintenant, lorsqu'on prédit moins de 8 partants, il s'agit quasiment la moitié du temps d'une course avec entre 8 et 13 partants. La précision chute à environ 71.8%, ce qui est toujours beaucoup mieux que la régression logistique. Ce résultat semble confirmer le soupçon que

true\pred	<8	[8 ;13]	>13
<8	17	44	0
[8 ;13]	16	242	25
>13	0	59	108

TABLE 3.6 – Gradient boosting sur mars et avril 2017

nous avions eu tantôt, concernant le changement de nomenclature. Cet impact est négatif vis-à-vis de notre capacité à prédire le nombre de partants des courses de 2017, mais est un signe encourageant, dans le sens où l'on peut espérer atteindre les 75% de précision comme sur les courses de 2016, une fois que l'on aura plus de données.

3.3.5.3 Conclusion sur la classification par gradient boosting

La classification en utilisant une méthode gradient boosting est assez prometteuse. Le problème de prédiction sur les courses avec moins de 8 partants subsiste, suggérant une influence importante du changement de nomenclature. Par contre, la précision est bien plus importante, en particulier en prédiction sur 2016, ce qui laisse penser qu'il existera une marge de progression lorsque suffisamment de données avec la nouvelle nomenclature seront disponibles.

Concernant le choix des caractéristiques ; nous utilisons pour faire ces prédictions uniquement les caractéristiques listées en partie 3.3.1. Ces caractéristiques ne décrivent que la nature de la course et non la façon dont elle a été planifiée. La prédiction est déjà efficace, mais si l'on veut automatiser la planification de façon à optimiser le nombre de partants, il serait important que le nombre de partants prédit dépende de la planification. L'ajout de caractéristiques autorisant une telle dépendance présente plusieurs difficultés. Notamment, pour pouvoir évaluer l'impact de la structure du programme sur le nombre de partants, il faut que les données d'apprentissage présentent des structures de programme variées. Ce n'est pas forcément le cas si toutes les données disponibles sont issues de la programmation manuelle. Nous avons essayé d'ajouter des caractéristiques tenant compte des courses programmées à proximité mais sans succès.

Enfin, concernant la fonction de prédiction obtenue par une méthode de type gradient boosting : il s'agit mathématiquement d'une fonction en escalier, de dimension égale au nombre de caractéristiques. Elle n'est pas continue, et ne présente aucune forme de monotonie a priori. Tout ce qu'il est possible de faire est de l'évaluer en un point. Pour une réunion, une catégorie de course, un âge et une distance,

la fonction de prédiction obtenue donne une probabilité d'appartenir aux différentes classes. Dans le chapitre 4, sur les méthodes exactes, nous allons voir qu'il est possible d'utiliser les valeurs obtenues comme coefficients d'une fonction objectif d'un programme linéaire. Il serait aussi possible d'utiliser la fonction de prédiction dans l'objectif de méthodes de planifications n'ayant pas besoin que la fonction objectif ait une expression analytique, comme des méthodes méta-heuristiques.

3.3.6 Exploitation d'une classification

La problématique de cette partie est de déterminer la meilleure façon d'intégrer la prédiction du nombre de partants à une fonction objectif. Intuitivement, on veut savoir quel bonus associer à une course avec « plus de 13 partants », à celle avec « moins de 8 partants » et à celle avec « entre 8 et 13 partants ». Ce bonus est en lien direct avec le bénéfice que génère une telle course. Soit $x \in \mathcal{X}$ un programme. Ce à quoi on s'intéresse est donc uniquement le bénéfice $\Gamma(x)$ généré par le programme. On veut donc trouver la fonction $f : \mathcal{X} \mapsto \mathbb{R}$ qui à un programme associe le revenu qu'il va générer.

3.3.6.1 Modèle

En pratique, le bénéfice $\Gamma(x)$ est une variable aléatoire, il n'est donc pas possible d'avoir une fonction f qui vaut exactement le bénéfice qui sera généré par x. On fait l'hypothèse que le nombre de partants d'une course c, indépendamment des autres courses éventuelles de x, appartient à la classe k avec une probabilité $p(k|c)$. Cette hypothèse d'indépendance est raisonnable car on peut inclure dans la description d'une course des caractéristiques sur la structure du programme. Le fait, par exemple, qu'une course avec beaucoup de courses similaires programmées à proximité ait éventuellement moins de partants n'est pas en contradiction avec cette hypothèse d'indépendance. Il s'agit d'une hypothèse d'indépendance vis-à-vis du nombre de partants des autres courses, et non pas vis-à-vis de leur planification. On suppose également qu'une course de classe k (par exemple moins de 8 partants, ou plus de 13 partants) génère un bénéfice fixe et connu γ_k. On a alors, en notant Γ_c la variable aléatoire associé au gain de la course c,

$$p(\Gamma_c = \gamma_k) = p(k|c)$$

Le gain espéré pour la course c est donc tout simplement la moyenne des gains des différentes classes, pondérée par la probabilité que c a d'appartenir aux dites classes.

$$\mathbb{E}(\Gamma_c) = \gamma_c = \sum_{k \in K} \gamma_k p(k|c)$$

Pour en revenir à la fonction objectif f, l'approche la plus simple est de s'intéresser uniquement à l'espérance du gain du programme, on prendrait alors comme fonction objectif la somme de l'espérance des gains des différentes courses du programme.

$$f(x) = \sum_{c \in x} \gamma_c$$

3.3.7 Estimation de la fonction objectif

La fonction f telle que définie ci-dessus nous est inaccessible. En effet, nous ne connaissons pas les probabilités $p(k|c)$. Ce sont les seules inconnues dont f dépend. Le but de notre classifieur est d'estimer ces probabilités. On note $\tilde{p}(k|c)$ les probabilités produites par le classifieur, et on note $\tilde{f}$ la fonction objectif estimée grâce à ces probabilités.

Si le classifieur était parfait, c'est-à-dire $p(k|c) = \tilde{p}(k|c)$, on pourrait estimer parfaitement f avec

$$f(x) = \tilde{f}(x) = \sum_{c \in x} \sum_{k \in K} \gamma_k \tilde{p}(k|c)$$

Dans le cas où le classifieur n'est pas parfait, i.e. en pratique, on a deux principales pistes, pouvant être utilisées conjointement : l'ajustement des probabilités et l'ajustement des gains.

3.3.7.1 Ajuster les probabilités

Le rôle d'un classifieur n'est pas, à l'origine, de produire des probabilités mais de déterminer la classe la plus probable pour un échantillon. La plupart du temps, la variable que l'on veut prédire n'est même pas une variable aléatoire. Par exemple, pour un classifieur devant décider si un visage est présent ou non sur une image. Il ne s'agit pas d'une variable aléatoire, ou alors de façon dégénérée les probabilités valent vraiment 0 ou 1. Ainsi, lorsqu'un tel classifieur annonce 75% de chance qu'il y ait un visage, cela ne reflète pas la probabilité que l'image contienne un visage (elle en contient une ou non), mais exprime et quantifie l'incertitude du classifieur vis-à-vis de sa réponse. Si les probabilités

sont bonnes on s'attend donc à ce que lorsque le classifieur annonce 75%, il y ait effectivement 75% du temps un visage. On dit alors que le classifieur est bien calibré.

Si, pour les courses, le classifieur est bien calibré, le mieux est d'utiliser les probabilités directement, comme dans le cas où le classifieur est parfait. Cependant, la plupart des classifieurs, et particulièrement les classifieurs produits par agrégation de classifieurs faibles, comme ceux obtenus par gradient boosting, sont mal calibrés. En effet, la probabilité retournée est le résultat du vote. Même si une course est assez facile à classifier, comme une course support d'événement, il est inévitable que quelques-uns des classifieurs faibles se trompent. Ainsi, si le classifieur annonce 95%, on peut s'attendre à être dans la bonne classe quasiment 100% du temps.

La marche à suivre est alors d'appliquer une procédure de calibration, en prenant pour référence des données de validation croisée. Il s'agit de trouver une transformation à appliquer sur les probabilités pour qu'elles correspondent au mieux aux résultats obtenus. Souvent, la transformation appliquée est une simple sigmoïde. Scikit-learn fournit des outils pour effectuer cette calibration.

En pratique la calibration n'a produit aucun résultat satisfaisant. Elle était inutile, voire contre-productive. Encore une fois, l'explication pourrait être la différence entre les données d'apprentissage et de test (2016 vs. 2017). En effet, si différence il y a, il peut être bénéfique de conserver une prudence excessive, et la calibration pourrait en fait induire une forme de sur-apprentissage.

3.3.7.2 Ajuster les gains

Pour étudier la qualité de la fonction objectif estimée $\tilde{f}$, on la compare avec la meilleure estimation que l'on ait de f, c'est-à-dire le gain observé empiriquement sur les données de test que l'on note $\hat{f}$. Le critère primaire est l'écart relatif entre $\tilde{f}$ et $\hat{f}$. Autrement dit,

$$m(x) = \frac{\hat{f}(x) - \tilde{f}(x)}{\hat{f}(x)}$$

On veut cet écart relatif aussi faible que possible. Un second critère peut être l'écart type entre le gain d'une course et son gain estimé $\sum_{k \in K} \gamma_k \tilde{p}(k|c)$, on note cet écart type σ. On veut également qu'il soit aussi faible que possible.

Une approche pour trouver une bonne fonction $\tilde{f}$ est de déterminer de nouveaux coefficients $\tilde{\gamma_k}$, tels que la fonction

$$\tilde{f}(x) = \sum_{c \in x} \sum_{k \in K} \tilde{\gamma_k} \tilde{p}(k|c)$$

	Résultats sans correction	Résultats avec correction
Gains	$\gamma = (0; 10; 20)$	$\tilde{\gamma} = (-1,85; 10,0; 21,0)$
Écart relatif	$m = 0,97\%$	$\tilde{m} = 0,57\%$
Écart type	$\sigma = 5,31$	$\tilde{\sigma} = 4,92$
Gains	$\gamma = (9; 10; 20)$	$\tilde{\gamma} = (8,6; 9,6; 21,0)$
Écart relatif	$m = 4,64\%$	$\tilde{m} = 0,20\%$
Écart type	$\sigma = 4,02$	$\tilde{\sigma} = 3,58$
Gains	$\gamma = (0; 10; 11)$	$\tilde{\gamma} = (-1,6; 10,4; 11,2)$
Écart relatif	$m = -5,3\%$	$\tilde{m} = 0,55\%$
Écart type	$\sigma = 3,42$	$\tilde{\sigma} = 3,11$

TABLE 3.7 – Efficacité des nouveaux gains sur mars et avril 2017

approche au mieux la fonction $\hat{f}$. Nous allons utiliser une régression linéaire, que nous allons calculer à l'aide de la méthode des moindres carrés. Il existe d'autres algorithmes de calcul de régression linéaire ayant pour but principalement de réduire l'impact des points abhérents comme Ridge [13] ou Lasso [14]. Elles ont été testées mais n'ont pas été plus intéressantes que les moindres carrés. Le tableau 3.7 compare les résultats utilisant γ, comme si le classifieur était parfait, et les résultats utilisant $\tilde{\gamma}$, obtenu par la méthode classique des moindres carrés.

Rappelons la démarche. Supposons qu'on connaisse combien chacune des trois classes (<8, [8 ;13], >13) est censée rapporter, par exemple $\gamma = (0; 10; 20)$. On connaît donc combien chacune des courses a rapporté. Avec une régression linéaire, on détermine de nouveaux gains $\tilde{\gamma}$, de telle sorte que la prédiction de gains pour chaque course soit aussi proche que possible de la réalité, pour mars à novembre 2016. On regroupe les résultats en terme d'écart relatif et d'écart-type sur mars et avril 2017 dans le tableau 3.7. Chaque ligne correspond à un jeu de poids théoriques γ testé. Dans la première ligne par exemple, nous avons supposé que $\gamma = (0; 10; 20)$. On lit alors que l'écart relatif entre le gain réel du programme de 2017 et le gain prédit est de 0,97%. En utilisant les coefficients modifiés présentés dans la seconde colonne, on obtient un écart relatif entre la réalité et la prédiction de 0,57%. On observe que la méthode d'apprentissage des nouveaux gains $\tilde{\gamma}$ est efficace. Les écarts relatifs sont très faibles, et l'écart type en profite également. Ce résultat est significatif car l'apprentissage de $\tilde{\gamma}$ a été fait sur les données d'apprentissage de 2016, on aurait donc pu s'attendre à des résultats moyens, mais il n'en est rien. Si on fait l'apprentissage de $\tilde{\gamma}$ sur les données de 2017, l'écart relatif est rigoureusement nul, mais on arrive pas à aller beaucoup plus bas en ce qui concerne l'écart type.

	Résultats avec Régression	Résultats avec Classifieur + Rég.Lin.
Gains	$\gamma = (0; 10; 20)$	$\tilde{\gamma} = (-1,85; 10,0; 21,0)$
Écart relatif	$m = 1,72\%$	$\tilde{m} = 0,57\%$
Écart type	$\sigma = 4,76$	$\tilde{\sigma} = 4,92$
Gains	$\gamma = (9; 10; 20)$	$\tilde{\gamma} = (8,6; 9,6; 21,0)$
Écart relatif	$m = 0,72\%$	$\tilde{m} = 0,20\%$
Écart type	$\sigma = 3,51$	$\tilde{\sigma} = 3,58$
Gains	$\gamma = (0; 10; 11)$	$\tilde{\gamma} = (-1,6; 10,4; 11,2)$
Écart relatif	$m = 1,89\%$	$\tilde{m} = 0,55\%$
Écart type	$\sigma = 3,02$	$\tilde{\sigma} = 3,11$

TABLE 3.8 – Comparaison entre Régression Gradient Boosting et classifieur sur mars et avril 2017

3.3.8 Gradient boosting regressor

3.3.8.1 Résultats numériques

Parlons enfin de l'approche régression, au lieu d'utiliser un classifieur. Ici, on essaie d'apprendre directement une méthode pour prévoir le gain d'une course, plutôt que d'essayer de la classifier. L'avantage principal est que le modèle prendra en compte les coûts γ dans son apprentissage, à l'inverse de la classification. En effet, avec le classifieur, on obtenait les mêmes probabilités quels que soient les coûts associés aux différentes classes.

Pour effectuer cette régression, nous nous proposons d'utiliser encore une fois une méthode de gradient boosting. La différence est que l'on étiquettera les feuilles des arbres de décision avec des gains, plutôt que de les étiqueter avec des classes. Ainsi, lors de l'agrégation des arbres, on prend une moyenne, éventuellement pondérée, au lieu de faire un vote. Par conséquent, le régresseur nous donne des valeurs réelles (ou décimales en pratique) correspondant directement à une estimation du gain de la course. Il n'est plus question du tout de probabilités, on utilise le résultat tel quel.

Dans le tableau 3.8, on compare les résultats obtenus avec le régresseur gradient boosting, et ceux obtenus précédemment avec l'association du classifieur gradient boosting et de la correction des gains par régression linéaire.

On observe que la prédiction du régresseur gradient booster produit des résultats avec des écarts relatifs plus de trois fois plus importants. La variance est cependant plus faible.

Remarque : on note que l'écart relatif est systématiquement positif dans les deux méthodes, autrement dit, on a tendance à sous-estimer le gain. Cela confirme en un sens le fait que le changement

de nomenclature a été bénéfique.

3.4 Évaluation d'un programme à l'aide d'une simulation

Comme expliqué dans la section 3.3.1, nous n'avons pas réussi à prendre en compte la structure d'un programme de courses dans les méthodes d'apprentissage présentées ci-dessus. Ces méthodes sont plutôt précises dans leurs prédictions compte tenu des caractéristiques exploitées, et de la variabilité du nombre de partants aux courses. Lors de la phase d'apprentissage, nous avons pris en compte le jour de la semaine et le mois, mais cela reste relativement peu utile pendant la phase de planification. En effet, on ne choisit pas dans quel mois on planifie les courses, et on ne peut pas planifier les courses qui attirent le moins de partants dans les jours de la semaine qui en attirent le plus tout en ayant des réunions hétérogènes (ce que la contrainte HR impose notamment). En somme, les méthodes de classification présentées apportent principalement une analyse fine des types de courses qui attirent le plus de partants, alors que ce qui serait utile pour guider la planification serait de savoir comme planifier chaque type de course pour qu'il attire un maximum de partants. Nous voudrions plutôt, par exemple, une méthode d'estimation qui soit capable de comprendre que des courses proches se concurrencent, et dans quelle mesure elles sont en compétition les unes avec les autres en fonction de leur type. Un telle méthode apporterait une information utile pour la planification.

Ici, on propose une méthode d'évaluation de la qualité d'un programme basée sur une simulation de la participation des chevaux.

3.4.1 Algorithme

On dispose d'un programme de courses, dont on souhaite évaluer la qualité. Au-delà du respect des contraintes métier, on peut estimer la qualité du programme a posteriori en mesurant le nombre de chevaux ayant participé à chacune des courses du programme. Dans cette section, nous proposons une méthode pour estimer au préalable le nombre de partants de chaque course d'un programme de courses.

Pour chaque course $c \in P$ du programme, on note $y_c \in \mathbb{N}$ le nombre de partants de la course c, et y le vecteur formé des y_c. L'idée est de développer une procédure spécifique permettant de simuler le parcours des chevaux dans le programme P pour estimer le vecteur y. Ainsi, nous allons simuler le

parcours de N chevaux, successivement. On considère que chaque cheval a un profil précis. On considère qu'à chaque type de course $t \in T$, correspond le profil de chevaux dont t est le type préférentiel. On définit des coefficients $\pi_{tt'}$ indiquant la propension pour un cheval de profil t à participer à une course de type t'. On définit des coefficients ω_t reflétant la proportion de chevaux de profil t dans la population de chevaux à l'entraînement. Ces deux familles de coefficients prennent dans notre implémentation des valeurs entières entre 0 et 1000. Seules leurs valeurs relatives sont importantes et on aurait pu les normaliser pour obtenir une probabilité dans $[0;1]$: une probabilité pour un type t donné d'effectuer une transition vers le type t' pour les $\pi_{t \cdot t'}$, et une probabilité pour un cheval d'être du type t pour les ω_t.

Pour modéliser la fréquence de participation des chevaux, nous avons observé expérimentalement qu'elle s'approche d'une loi bêta. On note α et β les paramètres de cette loi, et $D : [0;1] \mapsto \mathbb{R}^+$ sa fonction de densité définie comme suit, avec $D(x) = \frac{x^\alpha (1-x)^\beta}{B(\alpha,\beta)}$.
La figure 3.3 illustre le choix d'approcher par une loi bêta la fréquence des participations des chevaux. Pour obtenir ce graphique, nous avons regardé toutes les participations effectuées entre 2010 et 2017, et nous avons enregistré pour chaque paire de participations consécutives d'un même cheval, l'écart en jours qui les sépare. La courbe orange est un histogramme représentant le nombre de fois (axe des ordonnées) qu'une paire de courses est séparée de n jours (axes des abscisses). En bleu, nous avons tracé la fonction de répartition d'une loi bêta, de paramètres $\alpha = 2,5$ et $\beta = 7,5$. Nous avons appliqué une transformation affine détaillée dans le pseudo-code ci-après sur les valeurs d'entrées pour faire correspondre la courbe aux mesures. On remarque cependant que les valeurs associées à la queue de la distribution sont beaucoup plus importantes en pratique que dans la modélisation. La queue de la distribution correspond essentiellement à des chevaux qui vont participer sporadiquement une fois voire deux fois dans les trois mois planifiés. Leur participation est donc intrinsèquement imprévisible, et surtout peu dépendante de la structure du programme. Si ces participations imprévisibles touchent plus certains types de courses que d'autres, on pourra en tenir compte en ajustant les ω_t.

Notre algorithme de simulation simule successivement le parcours des N chevaux en utilisant la loi bêta et les paramètres définis ci-dessus comme suit. Pour chaque cheval, on commence par tirer au sort le type t du cheval suivant les coefficients ω_t, de sorte que la probabilité de tirer le type t soit $\frac{\omega_t}{\sum_{k \in T} \omega_k}$. On tire ensuite au sort uniformément parmi les 30 jours précédant le programme (ou le préprogramme) une date, qui sera la date (fictive) de la première participation du cheval. Les participations suivantes

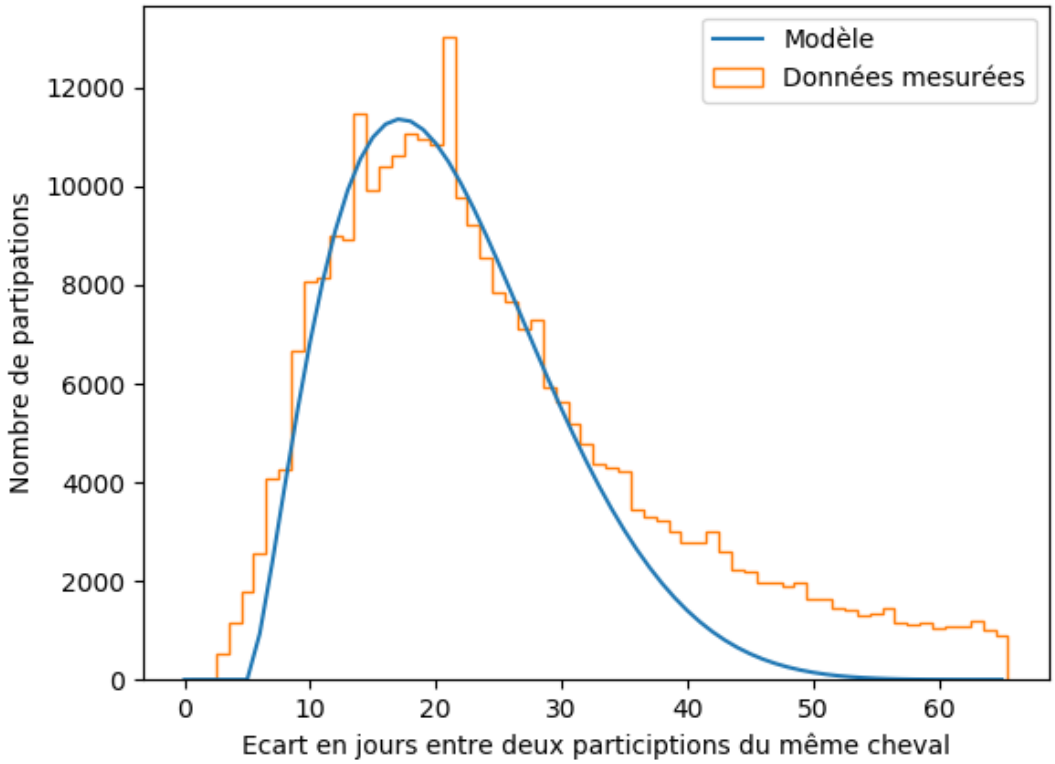


FIGURE 3.3 – Répartition des écarts en jours entre deux participations d'un même cheval comparée à la fonction de répartition d'une loi bêta avec $\alpha = 2,5$ et $\beta = 7,5$

sont ensuite tirées au sort en prenant en compte les coefficients $\pi_{tt'}$, qui traduisent la probabilité d'un cheval de profil t à participer aux différents types de courses, et de la densité de probabilité D, qui modélise la fréquence de participation des chevaux. En sommant les participations simulées de chaque cheval, on obtient pour chaque course un nombre de partants. Le pseudo code détaillant le calcul est le suivant :

Algorithme 1 : Algorithme de Simulation

```
Pour tout c, y_c = 0;
pour 1 ≤ i ≤ N faire
    Tirer au sort le profil t du cheval i suivant les ω_t;
    Tirer au sort d la date de dernière participation du cheval uniformément parmi les 30
     jours précédant le programme.;
    tant que d n'est pas une date proche de la fin du programme (voir ci-après) faire
        pour c ∈ P faire
            Soit t', le type de c.;
            Soit j, l'écart en jours entre d et la date de c.;
            x = (j − 5)/60;
            si x ∉ [0; 1] alors
                p_c = 0;
            sinon
                p_c = D(x) · π_{tt'};
        Tirer au sort c la prochaine participation du cheval i, suivant les p_c.;
        d = date de c.;
        y_c = y_c + 1;
```

La valeur x calculée est une valeur entre 0 et 1 indiquant où se situe le délai j parmi l'intervalle $[5; 65]$. Les bornes inférieure et supérieure de l'intervalle représentent respectivement le délai minimal en jours entre deux participations du même cheval et le délai maximal, vis-à-vis de la modélisation des fréquences de participation par la loi bêta. Si x est en dehors de l'intervalle, la probabilité p_c de participer à la course est nulle. Pour définir le critère d'arrêt, on ajoute une extension au programme de 30 jours de réunions fictives, après l'horizon de planification initial. Ces réunions sont remplies de courses à l'aide de l'heuristique présentée en section 5.1. La simulation pour un cheval i se termine lorsqu'il participe à une course de cette extension, c'est-à-dire que la simulation pour le cheval i est terminée en ce qui concerne l'horizon de planification initial. Sans cette extension et ce critère d'arrêt, on observe d'importants effets de bord et les courses de fin de programme ont beaucoup trop de partants.

Cet algorithme nous donne en sortie une valeur y_c, pour chaque course c du programme, c'est un nombre de partants prévu par la simulation. On peut également obtenir quel cheval a participé à quelles courses, et donc avoir son parcours. On peut donc vérifier dans une certaine mesure si l'algorithme produit ou non des parcours cohérents. Cet aspect est notamment intéressant pour convaincre les décisionnaires de la qualité des programmes produits par l'heuristique.

3.4.2 Réglage des paramètres

Il s'agit désormais de trouver des valeurs pour les paramètres ω, α, β et π, telles que la prédiction soit de qualité, c'est-à-dire corresponde à la réalité. Si on considère un programme ou un ensemble de programmes $\mathcal{P}$, on peut pour un jeu de paramètres donné obtenir à l'aide de l'algorithme un vecteur $\hat{y}$ des nombres de partants prédits par la simulation. Si on définit une fonction de coût $L(\hat{y}, y)$ qui mesure l'écart entre la participation simulée $\hat{y}$ et la participation réelle y, on peut alors chercher un jeu de paramètres qui minimise $L(\hat{y}, y)$ (en espérance par exemple, comme le calcul de $\hat{y}$ et donc de L n'est pas déterministe).

L'algorithme que nous avons utilisé pour déterminer des valeurs adaptées pour les paramètres est l'algorithme SPSA [15] (Simultaneous Perturbation Stochastic Approximation). Nous présentons tout d'abord la version standard de l'algorithme et ses caractéristiques, avant de détailler notre implémentation et les résultats obtenus.

