Titre: Title:	Conditions d'interaction entre apprentissage machine du prétraitement rapide et le problème des blocs mensuels personnalisés
Auteur: Author:	Philippe Racette
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Racette, P. (2025). Conditions d'interaction entre apprentissage machine du prétraitement rapide et le problème des blocs mensuels personnalisés [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/64916/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/64916/
Directeurs de recherche: Advisors:	Guy Desaulniers, & Andrea Lodi
Programme: Program:	Doctorat en mathématiques de l'ingénieur

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Conditions d'interaction entre apprentissage machine du prétraitement rapide et le problème des blocs mensuels personnalisés

PHILIPPE RACETTE
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques de l'ingénieur

Avril 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Conditions d'interaction entre apprentissage machine du prétraitement rapide et le problème des blocs mensuels personnalisés

présentée par **Philippe RACETTE**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Louis-Martin ROUSSEAU, président
Guy DESAULNIERS, membre et directeur de recherche
Andrea LODI, membre et codirecteur de recherche
Issmaïl EL HALLAOUI, membre
Axel PARMENTIER, membre externe

DÉDICACE

Au petit prince, qui saura se reconnaître. À mes parents.

REMERCIEMENTS

La réalisation de cette thèse n'aurait pas été possible sans l'aide de nombreuses personnes.

Sur le plan professionnel, je tiens d'abord à remercier celui qui a été mon directeur de thèse presque jusqu'à la ligne d'arrivée, François Soumis. Je le remercie pour l'opportunité qu'il m'a donnée, pour son soutien et pour sa confiance tout au long de ce parcours. Ce serait peu dire que son décès m'a touché. Au cours des plusieurs années qu'ont duré mon doctorat, j'en suis venu à voir M. Soumis comme un homme bon, humain et pragmatique à la fois. Il m'a accompagné au cours d'un processus ardu, intense, et qui m'a demandé de grandir en tant que personne sur plusieurs plans. Je n'oublierai pas l'opportunité qu'il m'a donnée et l'influence qu'il aura eue sur ma vie.

Je remercie également mon codirecteur de recherche, Andrea Lodi, pour son appui scientifique, technique et financier tout au long de mes études, ainsi que Guy Desaulniers pour avoir pris la relève en tant que directeur lors des dernières semaines.

Je remercie Frédéric Quesnel pour ses nombreux conseils, notamment sur le plan de la rédaction scientifique, et pour les collaborations fructueuses que nous avons eues. De plus, je remercie François Lessard et Benoît Rochefort pour le soutien technique qu'ils m'ont apporté à plusieurs reprises.

Je souhaite également souligner les différentes personnes que j'ai côtoyées durant mes années au GERAD. Ceci inclut Yassine Yaakoubi, pour ses conseils et encouragements, et Ryad Chelbani, qui a rendu plusieurs de ces années plus agréables et cordiales. Je remercie également Alice Wu, Patrick Munroe, Ali Barooni, Anis Noureddine et Nadia Rasouli.

Je tiens à exprimer ma reconnaissance envers Louis-Marc Mercier, qui est à la fois un ami et celui qui m'a recommandé de venir au GERAD. Ce parcours n'aurait pas eu lieu sans l'intérêt qu'il a manifesté à m'aider.

Je tenais à remercier les membres du jury qui ont accepté de lire cette thèse et de l'évaluer : MM. Desaulniers et Lodi, ainsi qu'Issmaïl El Hallaoui, Louis-Martin Rousseau et Axel Parmentier.

Je remercie les organismes qui ont contribué au financement de ce projet : le CRSNG, IBS Software, la CERC-DS4DM, et Polytechnique Montréal.

Je remercie également certaines personnes qui m'ont marqué plus tôt dans la vie. D'abord Christiane et Marilou, mes enseignantes des troisième et quatrième années et d'immersion anglaise. Je remercie aussi Mmes Nicole Chaput, animatrice de pastorale, et Isabelle Bailey,

qui m'a enseigné le français en 2ème secondaire. Finalement, je remercie M. Paul Guertin, qui a renforcé ma passion des mathématiques, notamment à travers la tenue de concours de mathématiques, sa bienveillance et ses nombreux encouragements.

Sur le plan personnel, je tiens à remercier ma famille et mes amis dont notamment Elisabeth (et mon jeune filleul Victor), Émilie, Jean-Michel, Kelvin, la famille Larocque-Sévigny, Mirabelle, Raphaël, mon père, ma mère, mes grands-parents qui m'ont toujours aimé, ainsi que mes amis à plumes et à fourrure. Dans l'ensemble, vous m'avez tous et toutes aidé à mettre les choses en perspective. J'ai connu de nombreux hauts et bas pour en arriver jusqu'ici. La constante dans tout ce processus fut vous tous, et vous m'avez fait réaliser à quel point nos relations sont ce qu'il y a de plus important dans la vie. Je vous remercie donc du fond du cœur. Vous êtes ma joie.