3.4.2.1 Algorithme SPSA

On note u_n le vecteur des valeurs des paramètres à l'itération n de l'algorithme, et J la fonction à optimiser. On se donne deux suites de réels positifs (a_n) et (c_n). L'algorithme est le suivant :

Algorithme 2 : SPSA

pour $n \in \mathbb{N}$ **faire**
- Tirer au sort Δ_n un vecteur de la même taille que u_n, constitué de -1 et de 1, suivant la loi de Rademacher (pour chaque composante, les valeurs 1 et -1 sont équiprobables);
- Calculer une estimation de la i-ème composante du gradient de J en u_n, $\hat{g}(u_n)_i = \frac{J(u_n + c_n\Delta_n) - J(u_n - c_n\Delta_n)}{2c_n(\Delta_n)i}$;
- Mettre à jour $u_{n+1} = u_n + a_n\hat{g}(u_n)$;

L'algorithme SPSA est donc une descente de gradient, dont l'idée centrale est d'estimer le gradient

de J à l'aide de seulement deux évaluations de J, quelle que soit la taille du vecteur u, en perturbant dans une direction aléatoire chacune des composantes. Cet algorithme est donc particulièrement avantageux lorsque le nombre de paramètres est important et que la fonction objective est coûteuse à évaluer. Il a notamment longtemps été l'algorithme de choix pour régler les milliers de paramètres des meilleurs algorithmes jouant aux échecs, un problème dans lequel l'évaluation de la fonction objectif (le taux de victoire) est extrêmement coûteuse (jouer une partie d'échecs), et extrêmement bruitée (l'issue ne peut être que défaite, gain, ou nulle). Il a aussi été appliqué avec succès à d'autres jeux aux problématiques similaires [16].

Le paramètre c_n détermine l'amplitude des perturbations. Si c_n est trop petit, l'impact de la perturbation des coefficients sur la mesure de J est noyée dans le bruit de la mesure de J, et l'estimation du gradient est mauvaise. Si c_n est trop grand, alors on n'estime plus le gradient. Le choix de c_n dépend donc de la sensibilité de J à la variation des paramètres, et du bruit dans la mesure de J.

Le paramètre a_n détermine l'amplitude de la mise à jour de u_n en fonction du gradient estimé. Idéalement a_n doit donc être de l'ordre de grandeur de la distance entre u_n et le vecteur des valeurs optimales des paramètres à régler.

La convergence théorique et pratique a été étudiée et débouche sur des recommandations détaillées dans [17]. On y trouve :

— $c_n = c_1/n^\gamma$ avec γ =0,101.

— $a_n = a_1(1+A)^\alpha/(n+A)^\alpha$ avec α =0,601, et A une constante.

Même en suivant ces recommandations, les choix de a_1, c_1 et A restent critiques et difficiles.

3.4.2.2 Implémentation de l'algorithme SPSA et résultats

Pour réduire le nombre de paramètres à régler, nous avons adopté plusieurs hypothèses simplificatrices.

Nous considérons qu'un cheval ne change pas d'âge, et que les changements de distances et de catégories sont indépendants. Nous allons donc, pour les $\pi_{tt'}$, seulement chercher à fixer des probabilités de transition de catégories, et des probabilités de changement de distances. Nous avons donc un paramètre par paire de distances et par paire de catégories. Nous avons 18 catégories et 5 distances, soit un total de 349 paramètres π.

Les paramètres α et β ont été fixés empiriquement (aux valeurs utilisées dans la figure 3.3). Les paramètres ω ont également été fixés de façon empirique comme suit. La valeur de ω_t est le nombre de participations attendues dans le type t, c'est-à-dire la somme des y_c pour c de type t.

Il reste alors le nombre de chevaux simulés N à régler, soit un total de 350 paramètres.

L'initialisation des valeurs de u_1 dans l'algorithme SPSA a été effectuée comme suit. Nous avons vu que nous pouvions normaliser les $\pi_{tt'}$ pour obtenir des probabilités. La valeur des coefficients de transition n'est donc importante qu'à une constante multiplicative près. Il est donc nécessaire de fixer des valeurs de référence. Nous fixons à 500 les valeurs des 2 coefficients suivants : celui correspondant à la transition d'une catégorie vers elle-même et celui associé à la transition d'une distance vers elle-même. Ces valeurs ne seront pas modifiées au cours de l'algorithme. Les autres paramètres π ont une valeur initiale arbitraire de 200.
Le nombre de chevaux N a, quant à lui, été initialisé à 1300. Cette valeur a été choisie de sorte que l'application de la simulation avec ces paramètres initiaux donne le même nombre de partants en moyenne aux courses qu'il y en a eu en réalité.
Pour terminer, nous avons choisi comme programme d'entraînement le programme le plus récent dont nous disposons, le programme d'automne 2019.

En ce qui concerne la fonction objectif (J) utilisée, nous avons considéré deux fonction différentes. Premièrement, nous avons essayé de minimiser l'erreur quadratique moyenne sur le nombre de partants prédit. En notant q le nombre de courses du programme, cela correspond à $L(\hat{y}, y) = ||\hat{y} - y||_2^2/q$. Nous notons cette fonction objectif J_{part}. Nous avons également essayé de minimiser l'erreur quadratique moyenne de classification, comme nous l'avons fait pour les autres méthodes d'apprentissage. Nous avons donc séparé les courses en trois classes : moins de 8 partants, entre 8 et 13 partants, plus de 13 partants. Les composantes des vecteurs y et $\hat{y}$ prennent alors les valeurs 0, 1 ou 2 suivant la classe de la course, et l'on minimise ici aussi l'erreur quadratique moyenne. Nous appelons cette fonction objectif J_{class}. Par exemple, si la simulation prédit 10 partants pour une course, et que la course a en réalité accueilli 13 partants, J_{part} va mesurer un écart de 3 partants, soit une distance quadratique de 9. Pour J_{class}, la bonne classe a été prédite (entre 8 et 13 partants) et la distance est nulle. La valeur de J_{part} ou de J_{class} pour une simulation, est la moyenne des distances obtenues pour chaque course.

Nous avons appliqué l'algorithme SPSA deux fois, une fois pour chaque fonction objectif, et ainsi obtenu deux jeux de paramètres, que l'on note u_{part} et u_{class}.

Objectif	J_{part}		J_{class}	
	Moyenne	Ecart Type	Moyenne	Ecart Type
Paramètres				
u_1	31,4	0,993	0,885	0,0327
u_{part}	27,8	1,72	0,873	0,0284
u_{class}	60,45	3,66	0,643	0,0136

TABLE 3.9 – Récapitulatif des performances en simulation pour les différents jeux de paramètres, pour 10 simulations

Dans chaque cas, 20000 itérations de l'algorithme SPSA ont été effectuées, c'est-à-dire que l'algorithme s'est arrêté après le calcul de u_{20000}.

Pour les paramètres de l'algorithme SPSA, nous avons utilisé les valeurs recommandées pour α et γ, à savoir α =0,601 et γ =0,101. Nous avons choisi $c_1 = 25$, l'amplitude des perturbations variera donc de 25 à environ 10 au cours de l'algorithme, ce qui nous a semblé suffisant sans être excessif pour que les perturbations aient un impact mesurable. Nous avons choisi $A = 1000$, ce qui nous semblait ici aussi raisonnable, la valeur de a_n sera environ divisée par 6 entre le début et la fin de l'algorithme. Le choix de a_1 dépend de la fonction objectif. On souhaite que les amplitudes des mises à jour de u_n soient comparables entre les deux exécutions, sachant que les amplitudes des gradients des 2 fonctions sont différentes. J_{part} prend typiquement des valeurs autour de 40 alors que J_{class} prend typiquement des valeurs autour de 0,8, soit un rapport de 50. Pour J_{part}, nous avons donc choisi $a_1 = 1$, et pour J_{class}, nous avons choisi $a_1 = 50$.

Résultats Dans la table 3.9, nous avons récapitulé les résultats obtenus. Pour chaque jeu de paramètres, nous avons effectué 10 simulations sur le programme d'automne 2019, et nous avons mesuré les valeurs moyennes et les écarts types obtenus pour J_{part} et J_{class}. Sur les lignes, nous avons le jeu de paramètres testé. Dans l'ordre, le jeu de paramètres inital, ayant servi de point de départ au SPSA, le jeu de paramètres entraîné sur l'erreur quadratique en nombre de partants, et le jeu de paramètres entraîné sur l'erreur quadratique de classification.

On remarque que pour les deux fonctions objectifs à minimiser, l'algorithme SPSA a réussi à produire des valeurs pour les paramètres plus performantes en moyenne que les paramètres initiaux. Pour J_{part}, les simulations effectuées avec les paramètres initiaux u_1 donnent en moyenne 31,4, alors que les paramètres u_{part} donnent en moyenne 27,8. Pour J_{class}, les paramètres u_1 donnent 0,885 en

moyenne alors que les paramètres u_{class} donnent en moyenne 0,643.

Au delà des purs résultats, il est intéressant d'examiner les valeurs obtenues pour les paramètres, et leur évolution au cours de l'algorithme.

Pour la minimisation de J_{class}, on observe globalement que les transitions des catégories des types de courses peu attractifs vers les types de courses attractifs ont une probabilité élevée, et que les transitions inverses ont une probabilité faible. On observe donc logiquement dans les résultats de simulation, que les courses à Handicap ont beaucoup de partants (plus que dans la réalité), alors que les courses C1, par exemple, en ont très peu (moins que dans la réalité). En somme, comme pour les méthodes d'apprentissage présentées précédemment, nous avons pu reconnaître les types de courses attractifs et les types de courses peu attractifs, et la simulation donne simplement plus de chance aux chevaux simulés de participer aux types de courses pour lesquels on anticipe plus de partants.

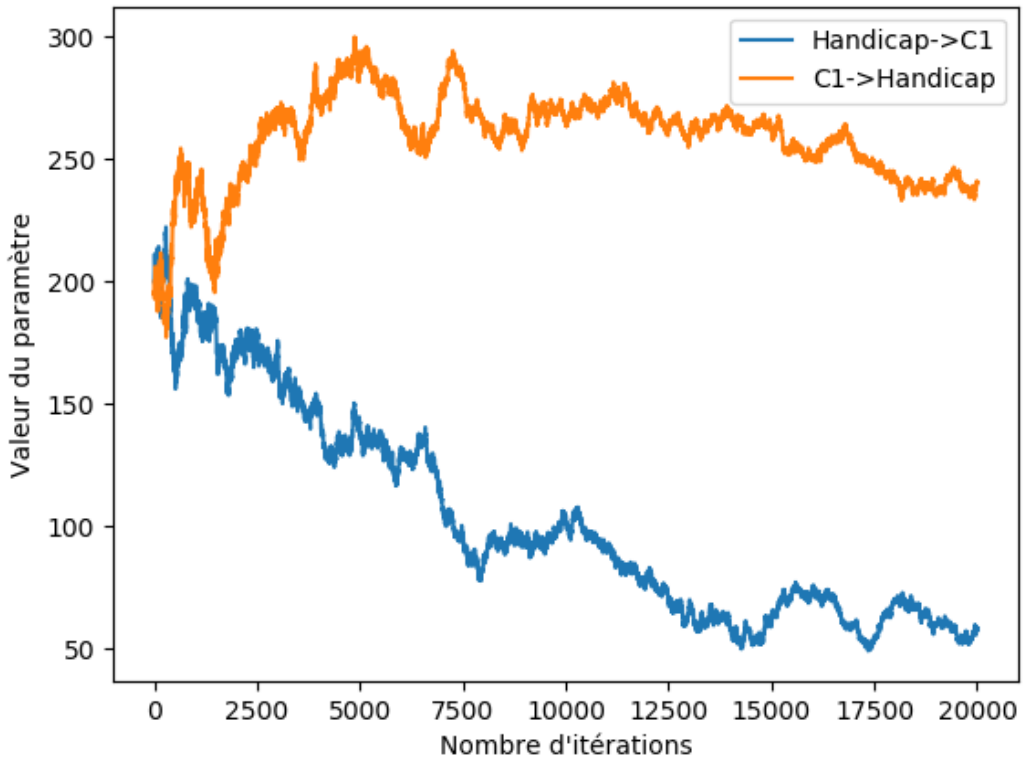


FIGURE 3.4 – Évolution de paramètres de transition lors de la minimisation de J_{class}

Pour illustrer notre propos, nous avons tracé, dans la figure 3.4, l'évolution des paramètres associés

aux transitions C1→Handicap et Handicap→C1, lors de l'exécution de l'algorithme. On observe bien que la probabilité pour un cheval d'un type Handicap de participer à une course de type C1 tend à diminuer, alors que la probabilité inverse reste élevée.

Si l'on s'intéresse à l'apprentissage pour minimiser J_{part}, on n'observe cette fois-ci pas vraiment de motif clair dans les coefficients de transitions. Une analyse plus fine des simulations produites met cependant en évidence plusieurs défauts de modélisation. On observe par exemple que la simulation prédit une quantité anormalement élevée de partants pour les courses réservées aux chevaux de 3 ans et plus. Les deux raisons faisant que cela se produit sont assez simples. Tout d'abord, nous avons supposé qu'un cheval ne participe qu'aux courses de la tranche d'âge qui lui est affectée, ce qui n'est pas la réalité puisqu'un cheval peut très bien participer à des courses « 3 ans et plus », et à des courses « 3ans », si il a trois ans par exemple. Ce n'est pas en général un problème majeur, sauf que les courses « 3 ans et plus » sont rares et planifiées exceptionnellement. Les chevaux simulés de la tranche d'âge « 3 ans et plus » ne peuvent donc pas courir autant que les autres chevaux, et sont donc, comme le nombre de chevaux de chaque âge est fixé empiriquement en fonction du nombre de participations dans cet âge-là, en nombre bien trop important. Il y a peu de courses pour chevaux de trois ans et plus, mais comme l'erreur quadratique est évaluée, elles comptent pour beaucoup dans le score global (on a typiquement 45 partants prédits au lieu des 18 attendus). En conséquence, un moyen efficace que l'algorithme SPSA a pour réduire l'erreur quadratique est de réduire l'erreur sur ces courses-là en diminuant la valeur du paramètre N qui contrôle le nombre de chevaux simulés, ce qu'on observe en figure 3.5. On observe alors que la simulation prédit environ 10 partants en moyenne aux courses, alors qu'il y a en réalité 11,2 partants en moyenne, ce qui n'est évidemment pas optimal.

3.4.3 Conclusion et perspectives sur la simulation

L'étude de l'approche par simulation présentée dans cette partie avait pour but de répondre à deux questions. D'une part, est-il possible de mettre au point une méthode prédictive prenant en compte la structure du programme, et donc potentiellement utilisable pour guider la planification ? D'autre part, est-il possible de trouver des paramètres produisant des simulations proches de la réalité pour un modèle de simulation donné ?

Pour cette seconde question, les premiers résultats obtenus avec l'algorithme SPSA sont plutôt encourageants et semblent répondre positivement. Sans trop de difficulté concernant l'implémentation

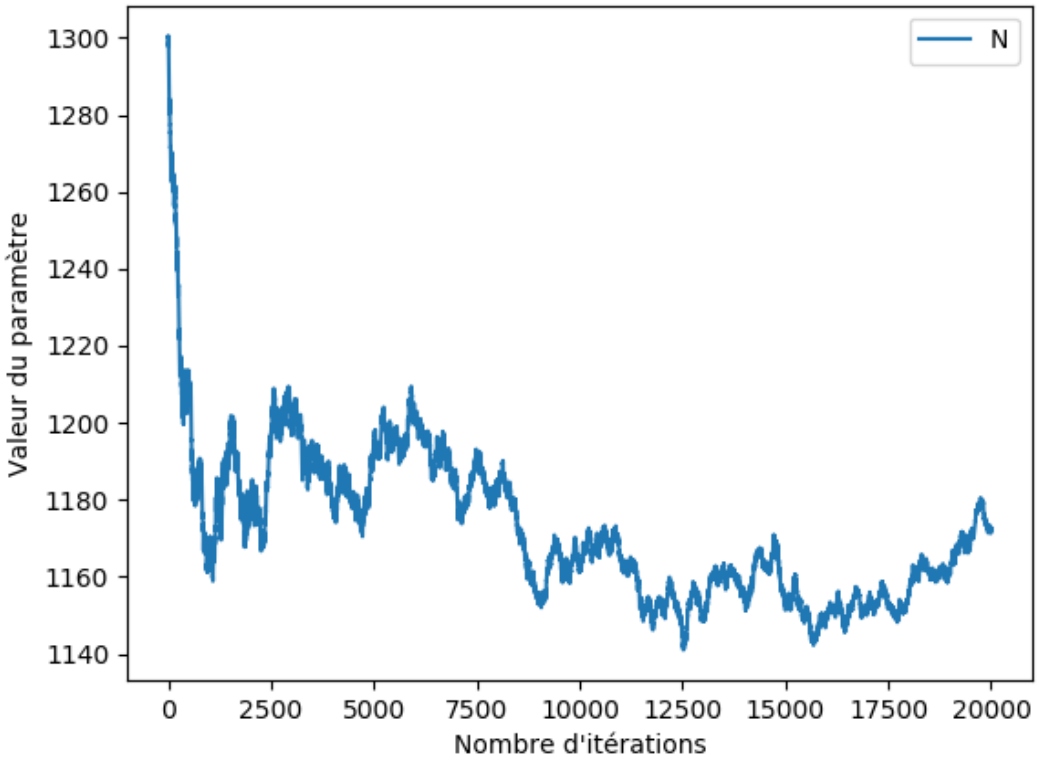


FIGURE 3.5 – Évolution du nombre de chevaux simulés lors de la minimisation de J_{part}

ou le choix des paramètres, nous avons pu obtenir des valeurs pour les 350 paramètres optimisés bien plus performantes en moyenne que les paramètres initiaux pour deux fonctions objectif différentes, et ce en seulement 40000 simulations. Il est tout à fait possible qu'en étudiant mieux le choix des paramètres de l'algorithme SPSA et en faisant éventuellement plus d'itérations, on puisse obtenir des valeurs encore meilleures.

Un tel travail n'est cependant intéressant qu'après avoir exploré le plus de pistes possibles pour améliorer le modèle, l'algorithme de simulation. Il est important de garder un nombre raisonnable de paramètres à optimiser, l'hypothèse d'indépendance des transitions de distance et de catégorie semble donc indispensable, mais il existe d'autres pistes d'amélioration. Tout d'abord, il serait judicieux d'affecter aux chevaux simulés non pas une tranche d'âge mais un âge (pouvant appartenir à plusieurs tranche d'âge). Un cheval de trois ans pourrait ainsi par exemple participer aux courses 3 ans, mais aussi aux courses 3 ans et plus. Pour mieux modéliser la réalité, il semble aussi important de permettre

à un cheval simulé de changer de type. On voudrait par exemple qu'un cheval d'un type de catégorie Inédits (course pour les chevaux n'ayant jamais couru) participe nécessairement à une course d'Inédits, puis change de type, par exemple devienne un cheval qui puisse courir une course Maiden (réservée aux chevaux n'ayant jamais gagné). Pour que cela fonctionne, il faudrait également modéliser l'arrivée de nouveaux chevaux en cours de programme pour alimenter les courses d'Inédits.

Enfin, le plus important reste la première question : la simulation prend elle en compte la structure du programme ? Malheureusement les jeux de paramètres produits ne prennent pas plus en compte la structure que les méthodes d'apprentissage présentées avant. En effet, les paramètres de transition ont tous (quasiment) des valeurs non nulles. Ainsi chaque cheval simulé aura toujours, peu importe la structure du programme, de nombreuses opportunités. La structure du programme est donc presque sans importance, et ce que fait la simulation est essentiellement de partager le pool de chevaux simulés de façon intelligente entre les différents types de courses. Heureusement, une piste d'amélioration simple par rapport à ce problème consiste à fixer à 0 les coefficients de transition que l'on estime irréalistes (ce qui au passage diminuerait le nombre de paramètres à optimiser). Même en faisant cela, il est cependant possible que l'on rencontre le même problème qu'avec les autres méthodes, à savoir que l'on ne dispose pas dans les données d'exemples de mauvais programmes. On aimerait avoir des programmes avec à la fois des courses de type t qui marchent bien, et des courses de type t qui attirent peu de partants pour des raisons structurelles, pour pouvoir apprendre les structures efficaces.

En somme, il est certain que fixer beaucoup de paramètres à 0 produira des simulations dépendantes de la structure du programme, et on a toutes les raisons de penser que l'algorithme SPSA pourra trouver des paramètres efficaces avec un nouveau modèle, mais seule l'expérience dira si les coefficients obtenus déboucheront sur un comportement généralisable et réaliste sur d'autres structures de programme. Il se dégage généralement des travaux d'apprentissage que les données disponibles sont très délicates à exploiter. Il semble intéressant toutefois de poursuivre les travaux de modélisation de la simulation pour voir ce qu'il est possible de faire, et de chercher à développer des méthodes de planification qui pourront utiliser cette simulation pour guider la planification.

Chapitre 4

Méthodes exactes pour générer une solution de planification

Contenu

4.1 Introduction

Dans le chapitre 3, nous avons étudié différentes méthodes d'évaluation de la qualité d'une solution de planification, vis-à-vis du nombre de partants attendus. Nous allons maintenant présenter les méthodes étudiées pour produire des solutions, soit des programmes de courses. Le présent chapitre regroupe les méthodes exactes, et le suivant les méthodes approchées.

Dans ce chapitre, nous présentons donc les travaux réalisés autour de la modélisation du problème de la planification des courses de galop sous la forme d'un programme linéaire en variables 0-1. En plus d'examiner une résolution par séparation et évaluation de cette formulation, l'objectif est d'évaluer la difficulté théorique et pratique du problème d'optimisation étudié.

Tout d'abord, nous revenons dans la section 4.2 sur l'énoncé du problème en présentant les notations utilisées par la suite. Une première modélisation simplifiée du problème est proposée en section 4.3. Nous montrons ensuite en section 4.4 que le problème de décision associé est NP-complet. Ensuite nous proposons deux généralisations du modèle en section 4.5, qui répondent mieux aux problématiques pratiques de la planification. Nous présentons enfin des résultats numériques sur des instances réelles en section 4.6.

4.2 Définition du problème et notations

Dans toute la suite de la section, nous utiliserons les notations suivantes :

— R : un ensemble de réunions.

— n_r :la capacité de la réunion $r \in R$. Il s'agit du nombre maximum de courses pouvant être planifiées dans la réunion r.

— T : un ensemble de types de courses.

— $A \subset R \times T$: un ensemble de courses planifiables compte tenu des contraintes. $\overline{A} = (R \times T) \setminus A$

— $P_{\text{re}} \subset A$ un ensemble de courses préprogrammées. Ces courses ont été planifiées à la main par les experts, et doivent être présentes dans toute solution.

— $S(r,t) \subset R$: un sous-ensemble de courses pouvant succéder à la course $(r,t) \in A$ au sens de la contrainte d'écart. Ces courses seront qualifiées de *successeurs* dans la suite. Les successeurs sont utilisés pour définir la contrainte d'écart détaillée ci-après.

Chaque réunion r appartenant à l'ensemble R des réunions est définie par la date et l'hippodrome dans lequel elle se déroule. On doit décider des types de courses à planifier dans chaque réunion r, en prenant en compte les courses de P_{re}. Il peut y avoir au plus n_r courses dans la réunion r.

Pour chaque course, de nombreux types sont possibles. Un type de course est défini par le triplet (âge, distance, catégorie). T est l'ensemble de ces types de courses.

La seule contrainte dure présentée dans le paragraphe 1.4 qui ne peut pas être modélisée à l'aide de A est le nombre maximal de courses que peut accueillir une réunion. Cette contrainte de capacité sera modélisée algébriquement via la donnée n_r introduite dans les notations.

La donnée $S(r,t)$ sert à modéliser la contrainte d'écart (section 1.4.4.1), la contrainte souple centrale du problème. La façon dont nous allons formaliser cette contrainte est la suivante : si une course de type t est planifiée dans la réunion r, une autre course de type t doit être planifiée dans une des réunions de $S(r,t)$. De part la nature chronologique de la contrainte d'écart, nous imposons sur S l'existence d'une fonction $date$ de R sur $\mathbb{N}$ telle que $(r',t) \in S(r,t) \Rightarrow date(r') > date(r)$. En pratique, la fonction qui à chaque réunion lui associe sa date convient, d'où la dénomination.

4.3 Modèle linéaire en variables 0-1

Pour justifier l'utilisation par la suite de méthodes approchées, nous souhaitons d'abord montrer que le problème est difficile à résoudre avec des méthodes exactes d'un point de vue théorique comme pratique. Dans cette section, nous présentons une première modélisation du problème en un programme linéaire en variables 0-1 que nous appelons (P0). Nous prouvons dans la section 4.4 que le problème de décision associé à (P0) est NP-complet. La formulation est la suivante :

$$
\text{(P0)} \begin{cases}
\min \quad 0 & & \\
\text{s.t.} & & \\
\quad \sum\limits_{t \in T} x_{r,t} \leq n_r & r \in R & (1) \\
\quad x_{r,t} \leq \sum\limits_{(r',t') \in S(r,t)} x_{r',t'} & (r,t) \in A,\ S(r,t) \neq \varnothing & (2) \\
\quad x_{r,t} = 0 & (r,t) \in \overline{A} & (3) \\
\quad x_p = 1 & p \in P_{\text{re}} & (4) \\
\quad x_{r,t} \in \{0,1\} & r \in R,\ t \in T &
\end{cases}
$$

Les $x_{r,t}$ sont les variables de décision, et valent 1 si une course de type t est planifiée dans la réunion r et 0 sinon.
Les contraintes (1) limitent le nombre de courses par réunion. Les contraintes (2) sont les contraintes d'écart. Les contraintes (3) fixent les variables associées aux courses non planifiables qui composent $\bar{A}$ à 0. Les contraintes (4) fixent à 1 les variables associées aux courses de P_{re}. Il n'y pour l'instant pas de fonction d'évaluation, qui sera ajoutée en section 4.5, $(P0)$ est donc un problème de satisfaction.

4.4 NP-difficulté

On considère le problème $HRSP$: « Est-ce que $(P0)$ a une solution ? ». Il s'agit du problème de décision associé à $(P0)$.

Théorème 1. *$HRSP$ est NP-complet.*

Démonstration. Tout d'abord, $HRSP$ appartient à la classe de complexité NP. En effet, on peut vérifier en temps polynomial si une solution de planification satisfait les contraintes de $(P0)$ (le nombre de variables et de contraintes est polynomial).
Dans un second temps, nous allons réduire le problème des k-chemins à sommets disjoints dans un graphe orienté acyclique (DAG) au problème $HRSP$. Nous proposons une transformation telle qu'à toute solution du problème de k-chemins à sommets disjoints dans un DAG corresponde une solution de $(P0)$, et inversement.

Le problème de k-chemins à sommets disjoints dans un DAG est un problème de décision. Soit $G = (V, E)$ un DAG , avec $s_1, \ldots, s_k \in V$ des sources, et $P, \ldots, p_k \in V$ des puits. On veut savoir si il existe k chemins à sommets disjoints reliant les $s_1, \ldots, s_k$ aux $P, \ldots, p_k$ respectivement. Ce problème est NP-complet [18].

Si une source et un puits de V sont confondus en un même sommet noté $p_i, s_{i'}$, on peut diviser ce sommet en un sommet source $s_{i'}$ et un sommet puits p_i (Figure 4.1). Au sommet source sont incidents les arcs sortants de $p_i, s_{i'}$, et au sommet puits sont incidents les arcs entrants. Si deux puits (resp. deux sources) de V sont confondus en un sommet, on peut aussi le séparer en deux sommets, chacun gardant une copie des arcs. En effet, comme nous travaillons sur le problème de k-chemins à sommets disjoints (par opposition au problème avec arcs disjoints), la duplication des arcs résultant de la séparation du sommet ne change pas la nature du problème. Ainsi, on peut supposer, sans perte de généralité,

que les puits et les sources sont disjoints. De plus, si un sommet de $V - \{P, \ldots, p_k\}$ ne possède pas d'arc sortant, il ne fera partie d'aucune solution. On peut le supprimer. De même, on peut supprimer du graphe les sommets autres que les sources n'ayant pas d'arc entrant. On peut donc, sans perte de généralité, supposer qu'un sommet est un puits si et seulement si il n'a pas d'arc sortant, et qu'un sommet est une source si et seulement si il n'a pas d'arc entrant.

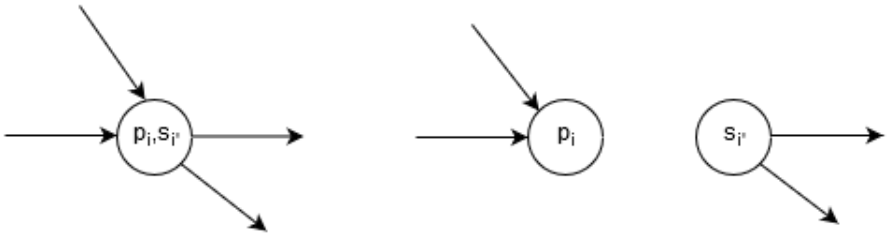


FIGURE 4.1 – Séparation d'une source et d'un puits joints en deux sommets

On donne maintenant une transformation d'une instance de k-chemins à sommets disjoints dans un DAG en une instance de $HRSP$. On montrera que l'instance de k-chemins à sommets disjoints dans un DAG a pour réponse « Oui » si et seulement si l'instance de $HRSP$ a pour réponse « Oui » également (c'est-à-dire que l'instance de ($P0$) a une solution).

Considérons la transformation suivante :

— $T = \{1, \ldots, k\}$

— $R = V$

— $S(r, t) = \{(r', t) | (r, r') \in E\}$

— $A = (R \setminus \{P, \ldots, p_k\} \times T) \cup \{(p_i, i) : i \in T\}$

— et $\overline{A} = \{(p_i, j) | j \in T, j \neq i\}$

— $P_{\text{re}} = \{(s_i, i) : i \in T\}$

— $n_r = 1$ pour tout $r \in R$.

Pour que cette transformation définisse une instance de $HRSP$, on doit vérifier qu'il existe une fonction date compatible avec S. C'est bien le cas puisque G est acyclique. On peut dès lors calculer un tri topologique des sommets de V et l'utiliser comme fonction date.

Supposons maintenant que cette instance de $HRSP$ a une solution, c'est-à-dire que l'instance de ($P0$) associée a une solution, et montrons que le problème de k-chemins à sommets disjoints dans un DAG a une solution également. Soit $I_i = \{(u, v) \in E | x_{u,i} = 1, x_{v,i} = 1\}$ $\forall i \in T$. On veut montrer que

l'union des I_i contient une solution au problème de k-chemins à sommets disjoints dans un DAG.

Montrons d'abord, pour i fixé, que I_i contient au moins un chemin de s_i à p_i. La contrainte (2) de $(P0)$ implique que si $(u, v) \in I_i$, on a un arc $(v, w) \in I_i$, si $S(v, i) \neq \varnothing$. La définition de P_{re} et les contraintes (2) et (4) de $(P0)$, impliquent qu'il existe $(s_i, u) \in I_i$, si $S(s_i, i) \neq \varnothing$, et c'est le cas puisque s_i n'est pas un puits. En effet, nous avons supposé que les puits sont les seuls sommets sans arcs sortants de V , de sorte que $S(r, i) = \varnothing \Rightarrow \exists i' \in \{1, \ldots, k\}$ tel que $r = p_{i'}$. Cependant, les seuls $(p_{i'}, i)$ dans A sont les (p_i, i), on peut ainsi en déduire que I_i contient un chemin de s_i à p_i. I_i peut contenir plus d'un tel chemin, peut-être avec des sommets en commun, ainsi que des arcs n'appartenant à aucun chemin de s_i à p_i, mais c'est sans importance.

Puisque $n_r = 1$, les contraintes (1) impliquent que chaque réunion est composée d'au plus une course. Ainsi, deux arcs appartenant respectivement à I_i et I_j, avec $i \neq j$, ne peuvent pas avoir la même extrémité terminale v' car $\forall v' \in V x_{v',i} + x_{v',j} \leq 1$. Puisque les chemins dans I_i n'ont ni la même source ni le même puits que les chemins de I_j, on peut en déduire que les chemins de I_i n'ont pas de sommet en commun avec les chemins de I_j. On obtient donc bien une solution du problème de k-chemins à sommets disjoints dans un DAG à partir d'une solution de (P_D) .

Supposons maintenant que le problème de k-chemins à sommets disjoints dans un DAG a une solution, et montrons que P_D en a une également. Soit I une solution du problème de k-chemins à sommets disjoints dans un DAG. On nomme I_i le i-ème chemin. On considère l'affectation suivante pour les variables de $HRSP$: $x_{s_i,i}$ vaut 1 pour tout $i \in [1, \ldots, k]$, et si $(u, v) \in I_i$, $x_{v,i} = 1$. Les autres variables sont mises 0. La fonction objectif est donc 0, et il ne reste plus qu'à démontrer que cette affectation respecte les contraintes. Puisque les chemins sont à sommets disjoints, ils sont arcs disjoints, et la contrainte (1) est respectée, puisque $n_r = 1$. Les contraintes (3) et (4) sont aussi satisfaites. Pour tout arc $(u, v) \in I$, où v n'est pas un puits, on a également un arc $(v, w) \in I$. Ainsi, (2) est satisfaite.

Ainsi, le problème NP-complet de k-chemins à sommets disjoints dans un DAG se réduit polynomialement au problème P_D . On peut donc conclure, P_D est NP-complet. □

4.5 Généralisations du modèle

Le modèle $(P0)$ étudié jusque là ne prend pas en compte toutes les considérations pratiques importantes liées à la planification manuelle des courses de chevaux. Ce modèle simplifié nous a, cependant,

servi pour étudier la complexité du problème de décision associé. Nous proposons, dans cette section, deux généralisations du modèle, qui reflètent mieux la réalité.

Commençons par clarifier la signification pratique de certaines des données du problème.

Les courses de P_{re}, sont des courses fixées en amont par les décideurs. Une grande partie de ces courses sont celles du *préprogramme*. En effet, dans les instances réelles, R contient les réunions des trois mois de l'horizon de planification, mais aussi les réunions du mois qui précède cette plage de trois mois, appelée préprogramme, pour permettre d'assurer la continuité entre les programmes trimestriels. Par exemple, lorsqu'on planifie les courses pour les mois de septembre à novembre, les courses du préprogramme, fixées à l'aide de P_{re}, sont les courses du mois d'août. Il est important de prendre en compte le préprogramme lors de la planification pour assurer un enchaînement réalisable entre les programmes, vis-à-vis de la contrainte d'écart.

P_{re} contient aussi d'autres courses préfixées. Certaines de ces courses sont des courses très sélectives et planifiées à la main au préalable, comme les courses de Groupe ou les courses Listed. Il peut également y avoir d'autres courses fixées au préalable, suivant l'usage que l'on veut faire de l'outil. En effet, l'outil de planification doit permettre à l'utilisateur de préfixer un sous-ensemble de courses en amont quelles que soient leurs natures avant la phase de planification automatique. On peut par exemple vouloir effectuer la planification des courses pour les chevaux de 3 ans et plus après avoir fixé les courses pour les chevaux de 2 ans. Typiquement, un scénario possible d'usage est la génération complète du programme par l'outil, suivie de l'ajustement manuel par les experts de la solution pour les chevaux de 2 ans, suivi enfin d'une deuxième passe de l'outil pour calculer à nouveau la planification des courses pour les chevaux de plus de 3 ans.

En ce qui concerne l'ensemble T, il contient dans nos instances réelles tous les triplets (âge, distance, catégorie) comme définis dans le paragraphe 1.3.

Dans ($P0$), on remarque que les contraintes dures présentées dans le paragraphe 1.4 ne figurent pas explicitement. Elles se traduisent par la fixation à 0 des variables associées aux courses de $\bar{A}$. Le sous-ensemble $A \subset R \times T$ sert à intégrer implicitement les contraintes dures en indiquant quelles sont les courses de $R \times T$ qui respectent ces contraintes. Les contraintes dures imposées par le code des courses et les spécificités de l'hippodrome sont détaillées dans le paragraphe 1.4. Voici comment il est possible de les prendre en compte à l'aide de A :

— Certains types de courses peuvent être interdits. Par exemple, les chevaux de 2 ans ne peuvent pas courir des courses de plus de 2000 m. Ainsi, tout élément $(r, t) \in R \times T$ où t désigne un type de course combinant une distance supérieure à 2000 m et une course réservée à des chevaux de deux ans peut être supprimé de A.

— On peut restreindre certains types de courses à certaines réunions. Par exemple, les courses de Sprint doivent être courues sur une ligne droite, et requièrent donc que l'hippodrome soit doté d'une ligne droite mesurant entre 800 m et 1400 m. Si t est un type de course de Sprint, et que r est une réunion se déroulant dans un hippodrome n'ayant pas de ligne droite, on peut enlever (r, t) de A. De même, certains types de courses sont saisonniers. Par exemple les chevaux de 2 ans ne courent qu'à partir de mi-mars, et ne courent plus de 1200m qu'à partir de mai. On peut ôter les combinaisons (r, t) correspondantes de A.

Finalement, un aspect important de la résolution des instances réelles est la gestion de l'effet de bord émanant de l'horizon de planification. En effet, le modèle $(P0)$ stipule qu'une course (r, t) de A n'ayant pas de successeur car elle se situe en fin de programme ($S((r, t))$ est vide) n'est pas soumise à la contrainte d'écart. Cependant il se peut qu'une course proche de la fin du programme ait peu de successeurs, ce qui forcerait le placement d'un de ces successeurs, alors qu'en pratique on pourrait très bien attendre la planification des mois suivants pour affecter un successeur à (r, t). L'effet de bord ainsi engendré par ces courses proches de la fin de l'horizon de planification peut être pallié par l'ajout de courses fictives après l'horizon de planification. Ces dites courses seront ignorées après avoir résolu le problème. Puisque le nombre de courses et de réunions ajoutées est borné, cet ajout n'a pas d'impact asymptotique sur la complexité du problème, et c'est pourquoi nous l'avons ignoré lors des considérations théoriques.

D'autres aspects pratiques de la planification ont été ignorées dans $(P0)$ et sont intégrés dans le modèle (P) suivant.

$$
(\mathrm{P}) \begin{cases}
\min \quad \sum_{(r,t)\in A} \gamma_{r,t}\lambda_{r,t} + \sum_{r\in R,\, t\in T} p_t \mu_{r,t} x_{r,t} & & \\
\text{s.t.} & & \\
\quad \sum_{t\in T} p_t x_{r,t} \leq n_r & r \in R & (1) \\
\quad x_{r,t} \leq \lambda_{r,t} + \sum_{(r',t')\in S(r,t)} x_{r',t'} & (r,t) \in A,\ S((r,t)) \neq \varnothing & (2) \\
\quad L_r \leq \sum_{t\in HR} p_t\ x_{r,t} \leq U_r & r \in R & (3) \\
\quad x_{r,t} = 0 & (r,t) \in \overline{A} & (4) \\
\quad x_p = 1 & p \in P_{\mathrm{re}} & (5) \\
\quad x_{r,t} \in \{0,1\} & r \in R,\ t \in T & \\
\quad \lambda_{(r,t)} \in \{0,1\} & (r,t) \in A &
\end{cases}
$$

Tout d'abord, le modèle (P) formalise sous forme de contraintes « dures » les deux ensembles de contraintes souples du problème suivants :

— La contrainte des Paires stipule que certains types de courses doivent être planifiés par paires. C'est le cas pour les courses à Handicap pour les chevaux de 4 ans et plus. Ces courses à Handicap attirant beaucoup de partants, elles sont planifiées deux fois dans la réunion pour faire plus de bénéfice et pour répartir les chevaux en une course de niveau supérieur et une autre de niveau inférieur. Les courses Maiden sont également planifiées par paires. L'une est pour les chevaux mâles et l'autre pour les chevaux femelles. La modélisation de cette contrainte est intégrée à la contrainte (1), à l'aide des coefficients p_t. Un type de course devant être planifié par paire prendra deux places par réunion, en prenant $p_t = 2$. p_t vaudra 1 pour les autres types de courses.

— Les experts veulent que chaque réunion contienne un mélange de courses sélectives et de courses ouvertes. On nomme l'ensemble des types de courses correspondant à des courses ouvertes $HR \subset T$, car il contient les courses à Handicap et les courses à Réclamer. La nouvelle contrainte (3) impose alors à chaque réunion de contenir entre L_r et U_r courses de HR. Comme dans (1), les courses de type t avec $p_t = 2$ comptent double. L_r et U_r sont des paramètres avec une valeur par défaut (typiquement 3 et 5 après concertation avec les planificateurs), mais peuvent être ajustées dans une phase de précalcul pour être compatibles avec n_r et le préprogramme.

Toujours à propos de la modélisation des contraintes souples, on observe que les contraintes (2) ne sont pas systématiquement satisfaites en pratique. Nous prenons en compte cette souplesse via l'ajout

de variables de pénalisation $\lambda_{r,t}$, pour que les contraintes d'écart soient traitées comme des contraintes souples. Ainsi, si la contrainte d'écart n'est pas respectée, la variable $\lambda_{r,t}$ vaut 1, ce qui aboutit à une pénalité dans la fonction objectif.

En plus de cette pénalisation, pondérée par des $\gamma_{r,t} > 0$, on affecte à la planification de chaque course un score $p_t\mu_{r,t}$, pour permettre une évaluation primitive de la qualité du programme produit. Les coefficients $\mu_{r,t}$ peuvent prendre des valeurs positives ou négatives. Ils pourraient par exemple être une estimation du nombre de partants des courses de type t, mais nous en parlerons plus en détail en section 4.6.

Le programme (P) est une généralisation du programme (P0), c'est-à-dire que toute instance de (P0) peut être transformée polynomialement en une instance de (P). Le problème de décision associé à (P) est donc également NP-complet. Pour obtenir une instance de (P) à partir d'une instance de (P0), on fixe les paramètres p_t à 1 pour que la contrainte (1) de (P1) soit la même que la contrainte (1) de (P0). On fixe les $\mu_{r,t}$ à 0 pour éliminer la seconde partie de la fonction objectif. On fixe les L_r à 0, et les U_r suffisamment élevé (le maximum des n_r par exemple) pour que la contrainte (3) de (P) soit respectée par toute affectation. En fixant les $\gamma_{r,t}$ à 1, on obtient une instance de (P) dont la fonction objectif vaut 0 si et seulement les $\lambda_{r,t}$ sont tous nuls. Il vient alors que le problème $(P0)$ a une solution si et seulement si le problème $(P1)$ associé a une solution de valeur nulle.

4.6 Limites du modèle linéaire en variables 0-1

La généralisation du modèle présentée dans la section précédente est seulement une façon de modéliser les contraintes du problème. Il est toujours possible de pénaliser différemment les contraintes, d'en ajouter, d'en modifier, pour tenter d'obtenir des solutions entières plus proches des attentes des experts. Il existe aussi de nombreux choix possibles pour les paramètres du modèle. Nous présentons dans cette section plusieurs essais que nous avons faits, et les difficultés rencontrées.

Dans un premier temps, nous avons travaillé avec le modèle (P) en utilisant le jeu de paramètres décrit dans la suite. Pour choisir la valeur des $\mu_{r,t}$, nous avons utilisé les résultats obtenus en apprentissage avec l'algorithme du gradient boosting (section 3.3.5). Pour chaque couple (r, t), nous avons un nombre de partants prédit. On utilise l'opposé de cette valeur pour $\mu_{r,t}$ (dans un contexte de minimisation). Le critère principal d'évaluation des solutions restant le respect des contraintes, on choisit

donc les $\gamma_{r,t}$ assez élevés (cinq fois plus grands que les $\mu_{r,t}$ dans nos tests). Nous avons testé ce jeu de paramètres sur l'instance de printemps 2018.

Plusieurs problèmes ont été observés dans les solutions produites :

— Aucune contrainte ne régule réellement la proportion de chaque type de course dans les solutions. La contrainte (3) a un effet régulateur, mais s'applique très largement à de nombreux types de courses à la fois. Dans les solutions entières trouvées, on observe qu'un type de course avec un $\mu_{r,t}$ faible va être rare, alors qu'un type de course avec un $\mu_{r,t}$ élevé sera beaucoup trop représenté. Il est vrai que l'on voudrait planifier plus de courses à Handicap, qui comptent beaucoup de partants, que de courses Listed ou de Groupe, qui sont très sélectives et peuvent être plus occasionnelles. On ne veut pas en revanche avoir beaucoup plus de courses Sprint que de courses Miler, par exemple, ce qui se produit pourtant même si leurs $\mu_{r,t}$ respectifs sont peu différents. Une idée que nous avons essayée pour remédier à ce problème est d'ajouter de nouvelles contraintes, mais ce n'est pas très raisonnable puisque notre problème est déjà très contraint. Si, par exemple, on borne le nombre de courses de chaque type, il y a de fortes chances que les instances n'aient plus de solution. En d'autres termes, si la contrainte est trop large, elle n'a pas d'effet ; si elle est, en revanche, trop serrée, le problème n'a pas de solution. Il faudrait au moins avoir des valeurs pour ces paramètres qui dépendent de l'instance, et nous n'avons aucun moyen de faire ça.

— Comme nous l'avons vu dans le point précédent, la fonction objectif ne représente pas correctement nos attentes, en tout cas pas sans l'ajout de contraintes plus complexes. Avec un objectif linéaire comme celui-ci, on planifie en nombre les courses prédites comme étant les plus profitables, or ce n'est pas la réalité de la planification. Pour obtenir de bonnes solutions avec un objectif linéaire, il est nécessaire d'avoir des contraintes complexes et des pénalités pour le non respect de ces contraintes. Par exemple il est possible d'introduire une nouvelle famille de contraintes et un nouveau jeu de pénalités pour ajouter de la nuance dans le respect de la contrainte d'écart. Pour modéliser le fait que l'on préfère qu'une course ait un successeur au centre de l'écart plutôt qu'au bord on peut par exemple ajouter les contraintes suivantes :

$$x_{r,t} \leq \lambda_{r,t} + \sum_{(r',t') \in S'(r,t)} x_{r',t'} \quad (r,t) \in A,\ S'((r,t)) \neq \varnothing \quad (6)$$

avec $S'(r,t) \subset S(r,t)$ pour tous r et t. De façon similaire, on pourrait rajouter une famille de

contraintes pour indiquer qu'on préfère un successeur en dehors de $S(r,t)$ que pas de successeur du tout. Autre biais de notre formulation : on préférerait ne pas respecter la contrainte des Paires que de ne pas respecter la contrainte d'écart, or dans notre modélisation, la contrainte des Paires est dure, et la contrainte d'écart est associée à une pénalité.

— Dans les solutions produites par le modèle linéaire en variables 0-1, on observe souvent que des courses de même type sont planifiées dans des réunions consécutives. Nous voulons éviter cela. Une explication à cette observation pourrait être la suivante : comme mentionné précédemment, les bonnes solutions vis-à-vis de l'objectif sont les solutions avec beaucoup de courses profitables (relativement aux $\mu_{r,t}$, qui sont ici négatifs). Les réunions consécutives ayant pour un type de course donné beaucoup de successeurs en commun en pratique, il est souvent facile de diminuer la valeur de la fonction objectif en planifiant des courses de même type dans des réunions consécutives.

Ce que nous avons essayé en réponse à ce problème, et aussi en réponse au premier, est d'ajouter aux $\mu_{r,t}$ une constante, pour qu'ils soient tous positifs. Les bonnes solutions deviennent alors les solutions avec le moins de courses possibles. Les solutions produites avec cette méthode auraient pour vocation d'être complétées, ou modifiées, par exemple à l'aide de méta-heuristiques ou par les experts. Si on fait cela, les valeurs choisies pour les coefficients de la fonction économique ont en fait très peu d'impact. On peut simplement fixer les $\mu_{r,t}$ à 1 et observer des solutions similaires en termes de structure. En fait, lorsqu'on veut planifier le moins de courses possibles, il n'y a pas vraiment de marge de manœuvre pour ajuster la proportion de chaque type de course dans la solution. Cette méthode atteint l'objectif recherché : aucun type de course n'est trop représenté, et aucune course n'est redondante.

4.6.1 Complexité pratique du modèle et conclusion

Comme nous l'avons expliqué, nous avons donc testé de nombreux jeux de paramètres et de contraintes pour pallier les biais de notre formulation. Il ne serait pas forcément pertinent de détailler les résultats obtenus dans le mesure où nous avons minimisé une fonction objectif essentiellement arbitraire, avec des contraintes ne représentant pas très précisément les attentes métiers. Nous allons cependant partager quelques observations d'ordre général. (Les tests ont été réalisés avec 32 threads, sur CPLEX 12.7)

Tout d'abord, s'il est clair que la difficulté augmente avec l'ajout de nouvelles contraintes et pénalités, nous avons réussi à obtenir des solutions exactes sur les modèles les plus simples. En fixant les $\mu_{r,t}$ à -1 et les $\gamma_{r,t}$ à 5, nous avons atteint l'optimum en 4 secondes. Avec les $\mu_{r,t}$ à 1, nous avons atteint l'optimum en 129 secondes. En rajoutant le jeu de contrainte (6) présenté ci-dessus, l'optimum est atteint en 1077 secondes. Lorsque nous avons utilisé les résultats de l'apprentissage pour l'objectif plutôt que des valeurs constantes pour les μ, l'optimum n'a pas été atteint en une heure, mais le gap final était de 0,35%. Ce que l'on observe à chaque fois cependant, est que de très bonnes solutions entières sont trouvées très rapidement. Dans la figure 4.2, qui représente l'évolution du gap et du nombre de noeuds restants pour l'exécution qui ne s'est pas terminée en une heure, on peut lire en effet que le gap est tombé très bas dès le début de la recherche. En réalité la dernière solution entière a été trouvée en seulement 100 secondes, et nous avions déjà des solutions excellentes en 30 secondes.

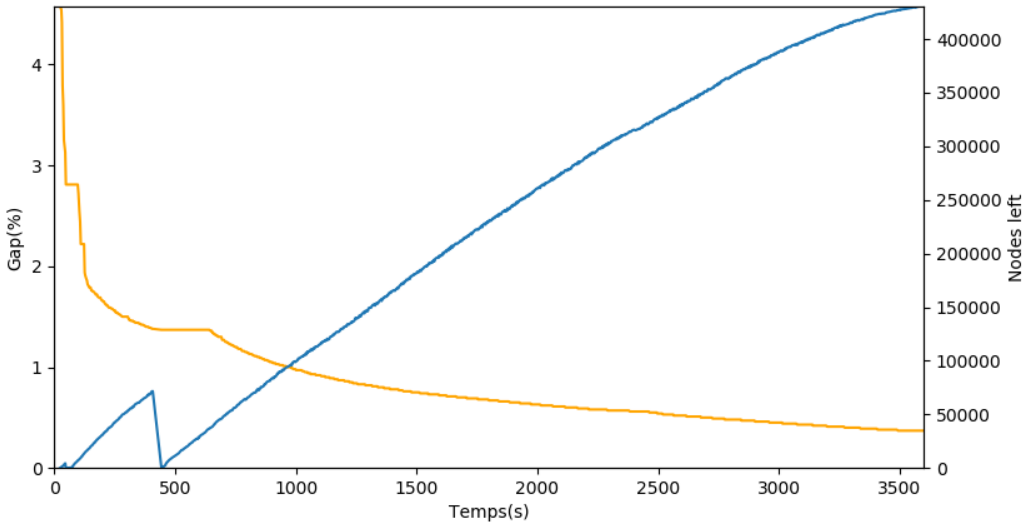


FIGURE 4.2 – Comportement typique du modèle linéaire en variables 0-1

Il est donc possible d'obtenir des programmes de courses à l'aide de la programmation linéaire, mais la méthode présente plus d'inconvénients que d'avantages. Tout d'abord, la programmation linéaire en variables 0-1 ne nous permet pas d'écrire un modèle représentant fidèlement la réalité de la planification hippique. Si l'objectif est seulement de trouver des solutions réalisables, ou d'obtenir des squelettes de solution (ce que fait le modèle avec les $\mu_{r,t}$ positifs), on peut tout aussi bien utiliser une heuristique. Il serait alors plus facile de modéliser correctement les contraintes souples et de retourner des solutions

lorsque les instances ne sont pas réalisables. Si on veut obtenir des solutions de qualité, le plus important est de pouvoir évaluer précisément la qualité des solutions et il faut donc trouver une alternative à la fonction objectif linéaire. L'utilisation d'une évaluation ad hoc sera bien plus facile avec des méta-heuristiques, qui peuvent fonctionner avec des fonctions d'évaluation « boîte noire ». Par ailleurs, et c'est peut être le problème principal, il est délicat de faire évoluer le modèle. La planification des courses de galop est un problème industriel inscrit dans un contexte où les contraintes et les modalités de planification évoluent régulièrement. Rien ne garantit qu'avec une modélisation plus précise ou que si les contraintes évoluent, il sera encore possible d'obtenir de bonnes solutions en un temps raisonnable.