RÉSUMÉ

Le problème de la création de blocs mensuels personnalisés pour pilotes (CRP) s'inscrit dans la famille des problèmes de création d'horaires d'équipage aérien. À partir de rotations, soit des séquences de quelques vols et repos commençant et se terminant à la même base, le problème exige de créer des horaires mensuels complets pour les pilotes disponibles de sorte que chaque rotation soit assignée à exactement un pilote. Ceci doit être fait en tenant compte de différentes contraintes portant pour certaines sur l'ensemble du bloc mensuel, et pour d'autres sur les horaires des pilotes individuels. Chaque pilote émet des préférences tant sur des vols spécifiques que sur des périodes de congé, et l'objectif à maximiser porte alors sur la satisfaction totale apportée aux pilotes par les horaires qui leur sont fournis.

La création de blocs mensuels personnalisés étant complexe, de nombreuses méthodes ont été développées afin d'en accélérer la résolution avec le moins d'impact possible sur l'objectif du modèle d'optimisation. Plusieurs de ces méthodes se basent sur la génération de colonnes. Cette technique permet de résoudre un problème restreint à chaque itération de l'algorithme d'optimisation. Le modèle est graduellement agrandi en résolvant un sous-problème par pilote représenté par un graphe et où trouver un horaire réalisable revient à résoudre un problème de plus court chemin avec contraintes de ressources. Cette forme de génération de colonnes a déjà été implantée commercialement à l'intérieur de solveurs performants.

De plus, au cours des dernières années et avec l'essor de l'apprentissage machine (ML), de nouveaux types d'algorithmes d'apprentissage et métaheuristiques ont été considérés tant pour suppléer ou assister les solveurs de recherche opérationnelle (RO) traditionnels. Ces innovations ont simultanément connu des succès intéressants et rencontré des limites. Toutefois, peu d'analyses systématiques ont été réalisées pour déterminer clairement dans quels contextes l'interaction ML-RO a du potentiel, et quel type d'algorithme devrait être utilisé le cas échéant.

La présente thèse aborde donc tant l'enjeu d'accélérer la résolution du CRP que celui d'évaluer le potentiel des méthodes d'apprentissage pour ce problème. Pour ce faire, nous créons une procédure d'affectation successive qui construit un bloc mensuel un jour à la fois. Chaque problème d'affectation est représenté par un graphe où les coûts des arcs indiquent la valeur d'affecter une rotation à un pilote. Ces coûts sont calculés à l'aide de poids appris par ML au travers d'un algorithme d'évolution.

À l'aide de la solution obtenue, cette dernière est utilisée pour mieux anticiper ou améliorer le fonctionnement d'un solveur CRP correspondant à l'état de l'art. L'apprentissage réalisé

fait donc partie de la famille des applications dites par apprentissage du prétraitement et qui visent à fournir de l'information utile au solveur avant que l'optimisation ne soit entamée. Un avantage de cette approche est que l'information fournie en entrée peut être la même pour différentes formes de prétraitement visant à améliorer la génération de colonnes. Puisque la qualité de ce qui est fourni en entrée reste inchangée également, il devient possible de mettre en lumière les types d'interaction ML-RO pouvant tirer parti d'une information supplémentaire même limitée.

Ainsi, dans un premier temps, nous considérons la planification du CRP. Avant le début de chaque mois, les employés de la compagnie aérienne doivent déterminer combien de pilotes doivent être rendus disponibles, gardés en réserve, ou assignés à des activités spéciales (formations, congés). Nous mesurons ensuite si la qualité des solutions obtenues par affectation successive permet de prédire celle obtenue par le solveur standard. Nous trouvons une relation solide entre la qualité de ces solutions qui dépasse ce que d'autres méthodes rapides obtiennent. Les planificateurs peuvent alors anticiper si la configuration actuelle du problème permet d'obtenir de bonnes solutions dans le solveur et modifier la planification en quelques secondes au besoin.

Ensuite, nous implantons une méthode de résolution du CRP par fenêtrage, où chaque partie de la solution fournie en entrée est réoptimisée à tour de rôle en gardant le reste du bloc mensuel fixé. Nous avons comparé l'utilisation du fenêtrage avec et sans solution ML initiale à d'autres heuristiques rapides ou naïves. Bien que dans les deux formes de fenêtrage, une accélération dépassant un facteur 10 ait été observée, les solutions de l'affectation successive obtenues grâce à l'apprentissage ML ont permis d'obtenir un objectif en moyenne à seulement 1% de l'optimalité pour deux grandes instances avec différents niveaux de productivité moyen attendue par pilote. Ces résultats dépassent également ceux obtenus par les autres méthodes testées.