En conclusion, l'étude de la modélisation du problème par un modèle linéaire en variables 0-1 justifie l'étude de méthodes de résolution approchées, que nous allons présenter dans le chapitre suivant. D'une part nous avons démontré la NP-complétude des problèmes de décision associés aux modèles proposés. D'autre part, l'utilisation d'une méthode de résolution exacte pour attaquer un problème dans lequel beaucoup de données sont incertaines n'est pas forcément adapté. Nous n'avons pas de fonction objectif précise, et les contraintes et la modélisation ne sont ni très clairement définies ni définitives. Comme nous avons vu que les modèles produisaient des solutions rapidement, la programmation linéaire pourrait être intéressante s'il était facile de modéliser fidèlement les contraintes, mais ce n'est pas non plus le cas.

4.7 Remarques sur la planification par contraintes

Avant de travailler sur la modélisation du problème sous forme d'un programme mathématique, nous avons essayé de résoudre le problème à l'aide de la programmation par contraintes (PPC). Les contraintes du problème nous avaient initialement été présentées comme des contraintes dures, et la PPC est souvent une approche utilisée en ordonnancement. Cependant, la contrainte d'écart est difficile à modéliser efficacement, et nous avons souvent rencontré des instances non réalisables avec les contraintes souples du problème modélisées comme des contraintes dures, notamment parce que les courses du préprogramme ne respectaient pas toujours ces contraintes. D'autre part, la PPC n'était pas très adaptée pour optimiser un objectif. En effet, lorsque les instances étaient réalisables, elles avaient en réalité énormément de solutions, et l'algorithme était incapable de sortir des optima locaux. Nous avons essayé d'exécuter plusieurs fois l'algorithme pour explorer un panel plus large de solutions,

mais cela n'a pas amélioré significativement la valeur des solutions obtenues. Des détails concernant la modélisation et l'implémentation sont fournis en annexe. (chapitre 5.7.2).

Chapitre 5

Méthodes approchées pour la planification des courses de galop

Contenu

L'étude théorique et pratique de plusieurs modélisations mathématiques du problème a mis en évidence à quel point il est difficile d'appliquer des méthodes de résolution exactes à la planification des courses de galop. Il est d'une part difficile d'écrire une modélisation qui représente fidèlement la réalité, et d'autre part difficile de résoudre les modèles mathématiques qui s'en approcheraient.

Dans ce chapitre nous examinons donc plusieurs possibilités pour rechercher des solutions approchées au problème. Dans un premier temps, nous présentons en section 5.1 une heuristique qui s'inspire du processus manuel de planification, capable de produire rapidement des solutions respectant la plupart des contraintes souples. Pour améliorer la solution trouvée et obtenir des alternatives à la planification manuelle, nous avons ensuite appliqué un algorithme de recherche à voisinages variables, et un recuit simulé au problème de planification. La fonction objectif à optimiser est détaillée en section 5.2, et les algorithmes de recherche locale sont donnés en sections 5.4 et 5.5. Les résultats numériques obtenus sont enfin présentés en section 5.6.

5.1 Heuristique inspirée du processus de planification manuelle

5.1.1 Réalisation manuelle du programme des courses de galop

Dans un premier temps, nous avons cherché à mettre au point une méthode heuristique pour produire rapidement des solutions de qualité raisonnable, dans le but de les améliorer et/ou les comparer à des solutions produites par des méta-heuristiques. Ainsi, nous nous sommes intéressés au processus de planification manuelle, qui semble être un point de départ pertinent pour construire une première solution heuristique. Il s'agit de la méthode utilisée actuellement pour produire les programmes de courses publiés par France Galop. Une consultation des experts de la planification au sujet de leur habitudes a donc été nécessaire et nous a permis de mieux comprendre le fonctionnement pratique des contraintes.

Lorsqu'ils créent un programme de course, les experts planifient chaque type de course successivement, suivant une ordre prédéfini. Il existe donc une hiérarchie dans le placement des courses. Les courses les plus prioritaires sont planifiées en premier et les autres successivement selon l'ordre préétabli. Les évènements annuels tels que le prix de Diane ou le prix de l'Arc de Triomphe sont bien sûr planifiés bien avant le reste du programme. Les courses de Groupe, et les courses Listed suivent également un calendrier précis, et sont planifiées en amont. Elle ne sont d'ailleurs pas du ressort de

l'outil et resteront planifiées intégralement à la main. Nous nous intéressons donc plutôt à la planification du reste des courses. Les courses pour les chevaux de deux ans sont typiquement planifiées en premier, suivis des courses pour les chevaux de trois ans, pour terminer par les courses réservées aux chevaux de quatre ans et plus. Parmi une tranche d'âge, les courses à Conditions sont planifiées en premier (notamment les courses Maiden et Inédits pour les chevaux de deux ans). Les courses à Handicap sont planifiées ensuite, et les courses à réclamer sont planifiées en dernier. Les contraintes vont être facilement respectées sur les courses placées en premier, et les courses placées en dernier doivent s'accommoder des places disponibles. Les experts ayant manifesté leur désir de conserver cette hiérarchie des types de courses dans la planification, nous avons choisi de garder cette hiérarchie des types de courses dans la mise en œuvre de l'heuristique.

Notons qu'un objectif secondaire de la mise au point d'une heuristique produisant des solutions d'une façon similaire à celle des experts est de pouvoir mettre facilement en évidence les contraintes mal interprétées ou encore implicites.

5.1.2 Description de l'heuristique

L'heuristique fonctionne comme suit (voir figure 5.1) : on commence par sélectionner un type de course, c'est-à-dire un triplet (âge, catégorie, distance). Les types de courses seront parcourus successivement suivant un ordre prédéfini. L'application des contraintes dures nous donne une liste de réunions candidates dans lesquelles on peut planifier une course du type de course en cours de traitement. Ensuite, pour déterminer dans quelle réunion il est préférable de placer la course, on calcule, pour chaque réunion candidate, un score associé au placement de la course dans cette réunion (détail en section 5.1.3). Nous déduisons ensuite des scores finaux la réunion préférentielle pour planifier la course.

Si aucune réunion n'est suffisamment intéressante au regard de son score, on passe au type de course suivant (par exemple passer des courses pour les chevaux de deux ans aux courses pour les chevaux de trois ans). Sinon, on place la course dans la meilleure réunion, et on répète le procédé pour placer la prochaine course du même type. Il s'agit donc d'un algorithme glouton. On a terminé lorsque tous les types de courses on été planifiés. Comme pour la planification manuelle, le respect des contraintes souples sera plus difficile à satisfaire pour les types de courses parcourus en dernier.

Remarque : À l'issue de l'algorithme, on va généralement avoir des réunions qui ne sont pas

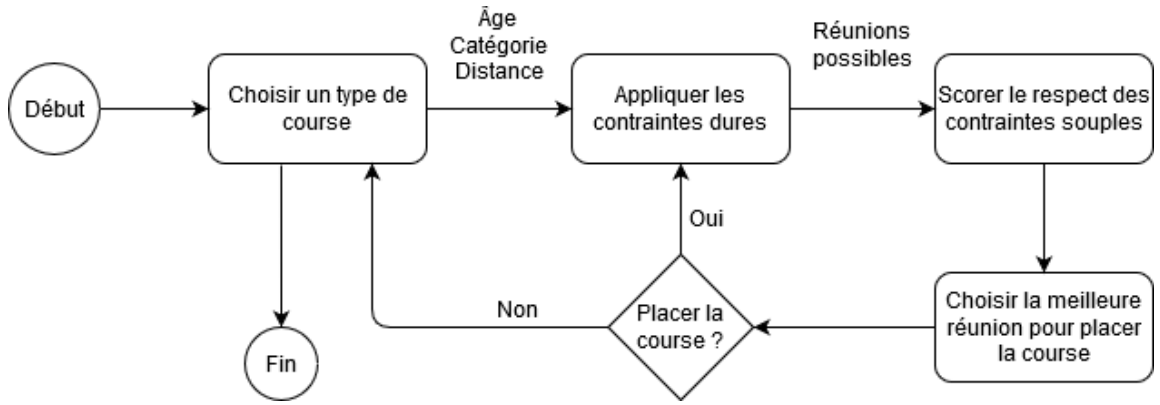


FIGURE 5.1 – Diagramme de fonctionnement de la planification manuelle

encore remplies. En effet, si toutes les contraintes sont respectées, l'algorithme glouton n'est pas poussé à planifier de nouvelles courses. Rien n'incite à ce que les réunions soient remplies. La solution produite par l'heuristique est donc une solution amenée à être modifiée et complétée. Ce n'est pas réellement un problème puisque d'une part cela laisse aux experts de la marge de manœuvre pour ajuster la planification, mais aussi parce que les experts utilisent déjà plusieurs méthodes pour remplir les réunions. Tout d'abord, les experts peuvent dédoubler les courses pour lesquelles ils anticipent beaucoup de partants. Nous planifions les courses à Handicap pour les chevaux de quatre ans et plus par paires parce qu'elles attirent beaucoup de partants, mais il n'est pas inhabituel de voir ces courses être planifiées par trois, ou par quatre en pratique. Dans le même ordre d'idée, on peut prendre une course à Handicap quatre ans et plus et la diviser en une course 4 ans et une course cinq ans et plus. Ensuite, il existe aussi des types de courses exceptionnels, planifiées à la main et qui ne sont pas gérés par l'heuristique. On trouve dans les programmes des courses trois ans et plus, des courses réservées aux jeunes jockeys, ou aux femmes jockeys par exemple. Les programmes produits par l'heuristique, et ensuite par les méta-heuristiques ont donc de toute façon pour vocation d'être complétés et retouchés par les experts.

5.1.3 Détail du calcul des pénalités du score d'une réunion dans l'heuristique gloutonne

Étant donné un programme en cours de construction, pour chacune des contraintes souples, nous avons besoin d'évaluer l'impact du placement d'une course de type t dans une réunion r. C'est l'agrégation de ces évaluations qui permet de déterminer dans quelle réunion il est préférable de placer la

course de type t. Nous détaillons ici comment cette évaluation est conduite.

Contrainte d'écart

Pour l'évaluation de la contrainte d'écart, on gère au cours de l'heuristique une liste des courses placées pour lesquelles la contrainte d'écart n'est pas encore respectée. Le placement d'une course de type t dans une réunion r se voit donc affecté un bonus s'il permet de satisfaire la contrainte d'écart d'une ou de plusieurs courses de cette liste. Le calcul est similaire à celui présenté en section 5.2.

Course à Handicap et à Réclamer

On pénalise le placement d'une course à Handicap ou à Réclamer dans une réunion qui en contient déjà assez, mais on veut aussi pénaliser le placement d'une course à Conditions dans une réunion qui ne contient pas suffisamment de courses à Handicap et à Réclamer. Le premier point est simple, il suffit de compter le nombre de courses à Handicap et à Réclamer dans r, et d'affecter la pénalité en fonction. Pour le second, on additionne le nombre de courses à Handicap et à Réclamer de la réunion r avec le nombre de places restantes dans la réunion pour voir s'il sera possible de respecter la contrainte, et on calcule la pénalité en fonction de ce nombre. Par exemple, si une réunion a une capacité de 8 courses, et que 5 courses à Conditions sont déjà programmées, on pénalisera l'ajout d'une sixième course à Condition, qui rendrait la planification de trois courses de catégorie Handicap ou Réclamer impossible.

Placement des courses par paire

Si le type t doit être placé par paire, et que la réunion r contient déjà une unique course de type t, le placement d'une nouvelle course t dans la réunion reçoit un bonus. On affecte également un malus à la planification d'une course de type t dans une réunion qui n'en contient pas et qui n'a plus qu'une place.

Pour ces trois contraintes, le but du calcul de pénalités est de guider le placement des courses pour l'heuristique gloutonne. En revanche, ces pénalités ne permettent pas l'évaluation de la qualité des solutions produites. Ces pénalités ne déterminent pas un score global, mais un score par réunion, vis-à-vis du placement d'une nouvelle course. Nous allons, dans la section suivante, présenter la fonction objectif que nous allons utiliser, pour évaluer les solutions produites par l'heuristique d'une part, et pour guider la recherche de solutions par les méta-heuristiques d'autre part.

5.2 Fonction d'évaluation

Pour guider la recherche de solutions améliorantes dans une méta-heuristique, nous avons besoin d'une fonction d'évaluation, qui peut affecter un score à tout programme de courses. Dans cette section, nous supposons que nous disposons d'un programme de courses initial, que nous cherchons à évaluer relativement aux préférences des experts en planification. Dans le chapitre précédent, nous avons utilisé une fonction d'évaluation linéaire, pour pouvoir l'utiliser dans un programme linéaire en variables 0-1. Dans une méta-heuristique, nous pouvons choisir n'importe quelle fonction d'évaluation (en particulier, non linéaire) que nous pouvons calculer, et l'utiliser comme une « boîte noire ». Nous ne savons cependant toujours pas ce qu'est une « bonne » fonction d'évaluation d'un programme de courses. Les contraintes souples reflètent l'idée qu'ont les experts de la planification manuelle d'un bon programme. Cependant, notre formulation algébrique des contraintes souples ne permet pas de traduire fidèlement leur utilisation dans la planification manuelle. Pour les contraintes d'écart, par exemple, les experts vont préférer que la course successeur d'une autre soit planifiée le plus au centre possible de l'intervalle choisi. De plus, ils préféreraient aussi avoir un successeur légèrement en dehors de l'intervalle, que de ne pas avoir de successeur du tout. La mise au point d'une nouvelle fonction objectif non linéaire permet donc de mieux prendre en compte les préférences des décideurs vis-à-vis de la satisfaction des différentes contraintes souples.

Ainsi, nous avons choisi d'utiliser une fonction d'évaluation qui évalue le respect des contraintes souples, et inclut d'autres critères empiriques relatifs à l'homogénéité des réunions. Puisque nous allons associer des pénalités à la violation des contraintes, nous allons *minimiser* la fonction objectif.

L'horizon de planification est typiquement de trois mois. Comme expliqué précédemment, nous ajoutons à la fin de cet horizon de planification trois semaines de réunions fictives, pour minimiser l'impact de l'effet de bord associé à cet horizon sur la planification finale. Les réunions de cette extension seront traitées comme les autres lors de l'évaluation du respect des contraintes, sauf que les courses de ces réunions n'auront pas besoin d'avoir un successeur. Grâce à l'extension, la méta-heuristique va être capable d'anticiper le prochain programme de courses et de garantir la faisabilité de la transition inter-saison. Lorsque l'on planifiera le programme suivant, il n'y aura donc a priori pas de difficulté particulière pour satisfaire les contraintes au début du programme, puisque le respect de ces contraintes aura été anticipé grâce à l'extension. L'extension de planification est bien sûr supprimée à

la fin de l'algorithme et n'est pas présente dans la solution de planification finale retournée par l'outil.

Nous allons maintenant détailler le calcul des pénalités associées à la violation de chacune des contraintes souples : la contrainte d'écart, la contrainte des Paires, et la contrainte HR.

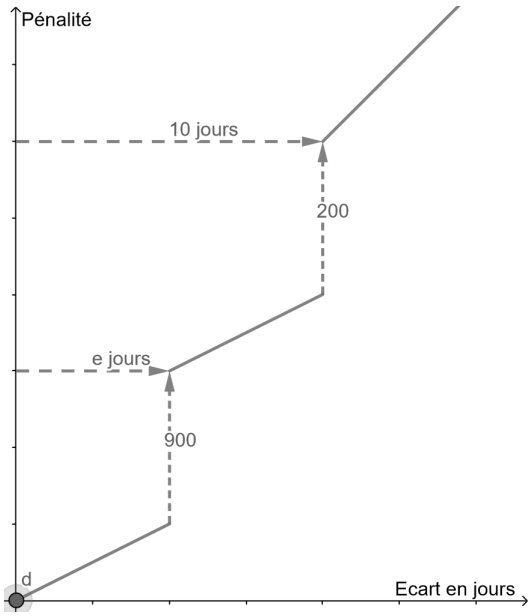
La contrainte d'écart Le respect de la contrainte d'écart est évalué ainsi : après chaque course de type t, on veut trouver une course de type t' planifiée entre a et b jours plus tard. On note d le centre de l'intervalle cible, avec $d = (a + b)/2$. On note e la demi-largeur de cet intervalle cible : $e = (b - a)/2$. L'intervalle cible dépend de la distance sur laquelle les courses du type t sont courues, on liste ci-dessous les différentes valeurs que prennent a et b en fonction de la distance :

- Pour les courses Flyer, correspondant à des distances inférieures 1400 m, l'intervalle cible est [15 ;21] jours, de sorte que d =18 j et e =3 j.
- Pour les courses Miler, Intermediate et Classical, avec des distances entre 1450 m et 2400 m, l'intervalle est de [18 ; 25] jours, de sorte que d =21,5 j et e =3,5 j.
- Pour les courses Stayer dont les distances dépassent 2450 m, l'intervalle est [21 ; 30] jours, de sorte que d =25,5 j et e =4.5 j.

Pour calculer la valeur de la pénalité associée à la violation de la contrainte d'écart, pour chaque course de type t qui ne fait pas partie de l'extension de planification, on cherche la course de type t' planifiée au plus proche de l'écart cible de d jours. Ainsi, pour chaque jour séparant le successeur de la date cible dans l'intervalle [d-10 ;d+10], on compte une pénalité 10. La pénalité maximale pour un écart entre la date de la course de type t et la date cible pour la course de type t' dans cet intervalle est donc de 100. De plus, si l'écart dépasse e jours, on compte une pénalité supplémentaire de 900. Si l'écart dépasse 10 jours, on compte une pénalité supplémentaire de 200, puis une pénalité supplémentaire de 20 pour chaque jour d'écart dans l'intervalle]d+10 ; d+20] ou [d-20 ; d-10[. Si la course de type t n'a pas de successeur du tout, même en dehors de l'intervalle cible, on fixe la pénalité à 11000. Le calcul de la pénalité en fonction de la valeur absolue de l'écart est résumé dans la figure 5.2. Avoir une pénalité qui dépend de façon discontinue de l'écart permet de mettre l'accent sur le respect de la contrainte souple définie par l'intervalle $[d - e; d + e]$, tout en guidant la méta-heuristique vers les meilleurs solutions.

Ces valeurs ainsi que le profil de la pénalité ont été choisis de la façon suivante, sur la base de critères que nous voulions que l'évaluation de la contrainte d'écart respecte. Les trois paliers de

FIGURE 5.2 – Pénalité associée à la violation des contraintes d'écart



malus correspondent respectivement aux courses qui respectent la contrainte, aux courses qui ont un successeur mais en dehors de l'intervalle cible, et aux courses qui n'ont pas véritablement de successeur. Le saut entre le premier et le second palier est bien sûr le plus important puisqu'il constitue la limite entre le respect et le non-respect de la contrainte. Nous voulions de plus qu'une course ne respectant pas la contrainte d'écart soit pénalisée plus fortement qu'une course ne respectant pas les autres contraintes souples. En effet, les experts nous ont notifié que cette contrainte d'écart est la plus importante. Nous avons choisi une pente plus forte dans le troisième palier pour inciter la méta-heuristique à rapprocher deux courses vraiment trop éloignées, et guider ainsi la recherche locale vers des solutions dans lesquelles les sorties des intervalles d'écart sont équilibrées et localement faibles plutôt que vers des solutions avec des intervalles globalement satisfaits mais des sorties très grandes. Par exemple, si trois courses a, b, et c se succèdent, on préfère avoir un écart entre a et b de 10 jours et un écart entre b et c de 11 jours, plutôt qu'un écart entre a et b de 6 jours et un écart entre b et c de 15 jours.

Comme décrit en section 1.4, la contrainte d'écart s'applique entre les courses d'un même type, c'est-à-dire $t = t'$ mais aussi entre les courses d'Inédits et les courses Maiden (une course d'Inédits doit donc avoir un successeur Inédits, mais aussi un successeur Maiden). En effet, les courses d'Inédits sont

n	0	1	2	3	4	5	6	7	8
Pénalité	1000	700	500	0	0	0	500	700	1000

TABLE 5.1 – Tableau des pénalités pour les contraintes CS3

réservées aux chevaux n'ayant jamais couru, alors que les courses Maiden sont réservées aux chevaux n'ayant jamais gagné. La très grande majorité du temps, un cheval ayant couru une course d'Inédits va ensuite courir une course Maiden. Il est donc important de planifier des courses Maiden après les courses d'Inédits suivant la contrainte d'écart.

Contrainte souple HR La contrainte souple HR impose aux réunions de contenir entre 3 et 5 courses de type Handicap ou Réclamer. Ces types de courses sont typiquement les moins sélectifs, et on veut que chaque réunion contienne un bon mélange de courses sélectives et de courses non sélectives. La pénalité donnée à une réunion r ne satisfaisant pas la contrainte CS3 dépend du nombre de courses de type HR programmées dans ladite réunion. Le tableau 5.1 récapitule ces pénalités en fonction du nombre n de courses de type HR programmées dans une réunion. Si $n > 8$, la pénalité est de 1000. Pour fonctionner, la fonction objectif doit être capable d'évaluer des solutions partielles, dans lesquelles les réunions ne sont pas encore remplies. Pour évaluer la pénalité affectée à une réunion, on calcule pour n la valeur encore atteignable en remplissant la réunion qui résultera en la pénalité la plus faible. Par exemple, si une réunion r a une capacité de 6 courses, parmi lesquelles 4 sont des courses de Groupes préprogrammées, et les deux autres places étant encore libre, il est possible de planifier deux courses à Handicap ou à Réclamer. Le n calculé est donc $n = 2$, et la pénalité est de 500. Si parmi les deux places on planifie une course à Conditions par exemple, il n'est plus possible d'atteindre $n = 2$, et on calcule donc $n = 1$ comme étant le meilleur scénario, et la pénalité devient 700. On voit donc que, en plus de produire un score plus fidèle à la qualité du programme pour des programme partiellement remplis, calculer le score de cette façon permet de pénaliser la planification de courses autres que Handicap et Réclamer lorsqu'elle sont déjà en nombre insuffisant.

Contrainte de Paires Cette contrainte souple indique que certains types de courses doivent être planifiés par paires. Si parmi les courses soumises à cette contrainte, l'une est programmée en monôme, on ajoute à l'évaluation une pénalité de 1000.

Contrainte des Types La contrainte des Types n'est pas une contrainte formellement énoncée par les experts, et elle n'a pas été présentée dans la section 1.4, mais elle est néanmoins importante dans le processus de planification. Avec les scores présentés jusqu'à présent, une méta-heuristique n'est jamais poussée à planifier des types de courses n'étant pas présents dans le préprogramme, aucun critère n'évalue le placement de nouveaux types de courses. C'est cependant indispensable dans plusieurs cas de figure. Le cas de figure le plus important est lors de la planification des courses de printemps. Puisque les chevaux de deux ans ne sont autorisés à courir qu'à partir du 25 mars, il est évident que le préprogramme des courses de février ne contient aucune course pour les chevaux de deux ans. Il est pourtant très important de planifier des courses pour les chevaux de deux ans dans le programme de printemps (et dans les programmes suivants), et de les planifier le plus tôt possible.

Le score pour cette contrainte est évalué comme suit : chaque type de course n'étant pas présent dans le programme ajoute 2000 au score du programme. Un type de course étant planifié pour la première fois n jours après le début du programme ajoute $10n$ au score du programme.

Pour que la contrainte fonctionne il est important qu'un type de course non planifié soit pénalisé plus qu'un type planifié en fin de programme. Le programme (en comptant l'horizon de planification, le préprogramme, et l'extension de planification) peut comporter 5 mois, soit environ 150 jours, d'où les valeurs choisies.

Une remarque importante sur ce critère d'évaluation, est qu'il n'est pas pris en compte par l'heuristique, ce qui sera important à considérer lorsque nous présenterons les résultats numériques. La nature gloutonne de l'heuristique fait qu'il est très difficile de savoir s'il sera possible de respecter les contraintes pour un nouveau type de course rajouté. Les tentatives pour incorporer cette contrainte dans l'heuristique ont résulté en une dégradation très importante de la satisfaction des autres contraintes pour les types rajoutés, et l'heuristique était simplement mauvaise. Il n'aurait pas été intéressant de se comparer à une telle heuristique. Nous reviendrons sur cette question lors de la présentation des résultats dans la section 5.6.

Hétérogénéité En complément de l'évaluation du respect des contraintes souples, la fonction d'évaluation mesure également l'hétérogénéité du programme. Nous voulons que les entraîneurs aient un éventail de possibilités aussi large que possible lorsque qu'ils veulent inscrire un cheval à une course. Nous voulons donc éviter que deux courses attirant une population de chevaux similaires soient plani-

fiées proches l'une de l'autre. Nous avons considéré dans le calcul de cette pénalité que deux courses sont d'un type proche, c'est-à-dire qu'elles se font concurrence, si elles ne diffèrent que par la distance courue, c'est-à-dire ont le même âge et la même catégorie.

Pour évaluer l'hétérogénéité du programme, on calcule sur une fenêtre glissante de quatre jours, combien de fois chaque couple (distance, catégorie) est planifié. Pour chaque couple planifié $k > 0$ fois, on affecte une pénalité de $k - 1$.

Exemple : Si on planifie une course de type (Handicap, 3 ans, 1400 m) le 21 janvier, et une course de type (Handicap, 3 ans , 1800 m) le 23 janvier, ces deux courses sont considérées comme étant de type similaire : l'âge et la catégorie sont les mêmes. Ces deux courses sont présentes dans les fenêtres de quatre jours suivantes : 20/01 -> 23/01 et 21/01 -> 24/01. Pour chacune de ces deux fenêtres, on compte une pénalité de 1 ($k = 2$), pour une pénalité totale de 2. Ainsi, en plus de dépendre du nombre de courses similaires proches les unes des autres, l'évaluation de cette contrainte prend aussi en compte le degré de proximité de ces courses.

Synthèse En résumé, les trois contraintes souples principales ajoutent une pénalité d'environ 1000 à l'objectif lorsqu'elles sont violées. La contrainte portant sur les courses à Handicap et à Réclamer pouvant être plus ou moins respectée, nous avons plusieurs pénalités entre 0 et 1000. Comme nous voulons que la contrainte d'écart soit la plus importante, la pénalité pour le non respect de cette dernière commence à 1000 et augmente avec la distance du successeur au centre de l'intervalle ciblé.

Remarque : Quelle que soit la méta-heuristique implémentée, l'évaluation de la fonction objectif est la partie la plus coûteuse, notamment l'évaluation de la contrainte d'écart puisqu'elle nécessite un examen de la globalité de la planification.

5.3 Voisinages

Pour améliorer une solution de planification de base, deux méta-heuristiques basées sur le parcours de voisinages ont été mises en œuvre : une recherche à voisinages variables (VNS) et un recuit simulé (SA). Étant donné un programme de courses, nous avons travaillé avec les voisinages $\{V_1, V_2, V_3, V_4\}$ suivants :

— V_1 : Ajout d'une course ou d'une paire de courses.

— V_2 : Suppression d'une course ou d'une paire de courses.

— V_3 : Échange de deux courses (2-opt), ou de deux paires de courses, espacées d'au plus trois jours.

— V_4 : Application du mouvement de V_3 trois fois.

Les ajouts, suppressions, et échanges, sont des voisinages très standards, mais nous avons eu besoin d'appliquer ces mouvements également à des paires de courses. Sans ce voisinage dédié aux paires, nous avons observé que le parcours effectué restait trop local. En effet, il n'est pas possible d'ajouter ou supprimer des courses devant être planifiées par paire (contraintes de Paires) sans violer la contrainte souple, cela dégrade beaucoup la valeur de la fonction objectif.

Pour les deux méta-heuristiques que nous présentons, nous ne travaillons que sur des solutions respectant les contraintes dures. Ainsi, ces dernières sont toujours vérifiées avant l'application d'un mouvement. De plus, le préprogramme n'est pas modifié.

5.4 RVNS

La première méta-heuristique que nous avons étudiée est la version « réduite » de la recherche à voisinages variables (RVNS) [19]. Nous avons en effet observé qu'une implémentation classique d'une VNS n'était pas efficace. L'évaluation d'un voisin combinée au parcours du voisinage rendaient la descente intra-voisinages bien trop chronophage et trop peu de voisins étaient alors explorés.

L'algorithme que nous avons utilisé est le suivant :

Algorithme 3 : RVNS

```
Iteration ← 0;
tant que Iteration < Iteration_max faire
    k ← 1 ; tant que k ≤ 4 faire
        Choisir un voisin dans V_k au hasard;
        si La valeur de l'objectif est améliorée alors
            Appliquer le changement;
            k ← 1 ;
        sinon
            k ← k + 1;
    Iteration ← Iteration + 1;
```

5.5 Recuit simulé

La seconde méta-heuristique que nous avons implémentée est un recuit simulé [20]. A l'inverse de la VNS, nous n'avons pas besoin de voisinage large pour éviter les optima locaux, puisque nous acceptons des changements qui dégradent la fonction objectif. Nous n'utilisons donc pas V_4 dans cette recherche locale. Dans le pseudo-code suivant, T est la température, de valeur initiale T_0. On note N, le nombre de transformations tirées au sort depuis la dernière amélioration de la meilleure solution rencontrée jusqu'à présent (dans le palier de température courant) ; il est borné par N_{Max}. La température diminue quand N_{Max} mouvements sont essayés sans amélioration de la meilleure solution. La variable Stagnation désigne, quant à elle, le nombre de paliers de température consécutifs n'ayant pas vu d'amélioration de la meilleure solution rencontrée jusqu'à présent. L'algorithme se termine lorsque 2 paliers de température consécutifs n'améliorent pas la meilleure solution rencontrée jusqu'à présent, et la température est assez faible. r est le taux de décroissance de la température.
L'algorithme 4 décrit l'algorithme de recuit simulé utilisé.

Algorithme 4 : Recuit simulé

```
T ← T_0;
Stagnation ← 0;
N ← 0;
tant que T > 1 ou Stagnation ≤ 2 faire
    pour V ∈ {V_1, V_2, V_3} faire
        N ← N + 1;
        Choisir un voisin au hasard dans V;
        Évaluer la variation de la valeur de la fonction objectif Δf;
        Appliquer le changement si Δf < 0 ou avec une probabilité de e^{-Δf/T};
        si La meilleure solution rencontrée jusqu'à présent est améliorée alors
            La meilleure solution sauvegardée est mise à jour;
            N ← 0;
            Stagnation ← 0;
    si N > N_Max alors
        T ← r · T;
        Stagnation ← Stagnation + 1 ;
        N ← 0 ;
```

Instance	Objectif	C. d'écart	C. HR	C. des Paires	Hétérogénéité	C. des Types
A2019	102812	18325	13100	4000	607	66780

TABLE 5.2 – Répartition typique des composantes du score

5.6 Résultats numériques

Nous avons effectué des tests sur quatre instances réelles et sur quatre instances artificielles, soit un total de huit instances. Les instances réelles sont la planification des programmes d'automne et de printemps, 2018 et 2019. On notera ces instances A2018, P2018, A2019, P2019. Nous rappelons que l'automne est la période de trois mois de septembre à novembre et que la période de printemps s'étend de mars à mai ; le programme manuel du mois d'août est ainsi utilisé comme préprogramme pour l'automne, et celui de février pour l'hiver. Nos tests sur des instances réelles sont concentrés sur ces quatre trimestres pour deux raisons. Les programmes d'été et d'hiver sont très différents puisque la plupart des courses prennent place dans le même hippodrome, dans ce qu'on appelle un meeting. Les planifications d'hiver et d'été ne correspondent donc pas vraiment au problème étudié. D'autre part, les règles évoluent au fil du temps : les catégories de courses étaient différentes avant 2018. Enfin, les programmes de 2020 on été fortement perturbés, le programme de printemps 2020 étant essentiellement annulé en raison de la pandémie. C'est pourquoi nous avons finalement choisi de nous concentrer sur les instances de 2018 et 2019. Pour les quatre instances générées, nous avons repris les quatre instances réelles, mais en mélangeant aléatoirement les courses du préprogramme. On les notera RA2018, RP2018, RA2019 et RP2019. On obtient ainsi des instances réalistes en termes de type et de quantité de courses dans le préprogramme, et en utilisant un calendrier réel de réunions.

Tous les tests on été effectués sur une machine dotée d'un Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz.

Avant de parler des résultats numériques, il est aussi important de comprendre certains aspects de la fonction objectif. Les valeurs prises par la fonction objectif ne reflètent pas la distance à un programme idéal, et peuvent seulement être utilisées pour comparer deux solutions de la même instance. Rappelons que certaines courses, pour les chevaux de 2 ans notamment, ne sont planifiables qu'en fin de saison et donc non présentes dans le préprogramme ; par exemple, les courses de deux ans Stayer. Si un type de course n'apparaît pas dans le programme final, il ajoutera 2000 au score calculé par la fonction objectif, du fait de la contrainte des Types. S'il est planifié pour la première fois plus de 30 jours (plage de temps

du préprogramme) après le début du programme, il pénalisera le score d'au moins $10 \cdot 30 = 300$. Il est donc inévitable, même dans une planification parfaite, que le score pour la contrainte des Types soit élevé. Pour les autres contraintes, il y a également des pénalisations inévitables. Par exemple, pour la contrainte d'écart et pour les courses qui ne sont pas des courses Flyer, la distance cible e de la course au centre de l'écart est un demi-entier, tout successeur sera donc éloigné d'au moins un demi jour de la date cible. Chaque course n'étant pas Flyer contribue donc au moins 5 dans le score vis-à-vis de la contrainte d'écart, ce qui cumulativement peut compter pour plusieurs milliers dans le score. À titre indicatif, nous avons représenté dans la table 5.2 le profil typique des différentes composantes du score. En regardant le détail de l'analyse on remarque que toutes les courses pour lesquelles la contrainte d'écart n'est pas respectée sont des courses du début du préprogramme pour lesquelles le successeur aurait dû se trouver en tout début de l'horizon de planification : il n'est pas toujours possible de respecter toutes ces contraintes compte tenu de la place disponible en début de programme. Pour la contrainte HR et la contrainte des Paires, les réunions ne les respectant pas sont toutes des réunions de l'extension de planification. Cela se produit car il est plus intéressant vis-à-vis de la contrainte des Types de planifier dans l'extension des courses n'étant pas encore dans le programme que d'y respecter les contraintes HR et des Paires. Au final, en dépit de valeurs pouvant paraître élevées, les contraintes sont respectées pour les courses de l'horizon de planification, et le programme est de qualité.

5.6.1 Recuit simulé

Pour tester les performances du recuit simulé, présenté en section 5.5, nous commençons par tester l'algorithme avec plusieurs jeu de paramètres pour N_{Max}, T_0, et r, sur l'instance d'automne 2019. Cette instance est pertinente pour sélectionner des bons jeux de paramètres puisqu'elle est la plus récente et dispose d'un préprogramme plus complet que les instances de printemps. Le but est notamment d'étudier l'impact des différents paramètres sur les résultats. Une fois le jeu de paramètres sélectionné, on teste avec ce jeu de paramètres les 8 instances générées. Lors de ces tests, on compare les résultats obtenus avec la solution heuristique.

Dans les tableaux présentés, les résultats de chaque ligne sont des valeurs moyennes calculées à la suite de dix exécutions du recuit simulé. Dans tous les tests sur le recuit, la solution initiale utilisée ne contient que les courses des réunions du préprogramme, et toutes les autres réunions sont vides.

Dans la table 5.3, on présente les résultats obtenus dans la phase du choix des paramètres. Les

N_{Max}	r	T_0	Objectif	Dont contraintes des Types	Autres Contraintes	Temps (s)
3000	0,85	1000	108256	68396	39860	764
5000	0,85	1000	106147	68955	37192	1126
5000	0,95	5000	104541	68016	36525	2216
10000	0,85	1000	103631	68167	35464	2139
10000	0,95	3000	103438	67433	36005	4088

TABLE 5.3 – Choix des paramètres pour le recuit simulé, instance A2019

trois premières colonnes indiquent le jeu de paramètres utilisé. La première colonne de résultats est la valeur finale de la fonction objectif englobant les pénalisations associées à la satisfaction de l'ensemble des contraintes souples. Dans les 2 colonnes suivantes, on donne la décomposition de cette valeur tout d'abord en le score attribué par l'évaluation de la contrainte des Types seule puis le score associé au reste des contraintes. Le score par rapport à la contrainte des Types met en lumière la quantité de types de courses différents qu'il a été possible de planifier. Le score associé aux autres contraintes reflète à quel point les contraintes de planification sont respectées. Dans la dernière colonne est reporté le temps d'exécution de l'algorithme de recuit.

La conclusion principale que l'on tire de ces tests est que le paramètre N_{Max} (qui représente le nombre d'itérations à température constante) est celui qui a le plus d'impact sur les résultats. Les résultats avec $N_{\text{Max}} = 3000$ sont moins bons que ceux avec $N_{\text{Max}} = 5000$, eux même moins bons que que ceux avec $N_{\text{Max}} = 10000$. Si on compare la troisième ligne ($N_{\text{Max}} = 5000$, r =0,95 et $T_0 = 5000$) avec la quatrième ($N_{\text{Max}} = 10000$, r =0,85 et $T_0 = 1000$), on remarque que les résultats sont meilleurs pour la quatrième ligne, alors que le temps d'exécution est inférieur. Cela montre bien que, pour obtenir des solutions de meilleure qualité, il est plus intéressant d'augmenter N_{Max} que de modifier r ou T_0. D'autre part, même en gardant $N = 10000$ et en augmentant r à 0,95 et T_0 à 3000, on voit, dans la dernière ligne, que la qualité de la solution n'est que très peu améliorée, pour un temps de calcul quasiment deux fois plus important. De plus, cette amélioration n'est pas statistiquement significative : sans rentrer dans les détails, l'écart type des valeurs des solutions produites par le recuit simulé tourne autour de 1500, les moyennes ont été obtenues avec 10 exécutions, et $1500/\sqrt{10}$ est sensiblement plus important que l'amélioration observée.

Nous avons donc choisi pour la suite des tests, le jeu de paramètres $N_{\text{Max}} = 10000$, $r = 0.85$ et $T_0 = 1000$. Par ailleurs, on peut noter qu'initialiser T_0 à la valeur 1000 revient à accepter la moitié des solutions dégradantes au début de l'algorithme, conformément aux recommandations standards [21].

	A2019	RA2019	A2018	RA2018	P2019	RP2019	P2018	RP2018
Objectif H.	233268	233621	251830	251196	279569	279191	265175	265239
Objectif	103631	101816	106415	106770	126284	126273	132643	132601
Dont Types H.	189970	190010	197530	197610	265070	265040	261680	261740
Dont Types	68167	67546	68849	68782	112209	112313	120566	120573
Autres H.	43298	43611	54300	53586	14499	14151	3495	3499
Autres	35464	34270	37566	37988	14075	13960	12077	12028

TABLE 5.4 – Tests du recuit simulé avec les paramètres sélectionnés sur les 8 instances

Avec ce jeu de paramètres, nous avons exécuté 10 fois le recuit sur chacune des 8 instances, et à chaque fois nous avons comparé les résultats à l'heuristique (qui elle est déterministe et est donc exécutée une seule fois). Les résultats sont reportés dans le tableau 5.4. Chaque colonne est associée à une instance. Comme précédemment, on retrouve sur chaque ligne les valeurs moyennes obtenues pour l'objectif, puis le score relatif à la contrainte des Types, et enfin le score relatif aux autres contraintes. Pour chacune des trois valeurs, on a renseigné la valeur obtenue par le recuit et celle obtenue par l'heuristique.

Comme mentionné dans la partie 5.2, l'heuristique ne prend pas en compte la contrainte des Types, c'est donc sans surprise que l'on observe une amélioration significative du respect de la contrainte des Types dans les solutions produites par le recuit, et cette amélioration se répercute sur l'objectif global. Il est donc plus pertinent de comparer le score de l'heuristique avec le score relatif aux autres contraintes, et c'est en partie la raison pour laquelle nous avons reporté séparément les trois scores. Pour les instances d'automne, le score suivant les contraintes « autres » est aussi considérablement plus faible pour les solutions produites par le recuit dans les quatre instances. Pour les instances de printemps, il faut noter que le préprogramme est très différent de celui des programmes d'automne (et c'est aussi l'intérêt d'avoir testé des instances d'automne et de printemps). Le préprogramme (le calendrier de février) ne contient que peu de courses, et ne contient notamment pas de courses pour les chevaux de deux ans, qui ne commencent qu'à partir du 25 mars. On observe donc assez logiquement du côté de l'heuristique des scores très importants pour la contrainte des Types, et très faible pour les autres contraintes (car il y a peu de courses à planifier pour respecter les autres contraintes). Comme pour les instances d'automne, le recuit obtient de bien meilleurs scores sur la contrainte des Types que l'heuristique, ce qui signifie qu'il planifie bien plus de types de courses, et que les programmes produits sont bien plus remplis. Il est donc positif que le recuit soit également meilleur que l'heuristique pour le respect des contraintes autres pour les instances de printemps 2019. Toujours pour les instances de

	Heur.	Recuit	Exéc.1	Exéc.2	Exéc.3	Exéc.4	Exéc.5
Objectif	233268	103631	143430	138098	172115	163374	135625
Dont Types	189970	68167	74870	75970	73900	69900	78030
Autres	43298	35464	68560	62128	98215	93474	57595

TABLE 5.5 – Tests de la RVNS sur l'instance A2019

printemps 2019, le score sur les contraintes « autres » pour les programmes produits par le recuit est, en revanche, bien plus important que pour l'heuristique. Cela s'explique par le fait que le recuit planifie bien plus de courses ; il ne faut donc pas forcément en déduire que les contraintes « autres » sont moins bien respectées en moyenne sur les solutions produites par notre méta-heuristique.

Enfin, nous n'observons pas de différences importantes entre les instances réelles et les instances artificielles, ce qui montre une certaine robustesse de la méthode vis-à-vis du préprogramme fourni en entrée.

5.6.2 RVNS

Pour l'algorithme RVNS, nous avons testé l'algorithme sur l'instance A2019. Puisque l'objectif est de comparer la RVNS avec le recuit simulé, nous avons fait tourner la RVNS pendant 3600 s, un temps comparable avec celui des exécutions du recuit simulé. La table 5.5 contient les résultats obtenus pour 5 exécutions. Comme précédemment, on présente tout d'abord le score global de l'objectif, puis celui associé aux contraintes des Types seules, et enfin celui du reste des contraintes. Dans les deux premières colonnes, nous avons rappelé, pour faciliter la comparaison, les résultats obtenus sur l'instance A2019 avec l'heuristique et avec le recuit (en moyenne sur 10 exécutions). Les cinq colonnes suivantes contiennent les résultats obtenus sur les cinq exécutions de l'algorithme RVNS. On lit d'abord que la RVNS est beaucoup moins efficace que le recuit simulé, à la fois sur la contrainte des Types et sur les contraintes autres. Comme la RVNS est capable de travailler avec la contrainte des Types, elle est meilleure que l'heuristique sur cet aspect, mais au détriment du respect des autres contraintes. De plus, on remarque que les résultats sont très variables. On atteint un objectif de 135625 dans l'exécution 5, mais de seulement 172115 dans l'exécution 3. Enfin, il apparaît en analysant la trace des exécutions qu'aucun changement améliorant n'est trouvé par la RVNS après les 15-20 premières minutes d'exécution. Il est donc clair que la RVNS tombe dans des optimums locaux, et n'est pas équipée pour en sortir, malgré l'utilisation du voisinage $V4$. La RVNS est donc en l'état incapable de

rivaliser avec le recuit. Nous n'avons donc pas poursuivi les recherches dans ce sens. Pour améliorer la RVNS, il serait nécessaire de trouver des voisinages plus complexes et plus spécialisés qui permettent à la RVNS de sortir des optima locaux. Par exemple, on pourrait essayer de déplacer toutes les courses d'un type à la fois, pour conserver le respect de la contrainte d'écart. L'implémentation serait compliquée et les résultats incertains, nous n'avons donc pas donné la priorité à ce genre de pistes.

5.7 Optimisation bi-objectif à l'aide d'un recuit simulé modifié

Jusqu'à présent dans le chapitre 5, nous avons cherché des solutions qui respectent les contraintes dures, et qui minimisent la fonction objectif présentée en section 5.2, c'est-à-dire qui respectent le plus possible les contraintes souples. Par ailleurs, nous souhaitons aussi produire des programmes qui vont attirer le plus de partants possible. Nous présentons, dans cette section, un algorithme de recuit simulé intégrant efficacement, dans sa fonction d'évaluation, les prédictions du nombre de partants calculées via une méthode d'apprentissage.

5.7.1 Présentation de l'algorithme de recuit simulé multi-objectif avec priorité

Nous nous donnons une fonction f_1 et une fonction f_2, et nous souhaitons minimiser la somme $f = f_1 + f_2$. Cette minimisation peut par exemple s'effectuer avec un recuit simulé en évaluant à chaque itération la valeur de l'objectif $f = f_1 + f_2$.

Dans notre problème de planification des courses de galop, la fonction f_1 est l'évaluation utilisée jusque-là dans le recuit simulé, et présentée en section 5.2. Évaluer f_1 prend 1,5 milliseconde (pour nos instances). La fonction f_2 serait une estimation du nombre de partants attendus pour chacune des courses. En prenant, par exemple, l'algorithme de simulation présenté en section 3.4, l'évaluation de f_2 prendrait 350 millisecondes. Dans la section 5.6 nous avions des temps d'exécution pour le recuit d'environ 2000 secondes. En supposant que la durée de chaque itération passerait de 1,5 ms à 351,5 ms, l'application du recuit simulé avec pour objectif $f = f_1 + f_2$ aurait donc un temps de calcul d'un peu plus de 5 jours ($2000 \cdot (351{,}5/1{,}5)/(3600 \cdot 24)$). Un tel temps de calcul, bien que pas totalement inenvisageable pour notre application, ne permettrait pas, par exemple, d'appliquer le recuit plusieurs fois pour soumettre plusieurs programmes aux experts parmi lesquels ils pourraient faire leur choix.

Nous proposons ici une variante de l'algorithme du recuit simulé ayant pour but la minimisation de

l'objectif $f_1 + f_2$ tout en faisant aussi peu d'appels à f_2 (l'évaluation la plus coûteuse) que possible. La modification porte uniquement sur le calcul pour déterminer l'acceptation ou non d'un changement. Dans l'algorithme classique du recuit, on calcule la variation Δf de la fonction objectif à minimiser, et on accepte le changement si $\Delta f < 0$ ou avec une probabilité $\exp(-\Delta f/T)$. Dans l'adaptation de l'algorithme proposée, pour décider de l'acceptation d'un changement, nous introduisons un paramètre μ, et appliquons le critère suivant :

Algorithme 5 : Critère d'acceptation des changements du recuit modifié

```
Calculer Δ1 = Δf1 + Tμ;
si Δ1 < 0 alors
    Calculer Δ2 = Δf2 − Tμ;
    si Δ1 + Δ2 < 0 alors
        Accepter le changement;
    sinon
        Accepter le changement avec une probabilité exp(−(Δ1 + Δ2)/T);
    fin
sinon
    Rejeter le changement avec une probabilité 1 − exp(−Δ1/T);
    si Le changement n'a pas été rejeté alors
        Calculer Δ2 = Δf2 − Tμ;
        si Δ2 < 0 alors
            Accepter le changement;
        sinon
            Accepter le changement avec une probabilité exp(−Δ2/T);
        fin
    fin
fin
```

Tout d'abord, remarquons qu'indépendemment de la valeur de μ, on a $\Delta_1 + \Delta_2 = \Delta_f$.

Ainsi, si $\Delta_1 < 0$, le critère standard est appliqué. Cela a d'autant plus de chance de se produire que μ est petit (négatif notamment). Pour une valeur suffisamment faible de μ (qui dépend des valeurs que Δ_1 et T peuvent prendre), on retrouve donc le critère standard. L'ajout du paramètre μ est donc une généralisation de l'algorithme du recuit simulé.

Si $\Delta_1 > 0$, on a alors une probabilité $1 - \exp(-\Delta_1/T)$ de rejeter le changement. C'est ici que l'on économise des appels à f_2. Lorsque le changement n'est pas rejeté, il est accepté avec une probabilité $\exp(-\Delta_2/T)$. Si l'on remarque que $\exp(-\Delta_1/T)\exp(-\Delta_2/T) = \exp(-\Delta_f/T)$, on a en supposant que Δ_1 et Δ_2 sont indépendants le récapitulatif suivant (table 5.6).

La seule différence avec le critère classique réside donc dans le cas où $\Delta_1 \geq 0$ et $\Delta_2 < 0$, qui

	$\Delta_1 < 0$		$\Delta_1 \geq 0$	
	$\Delta_f < 0$	$\Delta_f \geq 0$	$\Delta_2 < 0$	$\Delta_2 \geq 0$
Probabilité d'acceptation	1	$\exp(-\Delta_f/T)$	$\exp(-\Delta_1/T)$	$\exp(-\Delta_f/T)$

TABLE 5.6 – Probabilité d'accepter un changement en fonction des valeurs de Δ_1 et Δ_f

donnerait une probabilité de 1 ou de $\exp(-\Delta_f/T)$ dans le cas classique. Une valeur plus élevée pour μ augmente les chances de se retrouver dans ce cas de figure en augmentant Δ_1 et en diminuant Δ_2, et augmente également l'écart avec le critère classique lorsque ce cas de figure se produit en diminuant $\exp(-\Delta_1/T)$. Inversement, une valeur plus faible pour μ rapproche le comportement de ce critère de celui du critère classique. μ est donc un paramètre permettant de régler le compromis entre la qualité du recuit, et nombre d'appels à f_2.

5.7.2 Expériences numériques

Comme expliqué, la finalité de cet algorithme est d'être utilisé avec pour f_2 une fonction d'estimation du nombre de partants. Nous allons cependant utiliser dans cette section une fonction « jouet » pour tester l'intérêt de l'algorithme.

La première raison est que les méthodes prédictives présentées dans le chapitre 3 ne sont pas complètement satisfaisantes. La principale difficulté que nous avons rencontrée est le manque de diversité dans les données disponibles. Nous avons réussi dans une certaine mesure à comprendre quels sont les types de courses plus ou moins attractifs. Cependant, comme toutes les données sur le passé sur lesquelles on se base découlent de programmes faits à la main suivant une méthode fixe, nous n'avons pas réussi à recueillir beaucoup d'informations nous aidant à savoir comment planifier une course pour qu'elle soit la plus attractive possible. Dans la section 3.4.3, nous avons donné des pistes possibles pour améliorer l'algorithme de simulation de la section 3.4 et tenter de pallier le problème, mais il est nécessaire de poursuivre les travaux dans cette direction pour obtenir une fonction intéressante à tester sérieusement.

La seconde raison est que la simulation prend 350 ms à exécuter, et que nous pourrons faire bien plus de tests avec une fonction plus rapide à calculer.

Choisir une fonction f_2 jouet est donc intéressant pour faire plus de tests sur l'algorithme, sans que les résultats obtenus soient significativement moins pertinents.

Dans toute la suite de cette section, la fonction f_1 est toujours la fonction d'évaluation de la section 5.2, mais la fonction f_2 est une évaluation de la contrainte d'écart entre les courses $C2$ et les courses $C1$. La fonction f_2 évalue donc si les courses $C2$ sont bien suivies de courses $C1$ environ trois semaines plus tard. Nous avons choisi cette fonction car elle reflète un élément de la structure du programme potentiellement pertinent, qui n'est pas évalué par f_1 qui ne nécessite pas l'introduction de nouveaux concepts, et qui est relativement rapide à évaluer.

Pour effectuer les tests, nous avons repris l'implémentation présentée en section 5.5, mais en remplaçant le critère d'acceptation standard par le critère ci-dessus. Nous avons appliqué le recuit au programme d'automne 2019, avec les paramètres $T_0 = 1000$, r =0,85 et $N_{\text{Max}} = 500$. Cela nous permet d'avoir un recuit relativement rapide tout en testant des valeurs variées pour la température. Nous avons exécuté l'algorithme 41 fois, pour chaque valeur de μ entre -2 et 2 avec un pas de 0,1.

Nous avons commencé par exécuter le recuit simulé standard avec l'objectif $f = f_1 + f_2$ pour avoir une base de comparaison. Sur 3 exécutions, nous avons obtenu en moyenne comme valeurs finales, $f = 130156$, $f_1 = 120292$ et $f_2 = 9864$. Nous ferons apparaître ces valeurs dans les figures présentées pour pouvoir comparer les performances du recuit modifié par rapport au recuit standard.