Enfin, nous évaluons la capacité des solutions ML à maximiser les bénéfices associés à l'agrégation dynamique de contraintes (DCA). Ce but requiert d'apprendre quelles séquences de rotations sont susceptibles de se retrouver dans le bloc mensuel optimal. Nous considérons différentes méthodes afin de rehausser les solutions fournies à l'algorithme DCA, telles que varier le choix de l'algorithme ML, la création de variables ML supplémentaires et riches en information potentielle, et une affectation successive moins myope. Face à l'absence de résultats générés par ces techniques supplémentaires, nous concluons que prédire le contenu d'une solution optimale est un défi souvent peu réaliste par rapport au fait de générer des solutions contenant de l'information utile plus générale et à raffiner ensuite par un solveur traditionnel.

En conclusion, cette thèse permet de traiter en profondeur l'utilisation de méthodes ML pour le problème des blocs mensuels personnalisés. En comparant les succès et les difficultés rencontrés au cours de ce processus, nous avons trouvé de meilleures façons de résoudre le CRP. Nous avons également amorcé une démarche d'analyse structurée des interactions de type ML/RO, dans ce cas-ci pour le CRP avec l'apprentissage du prétraitement, qui a le potentiel de s'étendre à d'autres problèmes.

ABSTRACT

The crew rostering problem for pilots (CRP) is part of the family of airline crew scheduling problems. Starting from pairings, or sequences of a few flights starting and ending at the same base, the problem requires creating monthly rosters for the pilots available to work such that each pairing is assigned to exactly one pilot. This must be done while taking into account various constraints related to the entire roster, and others related to the individual schedules of the pilots. Each pilot states preferences over both specific flights and vacations, and the objective is to maximize the sum of the total satisfaction brought to the pilots by their schedules.

The creation of rosters being a complex task, many methods have been developed to speed up its resolution while limiting the impact of this acceleration on the quality of the optimization model's objective. Many of these methods build upon column generation, which has made large scheduling problems tractable. This technique makes it possible to solve a restricted problem at each iteration of the optimization algorithm. The model for each subsequent restricted problem is gradually expanded by solving one subproblem per pilot. The subproblems are represented by graphs where finding a feasible schedule comes down to solving a shortest path problem with resource constraints. This form of column generation has already been implemented commercially within high-performing CRP solvers.

Furthermore, in recent years and with the rise in the use of machine learning (ML), new types of learning algorithms and metaheuristics have been considered to either replace or assist operations research (OR) traditional solvers. These innovations have lead to interesting successes, but came with limitations. However, few systematic analyses have been conducted to determine clearly in which contexts ML-OR interaction has potential, and which types of algorithms should be considered when it does.

For this reason, this dissertation considers two main issues. The first involves accelerating the resolution of the CRP, while the other is assessing the potential of ML methods in an ML-OR context. We build a sequential assignment procedure that forms a monthly roster one day at a time. Each assignment problem is represented by a graph where arc costs show the utility of assigning a given pairing to a pilot. These costs are calculated with the help of weights learned by ML through an evolutionary algorithm.

With the solution obtained, we anticipate or improve the behavior of a state-of-the-art CRP solver. This type of learning is part of the family of learning to preprocess, where useful information is provided to the solver before the optimization procedure is launched. An

advantage of this approach is that the input used for preprocessing can be kept unchanged even when testing different ways of improving column generation (i.e., different forms of preprocessing). Since the input's quality remains the same as well, we can see which ML-OR interactions can leverage even limited additional information.

First, we consider CRP planning. Before the beginning of each month, the airline scheduling workers must determine how many pilots must be made available, stand by on reserve, or be assigned to special activities (e.g., specialized training, vacations). Then, we measure if the quality of the solutions obtained by sequential assignment makes it possible to predict that of rosters found in the standard solver. We find a solid relationship between the quality of these solutions that exceeds what other fast methods provide. The scheduling workers are then able to anticipate whether the current configuration of a CRP instance allows for good solutions in the solver and to modify the planning done so far in a few seconds as needed.

Next, we implement a solution method with windowing. In this context, each part of the solution provided as input is reoptimized in turn while keeping the rest of the roster fixed. We have compared windowing with and without an initial ML solution to other fast or naive heuristics. While an acceleration of the CRP's resolution of an order of more than 10 is observed with both forms of windowing, the sequential assignment solutions obtained with ML have allowed to get an objective on average only 1% away from optimality for two large instances with different levels of average expected productivity per pilot. These results outperform those found by other methods.