Tout d'abord nous nous intéressons aux valeurs obtenues pour les fonctions objectif f, f_1, et f_2, en fonction de μ. Ces valeurs sont tracées respectivement dans les figures 5.3, 5.4, et 5.5. Sur chacune des figures, la ligne horizontale représente la valeur obtenue avec le recuit classique. Les points rouges sont les résultats obtenus avec chaque exécution du recuit modifié. Nous avons également représenté d'une courbe bleue, dans ces graphiques et dans les suivants, une moyenne glissante sur 5 valeurs de μ, pour plus de lisibilité. C'est une courbe qui donne la tendance d'évolution des résultats en fonction de μ.

On remarque tout d'abord que les différents objectifs sont proches des valeurs obtenues avec le recuit classique. Choisir une valeur de μ négative ne semble quasiment pas impacter la qualité des solutions obtenues. Pour $\mu > 0$ l'objectif f se dégrade progressivement. Cette dégradation est due à parts environ égales à une dégradation de f_1 et une dégradation de f_2. Cependant, puisque f_2 prend des valeurs plus faibles que f_1, on observe sur la figure 5.5 une dégradation bien plus prononcée et significative que sur la figure 5.4. Cette observation pourrait s'expliquer du fait que les changements refusés par le recuit modifié que le recuit standard aurait acceptés sont des changements avec $\Delta_1 \geq 0$ et $\Delta_2 < 0$.

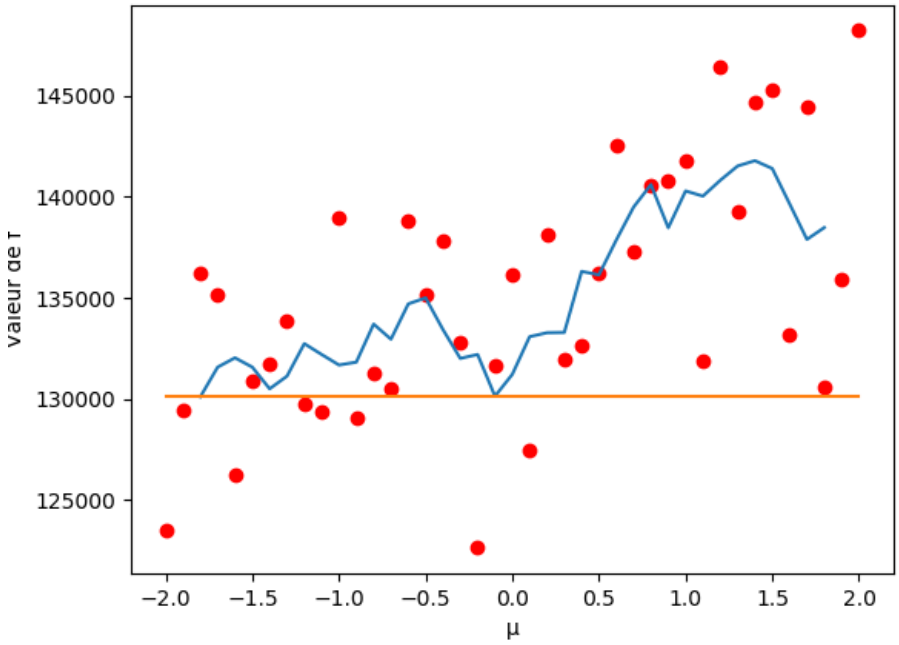


FIGURE 5.3 – Objectif f en fonction de μ

Pour évaluer les performances de l'algorithme, nous avons également mesuré la quantité d'appels à f_2 évités en fonction de μ, ce que nous avons représenté dans la figure 5.6. On lit que pour $\mu = 0$, nous avons évité 85% des appels à f_2, c'est-à-dire que f_2 n'a été calculé que dans 15% des itérations. On lit également que beaucoup plus d'appels sont évités si $\mu > 0$, sans qu'il ne soit vraiment clair pourquoi.

Pour mieux se représenter ce que signifient ces appels évités, nous avons calculé le temps qu'aurait pris le recuit simulé modifié si calculer la valeur de f_2 prenait 350 ms. Nous avons reporté ces temps dans la figure 5.7. On lit que pour un choix de $\mu = 0$, on aurait un temps de calcul d'environ 6000 s. À titre de comparaison, le recuit standard avec l'hypothèse qu'un appel à f_2 prendrait 350 ms, prendrait 50000 s. Ainsi, au vu des données précédentes, un choix de $\mu = 0$ permettrait de diviser le temps par 8 par rapport à un recuit classique en appelant f_2 à seulement 15% des itération, sans dégrader significativement la qualité des solutions. Ce temps libéré pourrait être utilisé pour générer plus de programmes, ou si nécessaire pour appeler f_2 plusieurs fois par itération. En effet, une fonction prédictive telle que la simulation présentée en section 3.4 n'est pas déterministe, et l'évaluer plusieurs

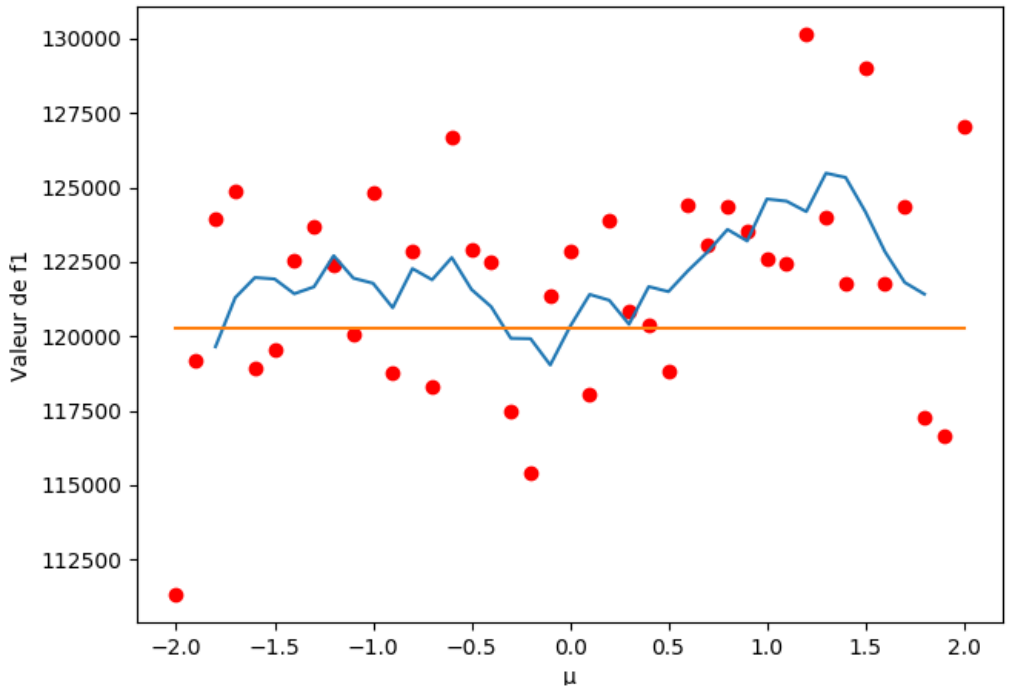


FIGURE 5.4 – Objectif f_1 en fonction de μ

fois permettrait de réduire sa variance, dans le cas où le recuit se comporterait mal avec une fonction objectif bruitée.

Un choix de μ négatif ne semble pas intéressant, puisque $\mu = 0$ est suffisamment performant, mais le choix d'un μ positif peut être pertinent suivant les applications. Si choisir $\mu = 2$ permet de diviser le temps par trois par rapport à $\mu = 0$, et que la dégradation de l'objectif reste raisonnable, comme le suggère nos essais, cela peut tout à fait être un compromis envisageable si le temps de calcul est aussi critique que la qualité des solutions.

Finalement, nous avons aussi mesuré le nombre de fois que le recuit classique aurait arbitré un changement différemment du recuit modifié. Autrement dit, nous avons compté le nombre d'itérations dans lesquelles le calcul de f_2 a été évité, mais qu'un calcul de f_2 révèle que $\Delta_f < 0$. Nous avons reporté dans la figure 5.8 la proportion d'itérations dans lesquelles cette « erreur » d'arbitrage se produit. On observe logiquement que cette proportion est croissante avec μ, en revanche, cette proportion reste très faible. De plus, nous n'observons pas, comme nous aurions pu nous y attendre, une grande différence

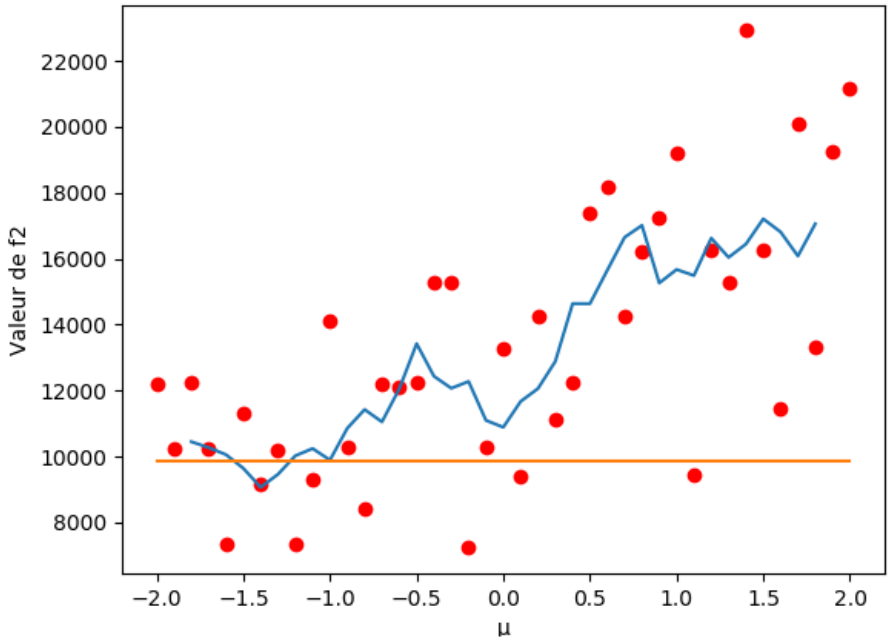


FIGURE 5.5 – Objectif f_2 en fonction de μ

entre $\mu = 0$ et $\mu > 0$. Elle est d'environ 0,15% pour $\mu = 0$, et d'environ 0,2% pour $\mu = 2$. Il n'est donc pas clair que la qualité des solutions obtenues et cette proportion soient intimement liées, mais parait surprenant qu'éviter de calculer f_2 98% du temps ne produise une différence d'arbitrage dans seulement 0,2% des itérations. En réalité, cette quantité serait à mettre en perspective avec le nombre de changements acceptés. Si le recuit modifié accepte 1% des changements, cela signifie que le recuit standard aurait accepté 20% de changements en plus que le recuit modifié, ce qui tout de suite n'est plus si négligeable.

En conclusion, nous avons montré expérimentalement que dans le problème de planification des courses de galop, et pour les fonctions objectif f_1 et f_2 choisies, le recuit modifié permet de n'appeler f_2 que pour 15% des itérations, et pour un coût quasi-nul. Il serait certainement intéressant de tester le recuit modifié avec d'autre fonctions objectif, et éventuellement sur d'autres problèmes, pour confirmer ses performances. En particulier il pourrait être intéressant de tester des fonctions f_1 et f_2 dont les ordres de grandeur et les variations sont différentes. Il serait, en effet, logique que l'algorithme soit efficace si il est rare que $\Delta_1 \geq 0$ et $\Delta_f < 0$ (il est toujours possible de faire en sorte que ce soit

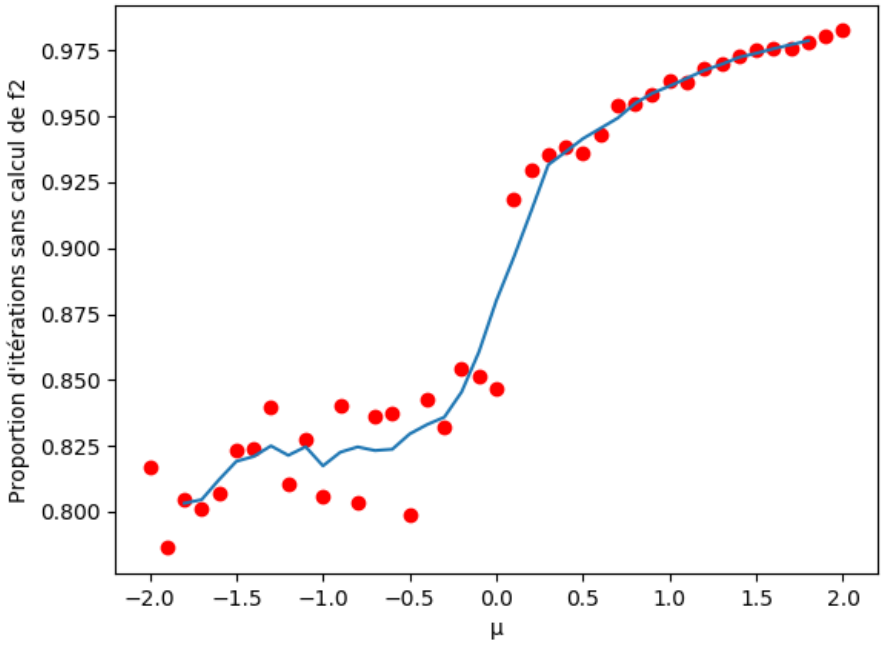


FIGURE 5.6 – Proportion d'appel à f_2 évités en fonction de μ

rare en prenant μ négatif ; ça pourrait être ainsi un critère de choix pour μ). Il est aussi probable que l'algorithme soit inefficace si f_1 et f_2 sont significativement corrélées. Nous avons aussi utilisé des recuits assez rapides, avec seulement $N_{\text{Max}} = 500$, mais il est possible que l'algorithme se comporte différemment lorsque l'on est plus proche des valeurs optimales, avec N_{Max} plus important. Enfin, à titre de remarque, il est tout à fait possible de généraliser la procédure à plus de fonctions objectif, où dans la même logique, on calculerait les différentes fonctions successivement.

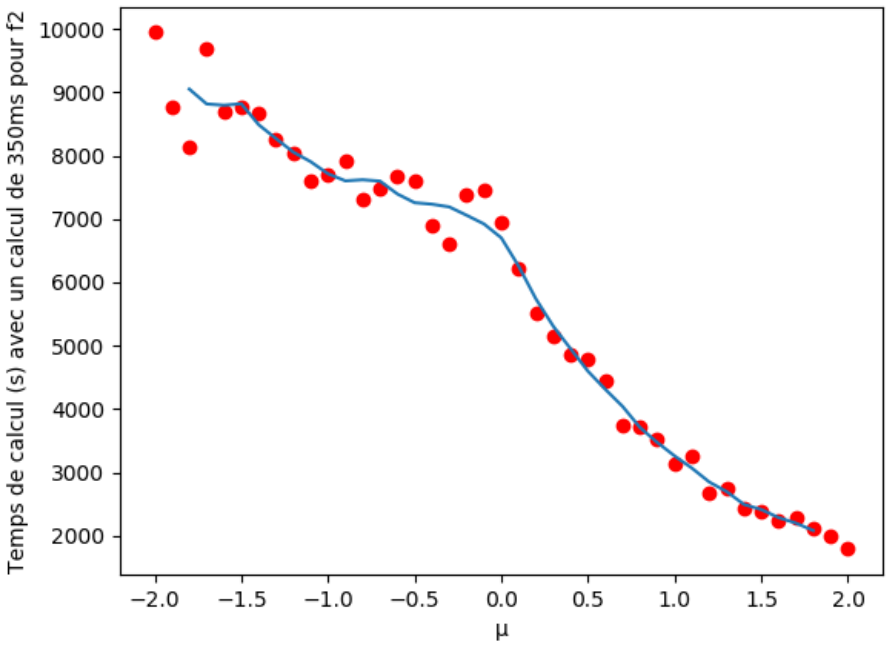


FIGURE 5.7 – Temps d'exécution en supposant un appel de 350ms pour f_2

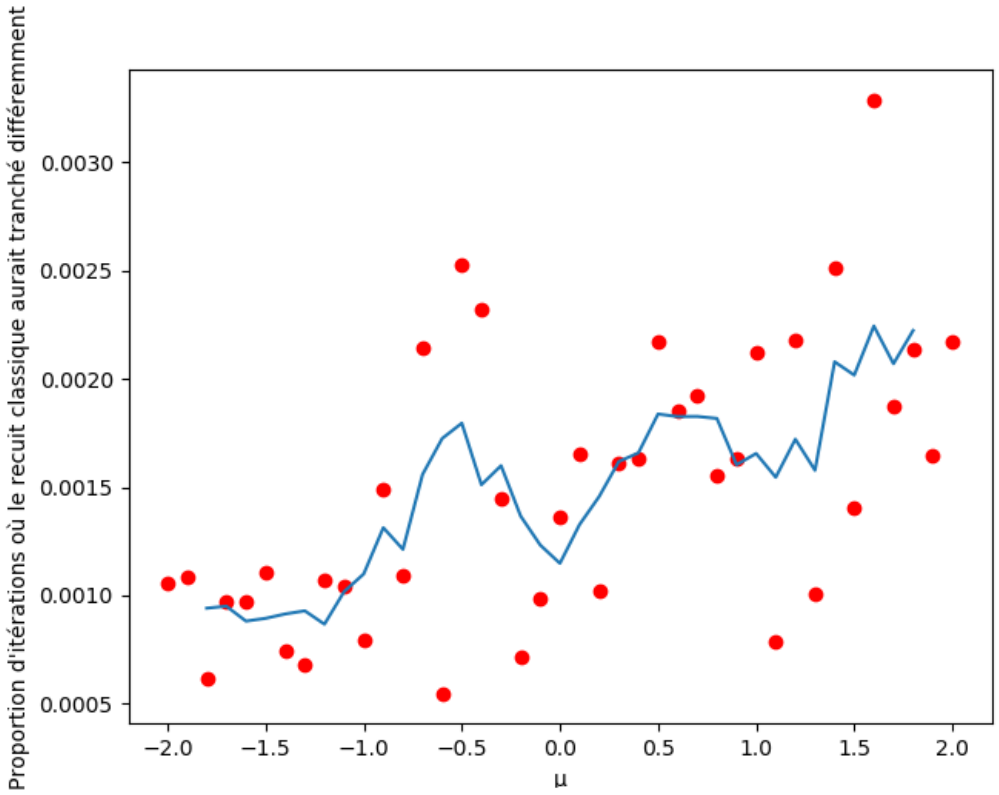


FIGURE 5.8 – Proportion des itérations qu'un recuit classique aurait arbitré différemment

Conclusion

Dans ce manuscrit, nous avons étudié le problème de la planification des courses de galop, pour France Galop. Ce problème se distingue des problèmes de la littérature par plusieurs aspects, comme vu au chapitre 2. Contrairement aux problèmes de planification sportive habituels, il ne s'agit pas de planifier des rencontres, puisque nous n'avons aucun contrôle sur la participation des chevaux aux courses. L'objectif est plutôt de trouver des solutions de planification dans lesquelles à tout moment un large éventail de types de courses est disponible. Nous souhaitons que les entraîneurs aient toujours des opportunités intéressantes pour faire participer des chevaux de tous profils et pour avoir autant de partants (de participants) que possible aux courses. Il est donc nécessaire que chaque type de course soit planifié régulièrement, ce qui rapproche le problème de planification des courses de galop des problèmes d'ordonnancement cyclique. On souhaite qu'après chaque course de type t, on en trouve une autre environ trois semaines plus tard ; il ne s'agit cependant pas, comme dans un ordonnancement cyclique, d'un délai minimum entre plusieurs planifications d'un même type de course, il peut s'en trouver d'autres entre temps. Plus généralement, nous n'avons pas une liste de courses (de tâches) précise à planifier, ce qui distingue notre problème de la majorité des problèmes d'ordonnancement. Il s'agit donc d'une thèse de nature exploratoire. Tout d'abord, nous avons dû modéliser les différents aspects du problème. Ensuite, nous avons recherché différentes méthodes pouvant être utiles pour résoudre ce nouveau problème de planification, nous les avons testées, et nous les avons comparées en étudiant leurs avantages relatifs.

Globalement, nous cherchons à mettre au point un outil informatique capable de produire des programmes (des calendriers) de courses de qualité. Cette problématique se décline en deux grands axes. Dans le chapitre 3, nous nous sommes intéressés à des méthodes d'apprentissage pour tenter d'évaluer précisément la qualité d'un programme de courses. Nous avons ensuite étudié des méthodes exactes et approchées permettant de produire des programmes de courses respectant les contraintes de la planification, respectivement dans les chapitres 4 et 5.

Une des mesures de la qualité d'un programme de courses, est de mesurer a posteriori le nombre de partants aux courses. Dans le chapitre 3, nous avons recherché des techniques d'apprentissage statistique qui nous permettraient d'utiliser les données historiques de participation pour prédire le nombre de partants de programmes futurs. Dans la section 3.3, nous avons notamment utilisé la régression logistique et le gradient boosting. Nous avons réussi à obtenir des prédictions raisonnables pour les nombres de partants, mais ces méthodes ont surtout permis de déterminer quels sont les types

de courses qui attirent en général le plus de partants. C'est intéressant d'un point de vue analyse des données, mais nous aimerions obtenir une fonction de prédiction qui puisse être utilisée pour produire de meilleurs programmes, et qui porte donc de l'information sur la façon de planifier les différents types de courses pour qu'ils soient les plus attractifs possibles. La raison principale pour laquelle il est très difficile de faire dépendre les résultats de la prédiction de la structure du programme a trait à la nature des données disponibles. Nous n'avons pas d'exemple dans nos données de courses qui attirent peu de partants à cause d'une mauvaise planification, puisque France Galop n'utilise que des programmes suffisamment bons. Il est donc difficile de déduire des données la différence entre une bonne planification et une mauvaise planification. Pour tenter de pallier ce problème, nous avons écrit et proposé en section 3.4 un algorithme permettant de simuler la participation d'une population de chevaux dans un programme de courses. Pour régler les nombreux paramètres du modèle de participation proposé, nous avons utilisé l'algorithme SPSA. Nous avons réussi grâce à l'algorithme SPSA à obtenir des paramètres produisant des simulations prédisant des nombres de partants bien plus proches de la réalité qu'avec les paramètres initiaux, ce qui montre que le SPSA est adapté au problème du choix des paramètres. En revanche nous rencontrons la même difficulté que précédemment, c'est-à-dire que les simulations produites ne sont pas très sensibles à la structure du programme de courses donné en entrée. Nous avons proposé en section 3.4.3 plusieurs pistes pouvant permettre à moyen terme de pallier ce problème. Outre de mineurs ajustements au modèle de participation, un avantage de cette approche par simulation est qu'il est possible d'interdire aux chevaux les parcours qui sont empiriquement irréalistes pour que leur participation soit plus contrainte et soit mécaniquement fortement dépendante de la structure du programme. Il nous semble donc possible d'améliorer encore cette méthode de prédiction de la participation et d'obtenir une fonction qui pourra être utile pour guider la planification. Nous avons mis au point et proposé en section 5.7 une variante de l'algorithme du recuit simulé pouvant utiliser efficacement une telle fonction.

Dans le chapitre 4, nous avons étudié la production de programmes de courses respectant les contraintes de planification à l'aide d'un modèle mathématique linéaire en variables 0-1. Ces contraintes nous ont été données par les experts qui planifient aujourd'hui les courses de galop à la main. Leur expression n'est ni complètement claire, ni définitive. Nous avons proposé plusieurs formulations du problème, qui modélisent les contraintes souples à l'aide de variables de pénalisations, et nous avons démontré que les problèmes de décision qui en découlent sont NP-complets. Il a tout de même été

possible de trouver des solutions exactes pour certains modèles, et d'obtenir de bonnes solutions entières dans les autres cas. Nous avons également montré qu'il était possible d'utiliser les résultats de la partie apprentissage, notamment du gradient boosting, pour décider des coefficients de la fonction objectif. Il est cependant très difficile de modéliser le problème sous forme d'un programme linéaire fidèle à la réalité. D'une part, la satisfaction des contraintes souples est en réalité souvent plus nuancée que le « respect » ou le « non respect » de ces dernières. D'autre part, l'utilisation de coefficients issus de l'apprentissage dans une fonction objectif linéaire produit des résultats peu exploitables en pratique. Enfin, la planification des courses de galop est un procédé en constante évolution, et on ne peut pas vraiment prévoir si le modèle se comportera bien avec des modalités de planification différentes. En résumé, malgré l'obtention de solutions entières à l'aide de la programmation linéaire, nous ne pensons pas que ce soit la méthode la plus adaptée pour répondre aux problématiques industrielles de France Galop, ou qu'il soit intéressant de dépenser du temps de calcul pour prouver l'optimalité de solutions par rapport à un objectif bien plus qualitatif que quantitatif.

Dans le chapitre 5, nous avons donc étudié différentes méthodes approchées pour produire des programmes de courses. Nous proposons une fonction d'évaluation des programmes se basant sur l'évaluation du respect des contraintes souples. La description du calcul de cette fonction objectif passe par une modélisation précise des différentes contraintes souples prises en compte, et des pénalités associées à leur violation. La première méthode approchée proposée est une heuristique gloutonne, qui s'inspire du processus manuel de planification, l'idée étant d'obtenir rapidement des solutions ayant une structure similaire aux programmes utilisés par France Galop, ce qui nous a beaucoup aidé pour discuter des contraintes de la planification avec les experts. Certaines contraintes souples prises en compte dans la fonction d'évaluation ne peuvent cependant pas être considérées par une heuristique gloutonne, nous proposons donc aussi deux méta-heuristiques, et nous comparons leurs performances à celles de l'heuristique. La première méta-heuristique utilisée est un recuit simulé classique, son originalité résidant dans un système de voisinages adapté au problème de planification. La seconde est une RVNS, version dite Réduite de la VNS, utilisant sensiblement les mêmes voisinages. Après avoir expliqué le choix des paramètres du recuit, nous présentons des résultats numériques obtenus sur des instances réelles et sur des instances générées artificiellement. Nous obtenons avec le recuit des solutions de très bonne qualité vis-à-vis de l'évaluation utilisée, meilleures que celles obtenues avec l'heuristique gloutonne, même en évaluant uniquement le respect des contraintes considérées par

l'heuristique. En revanche, les résultats obtenus avec la RVNS sont médiocres et suggèrent que les voisinages proposés ne sont pas suffisamment sophistiqués pour permettre à la RNVS de sortir des optima locaux. L'utilisation de nouveaux voisinages pourrait donc être une piste d'amélioration pour cet algorithme. Enfin, dans la section 5.7, nous avons proposé et étudié une variante de l'algorithme du recuit simulé qui permettrait d'utiliser les fonctions prédictives du chapitre 3 pour guider la recherche de solutions plus attractives. Le principal obstacle à l'utilisation de la simulation dans un recuit simulé est le temps de calcul. Le calcul d'une simulation prend 200 fois plus de temps que le calcul de la fonction d'évaluation utilisée dans les deux méta-heuristiques. L'algorithme proposé permet d'utiliser le recuit simulé pour minimiser une somme $f_1 + f_2$ de fonctions d'évaluation en calculant la valeur de f_2 un minimum de fois. Pour nos tests, nous avons utilisé notre fonction d'évaluation pour f_1, et, pour des raisons pratiques, une fonction d'évaluation « jouet » pour f_2 (basée sur le respect d'une nouvelle contrainte souple). Nous parvenons à économiser 85% des appels à la fonction f_2 par rapport à un recuit standard, sans compromettre la qualité des solutions obtenues.