Our last specific contribution lies in evaluating the ability of ML solutions to maximize the benefits associated with dynamic constraint aggregation (DCA). This goal requires learning which sequences of pairings are part of the optimal roster for the problem with good accuracy. We consider different methods to enhance the solutions given as input to the DCA procedure, such as varying the choice of the ML algorithm, the creation of additional ML features rich in potential information, and a less myopic form of sequential assignment. In the absence of convincing results yielded by these other techniques, we conclude that predicting the content of an optimal solution is an often unrealistic challenge in comparison to creating solutions containing useful, but more general information that is then to be refined by a traditional optimization solver.

In conclusion, this dissertation allows us to treat in depth the use of ML methods for the CRP. By comparing the successes and obstacles encountered during this process, we have found ways to solve the CRP more efficiently. We have also started a process of structured analysis of the suitability of ML/OR interaction in general, with in this case a focus on the CRP with learning to preprocess that has the potential to extend to other problems.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AIC	Aikake information criterion
BIC	Bayesian information criterion
CMA-ES	stratégie d'évolution avec adaptation de la matrice de covariance *(covariance matrix adaptation evolution strategy)*
CO	optimisation combinatoire (*combinatorial optimization*)
CPP	problème des rotations d'équipage (*crew pairing problem*)
CRP	problème des blocs mensuels personnalisés (*crew rostering problem*)
DCA	agrégation dynamique de contraintes (*dynamic constraint aggregation*)
ES	stratégie d'évolution (*evolution strategy*)
GNN	réseau de neurones en graphes (*graph neural network*)
IPS	simplexe primal amélioré (*improved primal simplex*)
LP	programmation linéaire (*linear programming*)
MIP	programme mixte en nombres entiers (*mixed integer program*)
MIQP	programme quadratique mixte en nombres entiers (*mixed integer quadratic program*)
ML	apprentissage machine (*machine learning*)
NES	stratégie d'évolution naturelle (*natural evolution strategy*)
OLS	moindres carrés (*ordinary least squares*)
OR	*operations research*
ReLU	unité linéraire rectifiée (*rectified linear unit*)
RL	apprentissage par renforcement (*reinforcement learning*)
RMP	problème maître restreint (*restricted master problem*)
RO	recherche opérationnelle
SPPRC	problème de plus court chemin avec contraintes de ressources (*shortest path problem with resource constraints*)
TSP	problème du voyageur de commerce (*traveling salesman/salesperson problem*)
VRP	problème de tournée de véhicules (*vehicle routing problem*)

CHAPITRE 1 INTRODUCTION

Le domaine des transports soulève plusieurs enjeux et problèmes importants. Étant donné l'envergure des tâches à réaliser, différentes méthodes d'optimisation ont été développées afin d'améliorer leur résolution. En effet, qu'il s'agisse de tournées de véhicules, de création d'itinéraires ou de création d'horaires, les applications réelles de ces problèmes doivent composer avec des réseaux de grande taille. La capacité d'une méthode à résoudre efficacement le problème approprié à une situation de la vie courante prend alors une importance économique et sociale, par exemple à travers des échanges de biens et services sans gaspillage ni retard.

Parmi les problèmes mentionnés ci-dessus se trouve la création d'horaires personnalisés pour pilotes d'avion (CRP). Cette tâche consiste à concevoir des horaires complets d'une durée d'un mois (aussi appelé « l'horizon »). Ces horaires doivent être assignés aux pilotes disponibles au cours de l'horizon à venir de sorte que chaque pilote en reçoive exactement un. Les horaires, aussi dénotés « blocs mensuels », sont constitués de rotations aériennes et de jours de congé, les rotations constituant chacune une série de vols alternant avec des nuits de repos. Chaque rotation commence et se termine au même aéroport, dure quelques jours (typiquement entre 1 et 5 jours), et doit se trouver dans exactement un horaire assigné.

De plus, ce travail doit être réalisé en tenant compte de diverses contraintes, dont plusieurs sont liées aux conventions collectives, et ce, en composant avec un nombre d'horaires possibles suffisamment élevé pour qu'ils ne puissent pas être énumérés dans un temps raisonnable. Ceci justifie le besoin de méthodes hautement performantes pour pouvoir traiter le défi que représente le CRP en un temps raisonnable.

La portée du problème de la conception de blocs mensuels personnalisés touche plusieurs plans. En effet, les rotations précises figurant à l'intérieur du bloc d'un pilote en particulier influent sur le degré de satisfaction qu'il retire de son horaire de travail. La manière dont les rotations et congés préférés sont accordés entre les pilotes a également un impact sur la somme totale de la satisfaction pour l'ensemble d'entre eux.

En outre, l'importance du CRP tient également au fait que les méthodes ayant du succès pour améliorer la résolution d'un problème d'optimisation combinatoire sont souvent transférables à d'autre problèmes. Par exemple, les méthodes d'apprentissage machine (*machine learning* en anglais, ou ML) développées dans un certain contexte sont souvent réutilisées et testées sur des problèmes différents. Étant donné l'engouement croissant envers ces méthodes dans le domaine de l'optimisation combinatoire (*combinatorial optimization*, ou CO), les observations émises sur la capacité de tels algorithmes à accélérer l'obtention de bonnes solutions dans le

cadre du CRP peuvent s'avérer utiles dans le cas de tâches similaires.

Dans cette thèse, nous présentons donc une méthode d'affectation successive qui est mise au point à l'aide d'un modèle ML. Cette procédure construit en quelques secondes une solution au CRP un jour à la fois et en ordre chronologique. La qualité de la solution peut alors être utilisée afin de prédire le comportement d'un solveur correspondant à l'état de l'art, ou d'en améliorer le fonctionnement. Certaines méthodes qui exploitent la solution en conjonction avec le solveur, telles que l'agrégation dynamique de contraintes (DCA), fonctionnent le mieux avec une solution préliminaire dont certaines parties apparaissent avec un niveau de confiance élevé dans la solution optimale. D'autres approches, telles que l'optimisation avec fenêtrage, au contraire, peuvent être moins sensibles à la nature de l'information fournie en entrée.

L'étude de l'efficacité de la procédure d'affectation successive selon son utilisation permet de tirer certaines leçons sur les forces et les limites de l'apprentissage machine pour rehausser la résolution de différents modèles d'optimisation combinatoire, et plus particulièrement dans le domaine des transports. En ce sens, les résultats et les conclusions obtenues pour le CRP font du problème des blocs mensuels personnalisés une étude de cas dont les faits saillants ont le potentiel de s'étendre au reste du domaine qui lui est apparenté.

Le reste de cette introduction vise donc à présenter les fondements essentiels à la compréhension des analyses qui sont décrites par la suite en lien avec ce problème. Dans la section 1.1 du présent chapitre, nous situons et décrivons de façon plus approfondie le CRP. Dans la section 1.2, nous développons l'idée de génération de colonnes à l'aide de la décomposition de Dantzig-Wolfe, une technique fréquemment utilisée en optimisation combinatoire dans le cas d'un nombre intraitable de variables. Dans la section 1.3, nous présentons brièvement certaines méthodes employées dans la littérature pour aider à la résolution de problèmes CO, y compris certaines approches traditionnelles et d'autres liées à l'apprentissage machine. Nous concluons ce chapitre avec la section 1.4, où les contributions principales de ce travail de recherche sont explicitées.

1.1 Description du problème

Le problème de la création des blocs mensuels s'inscrit dans le cadre plus large de la planification aérienne. Ce processus comprend plusieurs étapes, dont la planification des vols, l'affectation de la flotte aérienne, où les différents appareils sont attitrés aux vols créés à l'étape précédente, la conception d'itinéraires pour les avions, et la création d'horaires d'équipage. Cette dernière est elle-même divisée en deux étapes, soit la création des rotations et celle

des blocs mensuels. Dans certains cas, les rotations et les blocs mensuels sont créés de façon intégrée, soit en résolvant un seul modèle mathématique.

Il existe différentes manières de créer des blocs mensuels selon la façon de faire et les valeurs des compagnies aériennes. La méthode dite par blocs anonymes implique la création d'horaires anonymes, qui sont ensuite présentés aux pilotes et pondérés par eux selon leurs préférences personnelles. Il est fréquent dans ce contexte que les pilotes ayant le plus d'ancienneté puissent assigner davantage de poids au total à différents horaires. Dans certains cas, les horaires sont simplement attribués par ordre décroissant de séniorité sans tenir compte d'autres critères. La méthode dite par horaires personnalisés, quant à elle, demande plutôt aux pilotes de formuler leurs préférences au préalable. Les blocs mensuels sont alors créés de façon à satisfaire le plus possible de requêtes, par exemple en évaluant la somme des préférences pour les requêtes accordées. Enfin, la méthode par blocs personnalisés avec séniorité réplique largement celle des horaires personnalisés tout en tenant compte de l'ancienneté comme pour la méthode par blocs anonymes standards.

La création de blocs mensuels requiert de choisir un objectif à optimiser. Des critères courants sont la satisfaction des pilotes, l'équité, le coût (par exemple associé au fait de faire travailler certains pilotes plus ou moins expérimentés pendant un nombre d'heures données, ou au nombre d'heures supplémentaires travaillées), ou encore la robustesse des horaires, certains horaires étant plus faciles à ajuster en cas de perturbations. Il est parfois aussi possible de combiner plusieurs critères, l'objectif étant alors une somme avec plusieurs termes de manière à refléter l'importance relative de ceux qui sont retenus. L'objectif tient parfois compte de pénalités associées à la violation de certaines contraintes douces, telles que la non-affectation d'une rotation. Dans ce travail, nous nous concentrons sur le cas des horaires personnalisés, que nous utiliserons à partir de maintenant de façon interchangeable avec l'expression « blocs mensuels personnalisés ».