En conclusion, nous avons dans cette thèse proposé plusieurs méthodes permettant la génération de programmes de courses respectant au mieux un ensemble de contraintes dures et souples. Les méta-heuristiques, notamment, ont l'avantage de pouvoir modéliser les contraintes finement et aussi fidèlement à la réalité que possible, et peuvent s'accommoder des inévitables évolutions des modalités de planification des courses de galop. Nous souhaitons aussi, parallèlement au respect des contraintes, produire des programmes attractifs pour les entraîneurs et qui favorisent la participation. Nous avons rencontré plus de difficultés sur cet aspect du sujet, notamment en raison de la qualité des données disponibles portant sur les courses passées : nous avons besoin de données portant sur des mauvaises planifications pour discerner les bonnes planifications des mauvaises. L'algorithme de simulation proposé en section 3.4 tente de compenser les difficultés d'exploitation des données par une modélisation du mécanisme de participation des chevaux aux courses. Nous avons obtenu des résultats préliminaires encourageants, et nous avons proposé des pistes d'amélioration. Plus précisément, nous pensons affiner le modèle de participation, et nous espérons que ces nouveaux modèles pourront être utilisés pour guider les méta-heuristiques vers des solutions plus attractives. La variante de l'algorithme du recuit simulé proposé en section 5.7 est un grand pas dans cette direction, puisqu'elle permettrait d'intégrer les résultats de la simulation à la recherche efficace de solutions, sans avoir à évaluer systématiquement la simulation coûteuse. Nous aimerions poursuivre l'étude de cette variante du recuit simulé en la tes-

tant sur d'autres fonctions objectifs, et éventuellement avec plus que deux composantes dans l'objectif. Nous pensons que cet algorithme présente un intérêt théorique pour optimiser le temps de calcul des recuits simulés, de façon particulièrement efficace notamment lorsqu'une composante de l'évaluation est coûteuse à calculer mais est une constituante secondaire de l'évaluation.

Bibliographie

[1] F. Galop, “Code des courses et conditions générales,” http ://www.france-galop.com/fr/node/77.

[2] G. Kendall *et al.*, “Scheduling in sports : An annotated bibliography,” *Comput. Oper. Res.*, vol. 37, n°. 1, p. 1–19, 2010. [En ligne]. Disponible : https://doi.org/10.1016/j.cor.2009.05.013

[3] F. D. Croce et D. Oliveri, “Scheduling the italian football league : an ilp-based approach,” *Comput. Oper. Res.*, vol. 33, p. 1963–1974, 2006. [En ligne]. Disponible : https://doi.org/10.1016/j.cor.2004.09.025

[4] T. Kim, “Optimal approach to game scheduling of multiple round-robin tournament : Korea professional baseball league in focus,” *Comput. Ind. Eng.*, vol. 136, p. 95–105, 2019. [En ligne]. Disponible : https://doi.org/10.1016/j.cie.2019.07.016

[5] Y. Tian *et al.*, “Stochastic simulation optimization for route selection strategy based on flight delay cost,” *Asia Pac. J. Oper. Res.*, vol. 35, n°. 6, p. 1850045, 2018. [En ligne]. Disponible : https://doi.org/10.1142/S0217595918500458

[6] C. Chen et L. H. Lee, *Stochastic Simulation Optimization - An Optimal Computing Budget Allocation*, ser. System Engineering and Operations Research. WorldScientific, 2010, vol. 1. [En ligne]. Disponible : https://doi.org/10.1142/7437

[7] P. Chrétienne, “The basic cyclic scheduling problem with deadlines,” *Discret. Appl. Math.*, vol. 30, n°. 2-3, p. 109–123, 1991. [En ligne]. Disponible : https://doi.org/10.1016/0166-218X(91)90037-W

[8] ——, “On graham’s bound for cyclic scheduling,” *Parallel Comput.*, vol. 26, n°. 9, p. 1163–1174, 2000. [En ligne]. Disponible : https://doi.org/10.1016/S0167-8191(00)00033-8

[9] A. M. Kordon, “A graph-based analysis of the cyclic scheduling problem with time constraints : schedulability and periodicity of the earliest schedule,” *J. Sched.*, vol. 14, n°. 1, p. 103–117, 2011. [En ligne]. Disponible : https://doi.org/10.1007/s10951-009-0159-z

[10] C. Hanen, “Study of a np-hard cyclic scheduling problem : The recurrent job-shop,” *EJOR*, 1994.

[11] P. Matrenin et V. Manusov, “The cyclic job-shop scheduling problem : The new subclass of the job-shop problem and applying the simulated annealing to solve it,” *CoRR*, vol. abs/2006.10938, 2020. [En ligne]. Disponible : https://arxiv.org/abs/2006.10938

[12] J. Friedman, “Greedy function approximation : a gradient boosting machine,” dans *IMS 1999 Reizh Lecture*, 1999.

[13] A. Tikhonov, “Solution of incorrectly formulated problems and the regularization method,” *Soviet Mathematics*, vol. 4, 1963.

[14] R.Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society*, vol. 58, 1996.

[15] J. Spall, “Multivariate stochastic approximation using a simultaneous perturbation gradient approximation,” *IEEE T. AUTOMAT. CONTR.*, vol. 37, n°. 3, p. 332–341, 1992.

[16] L. Kocsis et C. Szepesvári, “Universal parameter optimisation in games based on SPSA,” *Mach. Learn.*, vol. 63, n°. 3, p. 249–286, 2006. [En ligne]. Disponible : https://doi.org/10.1007/s10994-006-6888-8

[17] J. Spall, “Implementation of the simultaneous perturbation algorithm for stochastic optimization,” *IEEE T. AERO. ELEC. SYS.*, vol. 34, n°. 3, 1998.

[18] A. S. Guyslain Naves, “Multiflow feasibility : an annotated tableau.” *Research Trends in Combinatorial Optimization.*, 2008.

[19] E. H. N. Mladenovic, “Variable neighborhood search,” *Computer. Ops Res*, 1997.

[20] S. Kirkpatrick, C. Gelatt et M. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, n°. 4598, May 1983.

[21] W. Ben-Ameur, “Computing the initial temperature of simulated annealing,” *Comput. Optim. Appl.*, vol. 29, n°. 3, p. 369–385, 2004. [En ligne]. Disponible : https://doi.org/10.1023/B:COAP.0000044187.23143.bd

[22] M. Luby, A. Sinclair et D. Zuckerman, “Optimal speedup of las vegas algorithms,” *Inf. Process. Lett.*, vol. 47, n°. 4, p. 173–180, 1993. [En ligne]. Disponible : https://doi.org/10.1016/0020-0190(93)90029-9

Annexe A : Compléments statistiques

Modèle de la régression logistique

Nous présentons ici le modèle mathématique derrière la régression logistique, présentée en section3.3. Présentons d'abord le cas ou l'on souhaiterais simplement faire une prédiction entre 2 classes. Soit $\mathcal{B}$ l'ensemble des courses possibles, et K l'ensemble des classes. On note $x \in \mathcal{B}$ un vecteur de caractéristiques $x = (x_1, \ldots, x_m)$. On note $y \in K$ la classe à laquelle x appartient. Si $|K| = 2$ (cas binomial), on a donc $y = 0$ ou $y = 1$. On veut prédire quelle est la probabilité que x soit dans la classe 1(i.e. $p(1|x)$). L'hypothèse de la régression logistique est la suivante

$$\log \frac{p(1|x)}{p(0|x)} = a_0 + a_1 x_1 + \cdots + a_m x_m$$

et le but de l'apprentissage est de trouver les $a_0, \ldots, a_m$ adéquats. a_1 représente a quel point x_1 est un critère en faveur de la classe 1. $a_1 = 0$ voudrais dire que x_1 est absolument sans importance. Ce modèle est relativement simple, il s'agit d'une régression linéaire généralisée, puisqu'on fait essentiellement une combinaison linéaire des différentes caractéristiques.

Pour faire une prédiction, il suffit de renverser l'équation en remarquant que $p(0|x) = 1 - p(1|x)$ dans le cas binomial :

$$p(1|x) = \frac{\exp(a_0 + a_1 x_1 + \cdots + a_k x_k)}{1 + \exp(a_0 + a_1 x_1 + \cdots + a_k x_k)}$$

Dans le cas multinomial, on a un jeu de coefficients a^k pour chacune des classes $k \in K$, de sorte que

$$p(k|x) = \frac{\exp(a_0^k + a_1^k x_1 + \cdots + a_m^k x_m)}{\sum\limits_{i \in \mathcal{K}} \exp(a_0^i + a_1^i x_1 + \cdots + a_m^i x_m)}$$

De même, le coefficient a_m^k représente à quel point la caractéristique x_m pèse en faveur de la classe

k.

Algorithme du gradient boosting

Nous présentons ici les grandes lignes de l'algorithme du gradient boosting, utilisé en section 3.3.5. Nous avons dans notre problème pour chaque course i un vecteur x_i qui contient les caractéristiques de la course, et y_i le nombre de partants réel de la course. Le postulat de base est que l'on a une fonction L qui mesure l'écart entre le nombre de partants prédit et le nombre de partants réel. On cherche donc une fonction F qui prédit pour chaque course un nombre de partants, et qui minimise $\sum_i L(F(x_i), y_i)$.

Le gradient boosting est une méthode qui va construire la fonction F de façon gloutonne, en sommant des fonctions de prédiction faibles. L'algorithme est le suivant :

Algorithme 6 : Algorithme du gradient boosting

Initialiser le modèle avec une valeur constante $F_0(x) = \gamma$;
pour $m \leftarrow 1$ **à** M **faire**
  Pour tout i, on calcule le résidu $r_{im} \leftarrow -\frac{\partial L(F_{m-1}(x_i), y_i)}{\partial F_{m-1}(x_i)}$;
  On détermine une fonction de prédiction faible h_m entraînée sur l'ensemble $\{x_i, r_{im}\}_i$;
  On choisit un coefficient γ_m pour mettre à jour F, $F_m \leftarrow F_{m-1} + \gamma_m h_m$

Les γ et γ_m sont choisis de sorte à minimiser l'erreur de prédiction. Ainsi, le principe de l'algorithme est donc, à chaque itération, de déterminer une fonction de prédiction faible h_m qui prédit l'erreur que fait F_{m-1} dans sa prédiction, et de l'utiliser pour obtenir F_m en corrigeant F_{m-1}. Si, par exemple, on veut minimiser l'erreur quadratique, c'est-à-dire, $L(F(x), y) = (F(x) - y)^2$, on a tout simplement $r_{im} = 2(y_i - F_{m-1}(x_i))$, et h_m est donc une fonction qui estime la différence entre les y_i et les $F_{m-1}(x_i)$.

Dans notre application, les h_m sont des arbres de décision peu profonds (de profondeur 3), ce qui est très classique, mais l'application de l'algorithme ne dépend a priori pas de la façon dont les h_m sont calculés. Il est cependant important de prendre des fonctions de prédiction faibles, c'est-à-dire qui ne prédisent pas trop bien, pour ne pas faire de sur-apprentissage.

Statistiques complémentaires

Nous présentons dans cette section quelques statistiques intéressantes que nous avons faites, mais qui n'ont pas trouvé leur place dans le corps du manuscrit.

Statistiques de transitions

Une statistique très intéressante concerne les parcours des chevaux dans le programme. Nous avons regardé pour chaque catégorie de course, vers quelles catégories de courses se dirigent ensuite les chevaux qui y participent. Les résultats sont présentés dans la figure 9. La coloration représente la probabilité de faire chaque transition. Par exemple, dans la ligne intitulée « Inédits », seule la case correspondant à la colonne « Maiden » est très colorée. Cela signifie qu'après avoir couru une course d'Inédits la très grande majorité des chevaux se dirigent vers une course « Maiden ». Inversement, rien n'est coloré dans la colonne « Inédits », ce qui montre bien que l'on ne se dirige jamais vers une course d'Inédits après avoir couru une autre course, ce qui est attendu comme les courses d'Inédits sont réservées aux chevaux n'ayant jamais couru. La présence de nombreuses cellules colorées dans la diagonale justifie l'application de la contrainte d'écart. Rien n'est coloré dans les lignes et colonnes des catégories C3 et C4 car nous avons fait la statistique sur des programmes antérieurs à l'instauration de ces catégories. C'est aussi la raison de la présence du label « Condition », qui désigne des catégories ayant disparu avec la nomenclature moderne.

Nous avons aussi fait la même statistique en regardant cette fois ci les transitions en termes de distances et non en termes de catégories. Les résultats sont dans la figure 10. On voit ici aussi que les cellules proches de la diagonale sont plus colorées, ce qui indique que les chevaux ont des distances préférentielles. Nous avons aussi fait ces statistiques en se restreignant à certains âges, ou en regardant d'où viennent les chevaux plutôt qu'où ils vont.

Ces statistiques sont utilisables pour fixer empiriquement les coefficients de transitions π de la simulation dans la section 3.4. Nous avons essayé de le faire, mais le modèle de participation n'est pas suffisamment fidèle à la réalité pour obtenir de bons résultats avec. Il sera cependant possible d'utiliser ces statistiques pour déterminer empiriquement une liste de transitions possibles, ce que l'on propose dans une des pistes d'amélioration.

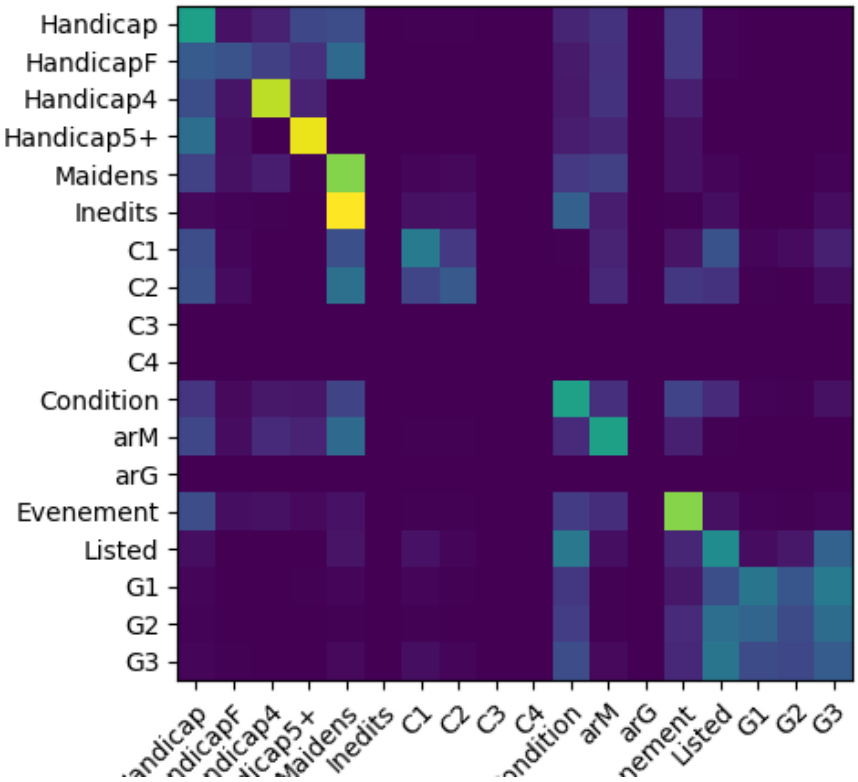


FIGURE 9 – Statistiques de transitions de catégories

Statistiques de déplacement

Nous avons fait une statistique pour mesurer le déplacement des chevaux. Le territoire français est divisé en fédérations. Nous avons mesuré où les chevaux courent en fonction de leur fédération d'entraînement, ce qui est représenté en figure 11.

On lit par exemple sur la quatrième ligne que 52% des chevaux entraînés en fédération 3 courent en fédération 0. On comprend bien par exemple en regardant la figure que la planification des courses dans les fédérations 5 ou 6 est complètement indépendante de la planification dans les autres fédérations. Dans cette thèse, nous avons cependant uniquement étudié la planification des courses planifiées à proximité de Paris, ce qui regroupe les fédérations 0, 3, 4 et 8. Nous n'avons donc pas eu l'utilité de cette statistique.

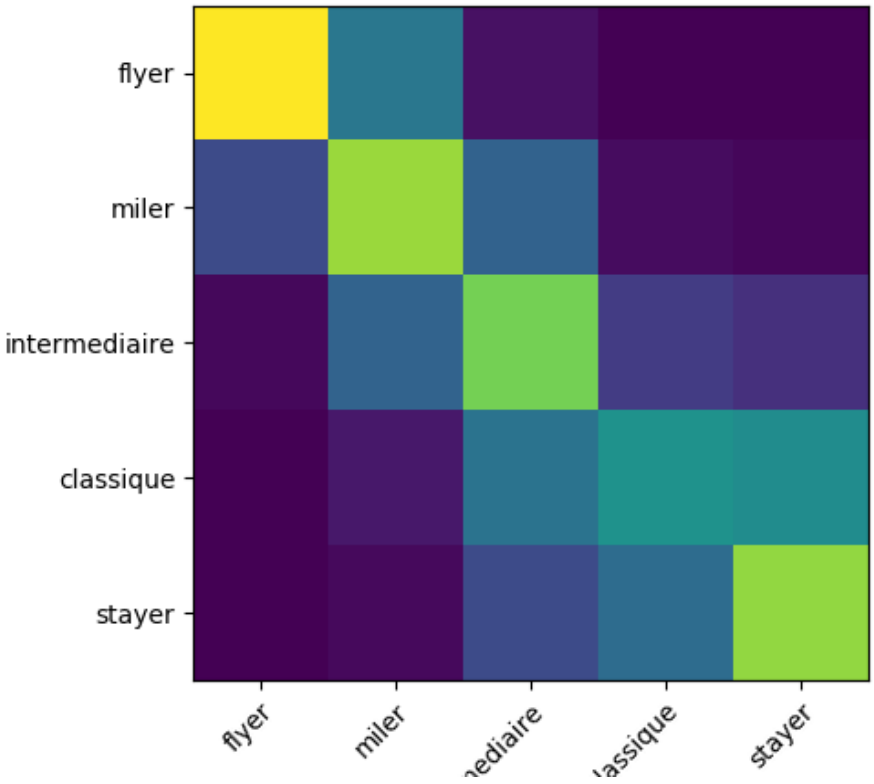


FIGURE 10 – Statistiques de transitions de distances

Statistiques sur les entraîneurs

Puisque nous souhaitions simuler la participation des chevaux, et que ce sont les entraîneurs qui les inscrivent aux courses, nous nous sommes intéressés aux données disponibles sur les entraîneurs. Nous n'avons finalement rien incorporé au modèle de participation relatif aux entraîneurs et ces statistiques n'ont donc pas été utilisées, mais elles sont très intéressantes et pourraient être utilisées dans une analyse plus poussée des données disponibles. Dans la figure 12, nous avons étudié la répartition des chevaux suivant les entraîneurs, pendant l'année 2014.

La courbe orange indique la proportion de chevaux entraînés par des entraîneurs en fonction du nombre de chevaux qu'ils entraînent. Si on regarde à 25 chevaux sur l'axe des abscisses, on lit environ 40% sur l'axe des ordonnées. Cela signifie que 40% des chevaux sont entraînés par des entraîneurs qui entraînent 25 chevaux ou moins.

FIGURE 11 – Statisique de déplacement des chevaux

La courbe bleu indique la proportion des entraîneurs par effectif de chevaux. Si on regarde toujours à 25 chevaux, on lit cette fois environ 90%. Cela signifie que 90% des entraîneurs entraînent moins de 25 chevaux.

La figure 13 représente la même chose sauf que, cette fois-ci, on s'est intéressés aux nombres d'inscriptions aux courses, au lieu de regarder les nombres de chevaux entraînés. On y lit, par exemple, sur la courbe orange que les entraîneurs comptabilisant moins de 200 inscriptions comptent pour environ 50% des inscriptions. En regardant la courbe bleue, on voit que cela représente plus de 90% des entraîneurs. Ces statistiques permettent de mesurer l'impact des petits et des gros entraîneurs dans les partants totaux aux courses.

Autrement dit, 10% des entraîneurs influencent plus de 50% des participations aux courses. Une tentative d'apprentissage sur leurs stratégies pour choisir les courses du programme auxquelles faire concourir leurs volumineux cheptels serait une piste intéressante pour la simulation.

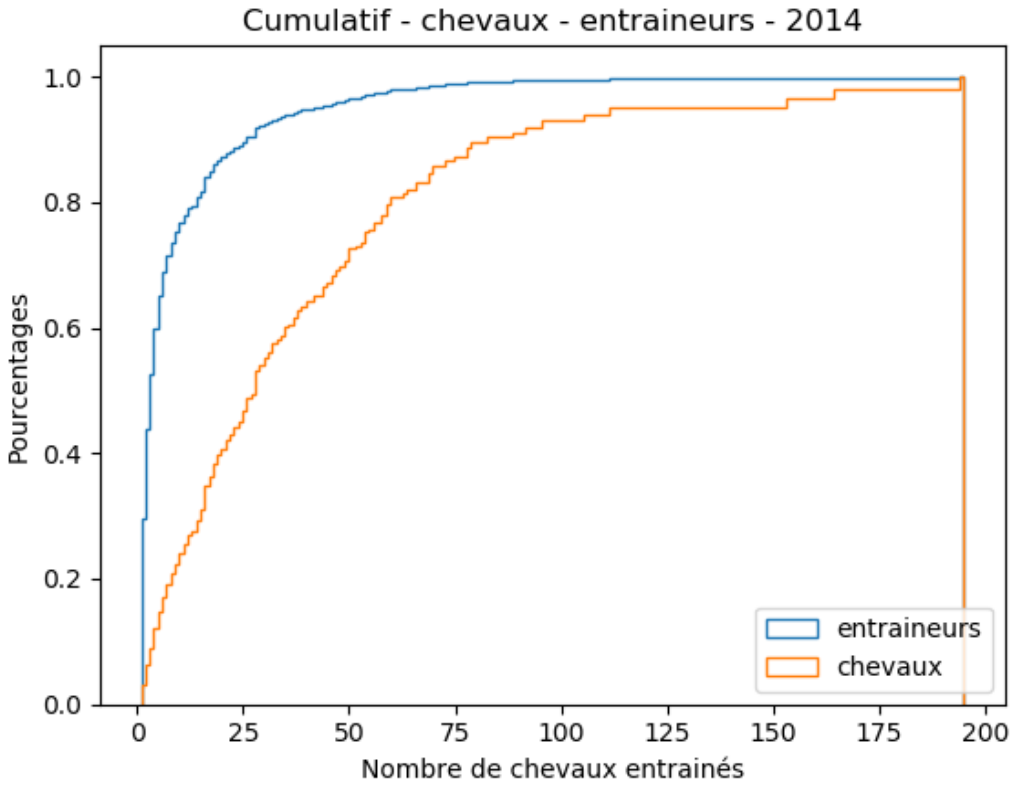


FIGURE 12 – Répartition des chevaux suivant les entraîneurs

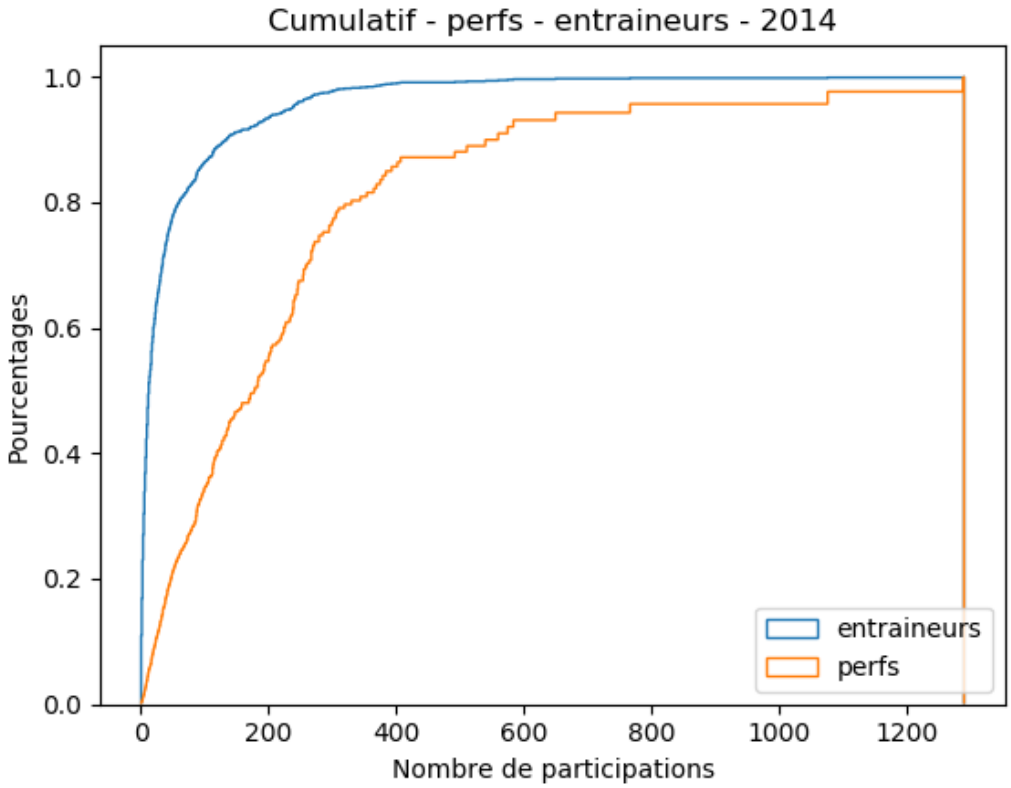


FIGURE 13 – Répartition des participations suivant les entraîneurs

Annexe B : Compléments PPC

Avant de travailler sur la résolution du problème à l'aide d'un modèle linéaire en variables 0-1, nous avions essayé d'utiliser la programmation par contraintes (PPC). Ce travail a été effectué en tout début de thèse, et l'énoncé du problème était différent de celui que nous avons présenté dans le manuscrit en plusieurs aspects. Principalement, toutes les contraintes de planification étaient dures, et nous étions donc sur un problème de satisfiabilité. Étant face à un problème de planification avec des contraintes dures, et pas d'objectif à optimiser, la PPC semblait être un outil adapté. Les changements apportés à la formulation du problème et les difficultés rencontrées nous ont ensuite amenés à abandonner cette piste, mais nous présentons tout de même ici les travaux réalisés.