Quel que soit le type d'objectif pertinent et la variante spécifique du problème considéré, le CRP inclut différentes contraintes qui font partie de catégories prévisibles. Les contraintes d'affectation des rotations assurent que chaque rotation soit comprise dans exactement un horaire affecté à un pilote. Selon le modèle, elles peuvent permettre la présence de variables d'écart pour le cas où une rotation ne peut pas être assignée : une pénalité est alors associée à cette variable d'écart. De plus, les contraintes d'horaires imposent que chaque pilote reçoive un bloc mensuel complet. Elles n'incluent aucune variable d'écart, car un pilote ne peut pas être dispensé d'horaire, même dans le cas où il se fait assigner des congés tout le mois. Ensemble, les contraintes d'affectation et d'horaires font en sorte que le CRP peut être considéré comme un problème de partitionnement d'ensemble.

Il existe aussi d'autres contraintes, dont la variété est trop grande pour qu'elles puissent toutes être nommées. Elle peuvent également être regroupées en classes plus générales. Par exemple, Kohl et Karisch [1] considèrent qu'il existe des contraintes horizontales et verticales. Les contraintes horizontales ne touchent que les blocs mensuels individuels. On peut affirmer que ces contraintes assurent la réalisabilité des blocs mensuels créés. Par exemple, on pourrait considérer le nombre de jours de repos octroyés au cours du mois, le total d'heures de vol, le nombre de jours de travail consécutifs, ou d'autres variables qui sont cumulatives tout au long de l'horizon. Les contraintes verticales, quant à elles, touchent plusieurs blocs mensuels simultanément et s'assurent le plus souvent que certaines tâches sont couvertes de manière suffisante. Elles se rapportent parfois aux contraintes de langues ou de qualifications techniques, où un certain nombre de membres d'équipage doivent posséder certaines compétences pendant un vol.

Dans ce travail, nous considérons le cas des pilotes qui sont considérés comme capitaines de la rotation, ce qui signifie d'une certaine façon que le problème est simplifié par rapport à d'autres cas, tel que celui des agents de bord où plusieurs agents de bord peuvent être requis, ou encore le cas où les pilotes peuvent être plus d'un, et parfois assumer des fonctions jugées moins complexes en cas de besoin. Cette variante du problème se concentre donc essentiellement sur les contraintes horizontales qui assurent que les blocs mensuels accordés aux pilotes soient à la fois réalisables et satisfaisants.

Les contraintes et l'objectif exacts utilisés dans ce travail sont présentés au chapitre 4 du présent document. Cependant, sur la base de ce qui précède, il est possible de fournir une formulation mathématique générale et flexible quant au choix de ces aspects essentiels du problème. En particulier, nous nous inspirons ici du travail de Desaulniers *et al.* [2], où les auteurs présentent une formulation unifiée des problèmes de tournées de véhicules (ou *vehicle routing problem*, VRP) avec contraintes de ressources. Ils introduisent des variables de flot correspondant à l'affectation de certaines activités qui se suivent dans l'horaire de chaque pilote de sorte que chaque horaire soit réalisable et que l'ensemble des horaires recouvre toutes les rotations. Ceci est réalisé en résolvant un sous-problème par pilote. Nous notons que nous pouvons aussi définir un modèle différent où les variables d'affectation portent sur les horaires réalisables générés par chaque sous-problème. Ce modèle a une formulation plus légère, car les sous-problèmes assurent que les contraintes horizontales portant sur le nombre de jours de congés assignés, le nombre de jours de travail consécutifs, et le nombre d'heures de vol soient respectées pour chaque pilote, notamment en les traitant comme des ressources. Nous utilisons ce modèle allégé au chapitre 4, mais détaillons ici l'approche avec variables de flot pour donner un aperçu plus complet du CRP.

Nous présentons donc ici cette formulation plus générale adaptée à notre cas particulier. Considérons tout d'abord la représentation d'une instance du CRP sous forme de graphe. En effet, les problèmes de planification aérienne peuvent souvent être représentés sous cette forme. On définit ici un graphe acyclique dont les nœuds correspondent au début et à la fin de différentes activités, tandis que les arcs les liant correspondent à la réalisation desdites activités. La topologie du graphe est particulière en ce que les nœuds et les arcs sont situés sur un continuum temporel : les arcs inclus lient seulement des tâches qui peuvent se suivre. Un horaire, ou bloc mensuel, est alors un chemin qui traverse le graphe en respectant certaines contraintes, tandis qu'une solution au CRP est un ensemble de tels horaires trouvés simultanément pour chaque pilote.

C'est dans ce contexte que la notion de ressource prend son sens. En effet, une ressource est une quantité évaluée à chaque nœud et qui est consommée ou accrue en parcourant un arc. Par exemple, le temps est un exemple de ressource qui intervient d'une façon critique, car il est essentiel que celui-ci se trouve dans l'intervalle requis pour commencer l'activité prévue à un nœud. Il est augmenté chaque fois qu'un arc est sélectionné par la durée de l'activité qui lui est associée. Les autres ressources considérées pour notre formulation sont le nombre de jours de repos à accorder avant que le minimum exigé soit atteint, le nombre de jours de travail consécutifs effectués, et le nombre d'heures de vols réalisées au cours de l'horizon.

Dans le cas qui nous concerne, il est possible de définir des sous-ensembles d'arcs, chacun d'entre eux permettant de définir un sous-graphe pour le sous-problème de chaque pilote, et de donner un coût à chaque arc. En effet, notre critère d'optimisation étant la satisfaction, il est possible d'accorder un poids positif à chaque arc affectant une rotation ou un congé (dans le cas présent, une période de trois jours consécutifs de repos) associé à une préférence exprimée par un pilote. L'objectif est alors la somme de la satisfaction obtenue à la suite de l'affectation de ces arcs, dont on soustrait des pénalités liées aux variables d'écart pour les rotations non-assignées.

Il est alors possible de définir le problème plus formellement. Soit le graphe acyclique $G = (V, A)$, où V est l'ensemble des nœuds du graphe, et A l'ensemble de ses arcs. Soit K l'ensemble des pilotes, et soient chacun de ses membres $k \in K$. Soit $G^k = (V^k, A^k)$ le sous-graphe de G associé au pilote k avec nœuds V^k et le sous-ensemble d'arcs A^k. On a alors $G = \cup_{k \in K} G^k$, l'union de ces sous-graphes. Soit $o(k)$ le nœud source, $d(k)$ le nœud puits, N les nœuds intermédiaires et $V^k = \{o(k), d(k)\} \cup N$ leur union pour le pilote k. On peut diviser l'ensemble des nœuds intermédiaires N davantage en considérant ceux liés au départ d'une rotation, W, et les autres nœuds, $N \setminus W$. Soit X_{ij}^k la variable binaire valant 1 si l'arc $(i, j) \in A^k$ liant les nœuds i et j dans V^k est emprunté et 0 sinon, et soit c_{ij}^k le coût de cet arc. Soit s_i la variable

d'écart pour l'absence d'affectation de la rotation $i \in W$ et c_i son coût (comme il s'agit ici d'une formulation générale, on omet dans ce chapitre le cas des congés préassignés et de leurs variables d'écart, qui ne font partie que de certains modèles particuliers et qui sont associés à des contraintes de couverture analogues à celles des rotations). Finalement, soit T_i^{kr}, t_{ij}^{kr}, a_i^{kr} et b_i^{kr} la valeur de la ressource $r \in R$ pour le pilote k au nœud i, la consommation de la ressource r pour le pilote k qui passe par l'arc $(i,j) \in A^k$, et les bornes inférieure et supérieure de la ressource r pour le pilote k au nœud i, respectivement.

On peut alors définir le modèle suivant :

$$\max \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij}^k X_{ij}^k - \sum_{i \in W} c_i s_i \tag{1.1}$$

sujet à :

$$\sum_{k \in K} \sum_{(i,j) \in A^k} X_{ij}^k + s_i = 1, \quad \forall i \in W \tag{1.2}$$

$$\sum_{j:(o(k),j) \in A^k} X_{o(k),j}^k = 1, \quad \forall k \in K \tag{1.3}$$

$$\sum_{i:(i,d(k)) \in A^k} X_{i,d(k)}^k = 1, \quad \forall k \in K \tag{1.4}$$

$$\sum_{(i,j) \in A^k} X_{ij}^k - \sum_{(j,i) \in A^k} X_{ji}^k = 0, \quad \forall k \in K, \ \forall i \in N \tag{1.5}$$

$$X_{ij}^k (T_i^{kr} + t_{ij}^{kr} - T_j^{kr}) \leq 0, \quad \forall r \in R, \ \forall k \in K, \ \forall (i,j) \in A^k \tag{1.6}$$

$$a_i^{kr} \leq T_i^{kr} \leq b_i^{kr}, \quad \forall r \in R, \ \forall k \in K, \ \forall i \in V^k \tag{1.7}$$