Rappels sur la PPC La programmation par contraintes (PPC) est un ensemble de techniques de programmation, permettant la description et la résolution de problèmes de satisfaction de contraintes à variables entières.

La première partie de la modélisation est d'établir une liste de variables et leurs domaines associés. Ces domaines n'ont pas besoin d'être des intervalles, il s'agit juste d'une liste de valeurs. La seconde partie de la modélisation est l'expression des contraintes. Dans la forme la plus générale, ces contraintes sont exprimées en décrivant les valeurs admissibles pour des tuples de variables, ce qui peut se ramener à la description des valeurs admissibles pour des couples de variables, quitte à rajouter des variables. La PPC permet donc la modélisation de problèmes très variés, tant son langage de description est général.

Pour ce qui est de la résolution, elle est en deux parties. La première partie est une recherche arborescente dans laquelle on affecte successivement aux variables une valeur de leur domaine, en respectant les couples de valeurs admissibles. Lorsqu'on rencontre une impossibilité, on backtrack. L'autre partie est la propagation de contraintes. Le but de la propagation de contraintes est de réduire

la taille des domaines des variables en examinant la topologie du problème. Le niveau le plus bas de la propagation de contraintes est l'*arc-consistance*. Essentiellement, il s'agit de vérifier pour chaque contrainte, qu'à toutes les valeurs de la première variable de la contrainte il correspond au moins une valeur admissible de la seconde variable, et inversement. Le procédé peut se généraliser pour réaliser une propagation plus profonde, au prix d'un temps de calcul plus important.

Remarque : on peut voir un problème de programmation par contraintes comme un problème de graphe. Chaque sommet est une valeur associée à une variable, et chaque arrête un couple de variables/valeurs admissible. Une variable est donc représentée par un ensemble de sommet (de même taille que son domaine). Trouver une solution réalisable revient à trouver un sous-graphe complet avec autant de sommet que le problème a de variables.

Sans rentrer dans les détails, une bibliothèque de PPC permet de décrire un problème en utilisant des contraintes élémentaires telles qu'imposer l'égalité/inégalité entre deux variables, et permet de créer de nouvelles variables à partir d'anciennes avec des opérations arithmétiques et logiques simples. La bibliothèque est capable d'effectuer la propagation de contraintes de manière efficace sur ce modèle. La résolution est ensuite automatisée, en déterminant un bon compromis entre l'exploration de l'arborescence et la propagation de contraintes. Il est souvent possible de choisir parmi de nombreuses heuristiques pour le parcours d'arbre, ce qui permet à l'utilisateur de mettre à profit sa connaissance du problème.

Modélisation

Caractéristiques d'une instance

Lorsque le logiciel doit préparer le programme, plusieurs éléments sont déjà en place. Le calendrier des réunions est établi, on connaît la date et le lieu de chaque réunion. On connaît également le nombre de courses qu'il faut prévoir pour chaque réunion. On connaît également les courses fixées à la main par les experts, comme les courses à support d'événement, et les courses de groupe. Enfin, on connaît aussi le préprogramme, c'est-à-dire le programme du mois qui précède l'horizon de planification. Par exemple, si l'on veut élaborer le programme de septembre à novembre, on sait quelles étaient les courses d'août. Cette partie est cruciale car le nouveau programme doit respecter la contrainte d'écart avec le programme précédent.

Variables

On rappelle que la contrainte dite *de successeur* est définie comme suit : chaque course doit être suivie d'une course du même type située dans un intervalle de temps dépendant de la distance uniquement. Il y a cinq plages de distance, trois plages d'âge (2 ans, 3 ans et 4 ans et plus), et cinq catégories de courses principales (Handicap, Réclamer, Condition, Maiden et Inédits). On a donc 75 types au maximum, mais en réalité, nombre de ces combinaisons sont impossibles, comme une course de deux ans sur 2500 mètres. Dans cette première modélisation, on se concentre essentiellement sur cette contrainte-ci.

Dans un premier temps, nous avons laissé de côté les courses Maiden et Inédits, le nombre de types est donc temporairement de 45. On les numérote de 0 à 44.

Variables de course Dans un premier temps, on peut associer une variable x_t prenant les valeurs entières entre 0 et 44 à chaque créneau t pour lequel une course doit être planifiée. Typiquement, on aura de l'ordre d'un millier de ces variables (un millier de créneaux disponibles dans l'horizon de planification). On autorisera également une valeur arbitraire qu'on notera NON_ASSIGNED pour indiquer qu'un créneau est laissé vide, pour laisser plus de souplesse au modèle. On pourra également avoir des variables fixées à une valeur en dehors de l'intervalle pour indiquer les courses Listed ou Groupe. Ces variables sont pratiques pour décrire une solution. Leur lecture est intuitive. Cette modélisation a également l'avantage de permettre de restreindre le domaine efficacement.

Variables de successeur Le principal problème des variables de course est qu'elles ne sont pas vraiment adaptées pour décrire la contrainte de successeur. Il est possible de l'implémenter avec celles-ci, mais il est bien plus difficile d'avoir un arbre de recherche intéressant. En effet, avec de telles variables, fixer une variable n'affecte pas vraiment le domaine des autres variables. Il faut aller très profondément dans une branche pour savoir si elle est viable ou non.

Ainsi, on utilise un nouveau jeu de variables en plus des variables précédentes : des variables de successeur. Pour chaque créneau, on va avoir, en plus de la variable de course qui indique quelle course est programmée sur le créneau, une variable de successeur y_t qui indique quand a lieu le successeur de la course x_t

INDEX	Date	C_FED	N_ORD_HIPP	CATEG	AGE	DISTANCE	TEXTE	NEXT	DISTANCE EN JOURS
27	20160903	2	12				---	---	---
28	20160903	2	12	L	2	1650	2 L [1450,1800]	---	---
32	20160903	6	4	Condition-	4+	2150	4+ Condition- [1800,2150]	392	23

FIGURE 14 – Exemple de résultat obtenu avec la PPC (1)

INDEX	Date	C_FED	N_ORD_HIPP	CATEG	AGE	DISTANCE	TEXTE	NEXT	DISTANCE EN JOURS
391	20160925	9	3				---	---	---
392	20160926	0	5	Condition-	4+	2000	4+ Condition- [1800,2150]	706	20

FIGURE 15 – Exemple de résultats obtenu avec la PPC (2)

Dans la figure 14, on peut voir à quoi ressemble un résultat classique. On peut lire que pour la course d'indice 32, qui se déroule dans l'hippodrome 6-4 le 3 septembre, le logiciel a programmé une course à Condition pour des chevaux de 4 ans et plus, sur une distance de 2150 mètres. Cette combinaison de caractéristiques correspond en interne au type de course 33(ce n'est pas sur la figure), soit $x_{32} = 33$. On lit également dans la colonne NEXT que le successeur du même type est la course d'indice 392, située 23 jours plus tard, c'est la variable de successeur. Autrement dit, $y_{32} = 392$. Dans la figure 15, on voit que la course d'indice 392 est bien une course du même type (et on a donc $x_{32} = x_{y_{32}} = x_{392} = 33$) : une course à Condition pour chevaux de 4 ans et plus sur une distance entre 1800 et 2150 mètres (2000 mètres).

Les variables de successeur vont permettre d'implémenter les contraintes de manière plus simple et plus efficace. Fixer la valeur de x_{32} réduit le domaine de y_{32}, et fixer la valeur de y_{32} fixe la valeur de $x_{y_{32}}$. Il sera également plus facile d'implémenter des heuristiques de recherche. Le domaine est déterminé intelligemment, en éliminant les valeurs que l'on sait impossibles au préalable. Ici, les variables de successeur y peuvent prendre la valeur NON_ASSIGNED. Elle correspond aux créneaux non assignés, pour les courses non soumises à la contrainte de successeur, comme les courses Listed, mais aussi pour les courses situées en fin de programme, pour lesquelles la course suivante se situe potentiellement dans le programme suivant.

Contraintes

Liaison entre les deux types de variables Avant de considérer l'implémentation des contraintes, il faut lier les variables de course et de successeur. Essentiellement, il faut imposer qu'une course programmée, et la course suivante qui lui est associée via la variable de successeur, soient bien du même type. Il faut également gérer les valeurs possibles pour la variable de successeur en fonction de

la valeur prise par la variable de course. En effet, la date de programmation de la prochaine course de même type dépend de la distance parcourue lors de la course. En termes de variables, $y_t = A$ doit impliquer $x_A = x_t$ (i.e. la course A est du même type que la course t), pour A différent de NON_ASSIGNED.

La valeur NON_ASSIGNED doit être traitée à part. Il faut l'autoriser ou l'interdire en fonction de la situation, de la manière décrite ci-dessus. Il faut l'autoriser (voire l'imposer) pour les courses en fin de programme, et pour les courses non assignées.

Contrainte d'écart Une fois que les variables de successeur sont bien définies par la famille de contraintes précédente, on peut les utiliser. Illustrons ces contraintes avec des valeurs fictives. On veut par exemple que si la course 0 est sur une courte distance, son successeur se situe entre une semaine et demi et deux semaines plus tard. On veut que si la course 0 est sur une plus longue distance, son successeur se situe entre deux semaines et deux semaines et demi plus tard. Cela pourrait correspondre à $150 \leq y_0 \leq 200$ dans le premier cas, et $200 \leq y_0 \leq 250$ dans le second. On veut aussi que $y_0 = NON\_ASSIGNED$ si $x_0 = NON\_ASSIGNED$. Enfin, on veut que $y_0 = NON\_ASSIGNED$ si la course 0 est en fin de programme, c'est-à-dire si son successeur peut se situer dans le programme suivant, ce qui dépend aussi de la distance. Il s'agit donc d'une contrainte assez délicate à implémenter, avec de nombreux aspects à prendre en compte.

Contrainte HR Une contrainte supplémentaire est d'avoir entre 3 et 5 courses à Réclamer ou à Handicap par réunion. Cette contrainte est relativement simple à implémenter avec les variables de course. Il faut toutefois que cette contrainte ait un peu de souplesse. Parfois il ne sera pas possible de mettre 3 courses Réclamer/Handicap, il faut alors en mettre le maximum (une ou deux).

Autres contraintes Il existe plusieurs autres contraintes qui ne sont pas incluses dans ce modèle. Par exemple, le cahier des charges stipule qu'il doit y avoir une course à Handicap réservée aux chevaux de quatre ans (par opposition à quatre ans et plus) planifiée périodiquement. Une telle contrainte n'est pas vraiment problématique. En effet, on peut simplement convertir une course à Handicap quatre ans et plus en course à Handicap 4 ans sans que cela n'affecte les autres contraintes. C'est donc une contrainte qui peut être traitée dans un second temps.

Un autre avantage de cette modélisation est que de nombreuses contraintes dures peuvent être au moins en partie pré-traitées en réduisant le domaine des variables. Sont listées ci-dessous un ensemble de ces contraintes.

— Nombre de courses par réunion : par défaut, on attribue 8 courses par réunion, le nombre maximal, soit 8 variables par réunion. Avoir un nombre régulier comme ceci permet notamment la manipulation efficace des indices. Pour les réunions ayant moins de 8 courses (par exemple lorsqu'il y a également des courses d'obstacle dans la réunion), il suffit alors de fixer la variable correspondante à NON_ASSIGNED.

— Contrainte hippodrome : Chaque hippodrome ne fait pas courir toutes les distances. En particulier, il n'est pas possible d'organiser des courses de sprint partout. Ainsi, le domaine de chaque variable est fonction de l'hippodrome associé. En programmation par contraintes, le domaine n'est qu'une liste des valeurs possibles, et n'est a priori pas un intervalle. On peut tout simplement éliminer des domaines les valeurs que le prétraitement a jugé comme impossibles.

— Contraintes d'âge : suivant leur âge, les chevaux ne peuvent pas forcément courir toutes les distances. Par exemple, les chevaux de deux ans ne peuvent pas courir plus de 1100 mètres en mai, plus de 1200 mètres en juin etc... Il est facile d'intégrer cette contrainte dans le domaine des variables également.

— Courses préprogrammées : On peut fixer une course avant la planification en réduisant son domaine à une unique valeur. Il peut s'agir de courses Listed ou de courses à support d'événement, mais il peut également s'agir d'une course mise à la main par l'exécutif, dans une utilisation normale du logiciel d'aide à la décision.

Raccord avec le programme précédent

Une autre contrainte importante est le raccord avec le programme précédent, comme mentionné en partie 5.7.2. Il faut que la contrainte de successeur soit respectée en tenant compte du programme précédent. Cela aurait pu se faire sous forme de contrainte. La solution retenue est d'intégrer plutôt le programme précédent à la résolution. Pour ce faire, on crée également une variable de course (préfixée) et une variable de successeur à chacun des créneaux du dernier mois du programme précédent. En effectuant un prétraitement particulier lors du calcul du domaine de la variable de successeur, on peut essentiellement traiter les créneaux du programme précédent comme des créneaux classiques.

```
for (int j = N_START; j < vars.size(); j++) {
  for (int i = 0; i < vars.size(); i++) {
    // si y_i=j, x_j=x_i
    if (!ecart_type_vars[i]->Contains(j))
      continue;
    IntVar *JFollowsI = s.MakeIsEqualCstVar(ecart_type_vars[i], j);
    IntVar *IEqualsJ = s.MakeIsEqualVar(vars[i], vars[j]);
    Constraint *cte = s.MakeLessOrEqual(
        JFollowsI, IEqualsJ); // JFollowsI implique IEqualsJ
    s.AddConstraint(cte);
  }
}
```

FIGURE 16 – Exemple d'implémentation de contrainte

L'intérêt est double. D'une part, c'est une façon assez élégante de résoudre le problème de raccord, avec assez peu de code et de contraintes. D'autre part, cela permet de traiter un autre aspect de la programmation. Par exemple, on sait qu'il faut attendre septembre avant de proposer des courses de 2000 mètres aux chevaux de deux ans. A partir de septembre, il faut en proposer à intervalle régulier. Cette contrainte implicite n'est pas implémentée pour l'instant et n'est généralement pas nécessaire. En effet, la contrainte de successeur implique que cette dernière soit vérifiée au vu des valeurs, *pour peu qu'une course du même type soit présente dans le programme précédent.* Une solution très simple pour les courses de 2000 mètres pour deux ans est donc de placer une course fictive dans le programme précédent, par exemple le 15 août, et de laisser la contrainte de successeur faire le travail.

N.B. : La transition décembre-janvier est beaucoup plus subtile puisque les chevaux changent d'âge en janvier. Cette transition n'est pas étudiée pour le moment, puisque le programme décembre-février est constitué de meetings, qui sont programmés à la main.

Implémentation

La bibliothèque de programmation par contraintes or-tools, développée par Google en C++ sous licence GPL, a été utilisée.

La figure 16 donne un exemple d'implémentation de la contrainte liant les variables de successeur (nommées ECART_TYPE_VARS) et les variables de course (nommées VARS). Successivement, on crée une variable booléenne JFOLLOWSI qui indique si $y_i = j$, puis une variable booléenne IEQUALSJ qui indique si $x_i = x_j$. On impose ensuite que JFOLLOWSI implique IEQUALSJ en ajoutant au solveur une contrainte de type inférieur ou égal.

Pour cette contrainte, la démarche est assez simple car la contrainte se traduit simplement en

terme de contraintes élémentaires. La contrainte de successeur en revanche est bien plus subtile et utilise huit variables intermédiaires.

Dans l'exemple, la double boucle laisse penser que le nombre de contraintes générées est quadratique en le nombre de créneaux. C'est un peu plus délicat que cela puisque la boucle interne est en réalité parcourue environ autant de fois qu'il y a de créneaux dans la plage de temps disponible pour le successeur d'une course. Si on remarque que la longueur en jours de cette plage de temps, le nombre de réunions par jour et le nombre de créneaux par réunion sont des quantités bornées, le nombre de contraintes générées devient linéaire en le nombre de jours du programme. Cela reste vrai en incluant les autres familles de contraintes. Au final, environ 300.000 contraintes sont générées.

Heuristique de recherche

Comme mentionné en introduction, il n'est pas envisageable de parcourir l'intégralité de l'arbre de recherche. Après avoir défini le domaine du problème avec les contraintes, nous allons donc explorer une portion seulement de l'arbre de recherche. Nous avons essayé de parcourir par exemple 10 solutions, mais ce n'est pas utile. Puisque le solveur explore l'arbre de recherche de façon systématique, les dix solutions sont très similaires. En pratique, il est bien plus intéressant de repartir de la racine à chaque solution trouvée. L'heuristique est donc la suivante : générer les contraintes, parcourir l'arbre d'une manière bien définie, trouver une solution, repartir de la racine et répéter le procédé jusqu'à avoir assez de solutions. Enfin il faut choisir une solution. Nous nous sommes surtout intéressé à la façon de trouver cette première solution pour le moment.

Il faut donc se demander quel genre de solution on veut obtenir et comment la chercher. Nous allons reléguer à une phase ultérieure le remplissage de tous les créneaux d'une réunion. En effet, il n'est pas nécessaire pour respecter les contraintes présentées que tous les créneaux disponibles soient remplis. On espère qu'il ne sera pas trop difficile de les remplir plus tard. Par exemple, il sera possible de dédoubler une course programmée, pour peu que l'on ne dépasse pas cinq courses « Réclamer+Handicap » dans une réunion.

Pour fonctionner, le solveur or-tools utilise à chaque nœud de l'arbre l'instance de la classe DecisionBuilder, passée en paramètre du solveur. La classe DecisionBuilder possède une méthode Next qui indique au solveur quelle variable fixer à quelle valeur. La fonction Next doit aussi déterminer si l'on a atteint une feuille de l'arbre ou non. En C++, il est tout à fait possible d'écrire une classe qui

hérite de DecisionBuilder, pour définir la fonction Next de façon complètement personnalisée. Autrement dit, il est possible de choisir comme l'on veut la prochaine variable à fixer, et on peut choisir exactement quelle valeur lui donner, et ce à chaque nœud de l'arbre. La souplesse est maximale, et de nombreuses pistes méritent d'être étudiées. Après avoir présenté ce qui a été fait, on parlera brièvement des perspectives futures.

L'écriture de la fonction Next se divise typiquement en deux phases : choisir une variable, et choisir une valeur. On pourrait imaginer des variantes, comme choisir d'abord quel genre de course on veut placer (la valeur), puis de décider où on veut la placer (la variable). On pourrait aussi faire des allers/retours entre les deux phases, ou encore tirer au sort parmi plusieurs couples variable/valeur. Il n'y a pas de limitation tant que la décision finale est le choix d'un couple variable/valeur. Nous avons fait le choix de nous intéresser uniquement aux variables de successeur, puisque les fixer attribue aussi des valeurs aux variables de course, par contrainte. De par la nature des variables de successeur, l'idée naturelle est d'essayer de les fixer par ordre chronologique. On choisit donc la première variable qui n'est pas fixée dans la liste des variables. De plus, on souhaite programmer le moins de courses possible, c'est-à-dire que l'on veut autant de x_t à $NON\_ASSIGNED$ que possible. Ainsi, si $x_t \neq NON\_ASSIGNED$, on aura envie de choisir dès que possible $y_{t'} = t$, pour ne pas fixer de nouvelle variable à une valeur autre que $NON\_ASSIGNED$. C'est typiquement le cas lorsque deux courses identiques sont programmées le même jour, elles peuvent alors avoir le même successeur. Dans le cas où la création d'une nouvelle course est inévitable, on choisit un créneau au hasard dans la liste des valeurs possibles. Il est toujours possible que le solveur s'engage dans une mauvaise branche et ait du mal à en ressortir, c'est pourquoi on redémarre la recherche lorsqu'un certain nombre d'échecs est atteint, selon la séquence de Luby et al. [22]. Il s'agit finalement d'une heuristique simple, mais qui produit des résultats intéressants. Pour effectuer des essais, on a essayé de recréer un programme pour la période septembre-novembre 2016. Sur les 945 créneaux disponibles, il a été possible de constituer un programme satisfaisant les contraintes décrites dans le modèle en utilisant seulement environ 400 créneaux (variable car non déterministe) soit environ 42% des créneaux disponibles. En l'état actuel des choses, il est difficile d'apporter un bonne interprétation de ce résultat. D'une part, c'est une bonne chose qu'une version simplifiée du problème ne soit pas trop difficile à résoudre. Les courses Inédits et Maiden ne sont pas encore vraiment gérées, et d'autres contraintes sont à prévoir. Par exemple, on souhaite en réalité que les contraintes de successeur soient valides au sein d'une même fédération

(En particulier la fédération Paris). On apprécie donc d'avoir beaucoup de souplesse, par rapport à la technique utilisée mais aussi par rapport aux ajustements faisables sur les courses laissées libres. D'autre part, il est difficile encore de savoir si il est possible de remplir tous ces créneaux vides de façon intéressante. Concernant la vitesse de résolution (i.e. pour trouver une solution réalisable), le temps total, comprenant le temps de génération des contraintes et le temps de résolution, est de l'ordre de la seconde sur un ordinateur utilisant un i7-7600U, et suffisamment de RAM. Grâce à l'heuristique de recherche, le nombre d'échecs est très faible et le solveur trouve sans difficulté une solution. Il est difficile de prévoir comment le temps de résolution peut évoluer au fur et à mesure de l'ajout de contraintes, mais la souplesse du DecisionBuilder est un atout important dans l'adaptabilité à de nouvelles contraintes.

Pour l'instant le solveur résout une version minimaliste du problème, en positionnant le moins de courses possibles. Il y aurait eu d'autres manières de procéder. On aurait pu, comme le font les experts de l'équipe de planification de France Galop, essayer de placer en priorité les courses à Handicap, puis celles à Conditions, et enfin les courses à Réclamer. On pourrait aussi placer les courses à suivre, c'est-à-dire décider la valeur de la variable de successeur, au plus proche possible de la valeur cible de la périodicité requise. Il a été jugé qu'il était plus intéressant de prioriser les courses à Handicap par exemple, dans une phase ultérieure, après avoir déjà positionné cet ensemble « minimal » de courses. En effet, en priorisant les Handicaps dès le début, on prend le risque d'avoir des difficultés à placer les courses nécessaires ensuite. On a aussi estimé que le non déterminisme était très important, pour pouvoir comparer différents résultats et choisir le plus satisfaisant, vis -à-vis de la fonction objectif apprise, mais aussi des responsables de la planification. On pourrait par contre utiliser une distribution de probabilités non uniforme pour tomber le plus près possible de la périodicité de courses requise.

Les pistes pour d'autres phases de résolution sont aussi multiples. Par exemple, pour remplir les créneaux laissés libres, on peut tenir compte de ce qui est déjà placé. En effet, on a une idée générale des proportions des différents types de courses que l'on souhaite avoir. Il ne parait pas très difficile de traiter cet aspect. Encore une fois, si on veut que ce ne soit pas déterministe, on peut utiliser des probabilités non uniformes. Autre exemple : il est préférable de placer les courses à réclamer supérieures sur Paris, autant que possible. Même couplée à d'autres contraintes, on voit que cette préférence peut être implémentée dans le solveur avec un DecisionBuilder de façon triviale. De façon similaire, on essaie d'éviter qu'un hippodrome soit toujours l'hôte d'un même type de course. Ce genre de préférence peut

aussi être pris en compte.

Objectif

Nous avons finalement essayé d'utiliser un objectif à minimiser se basant sur l'apprentissage du chapitre 3. La minimisation d'un objectif en PPC consiste à rajouter une contrainte portant sur la valeur de l'objectif, de plus en plus serrée, et de voir à partir de quand le problème n'a plus de solutions. Cependant, réduire le domaine de l'objectif (le contraindre) n'induit pas de réduction du domaine des variables, et la PPC est donc très mal équipée pour cette tâche. Une idée que nous avons essayée est de générer plusieurs solutions pour sélectionner la meilleure, mais aucune optimisation n'avait lieu et les valeurs des différents solutions étaient très proches. L'impossibilité de travailler avec un objectif est une des raisons principales de l'abandon de cette méthode.

le cnam | Antoine HOUDAYER
Planification des courses de plat au galop | le cnam

Résumé : Nous étudions, dans cette thèse, la conception et la réalisation d'un outil informatique pour assister la planification des courses de plat en contexte incertain. L'objectif est de pouvoir produire des programmes de courses de «qualité» qui respectent des contraintes de planification. Nous proposons une modélisation mathématique linéaire du problème en variables 0-1, ainsi qu'une preuve de la NP-complétude du problème de décision associé. Un algorithme de recuit simulé adapté est alors utilisé pour apporter en pratique des solutions à des instances réelles du problème. Des méthodes d'apprentissage statistique sont implémentées dans le but de prédire le nombre de participations aux courses, quantité d'intérêt incertaine et pertinente pour évaluer la qualité des programmes outre le respect des contraintes. Ces travaux conduisent au développement d'un algorithme de simulation de la participation des chevaux aux courses. Enfin, un algorithme de recuit simulé multi-objectif est proposé, permettant l'intégration de cette simulation chronophage au recuit simulé.

Mots-clés : Planification, incertitude, programmation linéaire en variables 0-1, apprentissage, simulation, recuit simulé, courses hippiques

Abstract : We study in the present thesis the conception and implementation of a software that would assist the scheduling process of flat horse races in uncertain context. It aims to produce «good» race programs, that also obey scheduling constraints. We write a mathematical 0-1 linear formulation of the problem, and we give a proof of the NP-completeness of the associated decision problem. We then use an adapted simulated annealing algorithm to produce solutions for real instances. Machine learning algorithms are implemented to try and predict the number of participants to the races, which is an uncertain quantity of interest to estimate the quality of the programs, besides the satisfaction of the constraints. It leads to the developpement of an algorithm that simulates the participation of the horses to the races. Finally, we design a multi-objective simulated annealing algorithm, that can take into account this time consuming simulation.

Keywords : Scheduling, uncertainty, 0-1 linear programming, machine learning, simulation, simulated annealing, horse races