$$X_{ij}^k \in \{0,1\}, \quad \forall k \in K, \ \forall (i,j) \in A^k \tag{1.8}$$

$$s_i \in \{0,1\}, \quad \forall i \in W \tag{1.9}$$

L'objectif décrit par l'expression (1.1) montre que la qualité d'une solution au CRP est évaluée par la somme de la satisfaction apportée par chaque arc emprunté par un pilote dont est soustraite toute pénalité pour non-affectation d'une rotation. Les contraintes (1.2) assurent l'affectation des rotations (possiblement avec une variable d'écart). Les contraintes (1.3) et (1.4) assurent que chaque pilote ait un horaire complet en imposant le départ du nœud source et le retour au nœud puits appropriés, respectivement. Les contraintes (1.5) font en sorte que le flot sur les arcs de chaque sous-graphe soit conservé. Les contraintes (1.6) imposent que la valeur de toute ressource pour un pilote à un nœud donné soit au moins égale à sa valeur au nœud précédent à laquelle est ajoutée la consommation de la ressource sur l'arc emprunté. Les contraintes (1.7) s'assurent que les ressources soient dans

la fenêtre nécessaire pour tous les nœuds et pilotes. Finalement, comme le problème requiert une solution en nombres entiers, et en fait ici binaires, les contraintes (1.8) et (1.9) assurent que les variables de flot et d'écart soient binaires, respectivement.

La formulation ci-dessus est correcte, mais elle inclut des contraintes particulièrement complexes, telles que les contraintes de ressources (1.6) et (1.7). Cependant, il peut être constaté que celles-ci peuvent être traitées séparément par pilote, n'étant pas des contraintes verticales. Cette observation permet de simplifier considérablement le modèle lorsque intervient la génération de colonnes.

1.2 Génération de colonnes et branchement

Afin de pouvoir reformuler le modèle du CRP, nous décrivons la génération de colonnes. L'idée générale de cette technique est que pour les problèmes d'optimisation de grande taille, il peut être difficilement faisable de résoudre un modèle incluant toutes les variables possibles, voire d'énumérer ces dernières. Le principe est alors de résoudre un modèle plus restreint, limité à un sous-ensemble de variables ou colonnes (termes que nous utilisons de façon interchangeable) particulièrement prometteuses, et de répéter ce processus en ajoutant de nouvelles variables au fur et à mesure que la résolution progresse. Il faut cependant un moyen efficace de trouver les nouvelles colonnes.

À ce sujet, une technique qui est souvent mentionnée dans la littérature est la décomposition de Dantzig-Wolfe [3]. Cette reformulation divise le processus de résolution en deux phases alternantes : la résolution d'un problème maître restreint, et celle de sous-problèmes. Cette formulation permet d'utiliser la génération de colonnes, celles-ci étant alors créées en résolvant les sous-problèmes. Ainsi, pour mieux traiter les contraintes complexes se rapportant aux ressources, nous reformulons le problème.

Nous introduisons donc un modèle traitant les blocs mensuels personnalisés comme un problème de partitionnement (avec des variables de recouvrement portant sur les horaires). Soient K, W et les variables d'écart $s_i, i \in W$ définis comme précédemment. Soit Ω l'ensemble des horaires réalisables du point de vue des contraintes de ressources générés jusqu'à présent. Soit Ω_k un de ses sous-ensembles pour le pilote k, et p un horaire qui en fait partie. Soit c_p^k le coût associé à l'horaire p affecté au pilote k, et x_p^k la variable binaire représentant cette affectation. Soit a_{ip}^k une constante prenant la valeur 1 si la rotation $i \in W$ se trouve dans l'horaire p affecté au pilote k, et 0 autrement. Alors on peut formuler le problème ainsi :

$$\max \sum_{k \in K} \sum_{p \in \Omega^k} c_p^k x_p^k - \sum_{i \in W} c_i s_i \tag{1.10}$$

sujet à :

$$\sum_{k\in K}\sum_{p\in\Omega^k} a_{ip}^k x_p^k + s_i = 1, \quad \forall i \in W \tag{1.11}$$

$$\sum_{p\in\Omega^k} x_p^k = 1, \quad \forall k \in K \tag{1.12}$$

$$x_p^k \in \{0,1\}, \quad \forall k \in K, \forall p \in \Omega^k \tag{1.13}$$

$$s_i \in \{0,1\}, \quad \forall i \in W \tag{1.14}$$

où la fonction objectif (1.10) maximise la somme totale des coûts pour les blocs mensuels moins les pénalités pour les variables d'écart. Les contraintes (1.11)-(1.14) assurent l'affectation de chaque rotation à exactement un bloc mensuel, l'affectation d'exactement un bloc mensuel à chaque pilote, et la binarité des variables d'affectation et d'écart, respectivement.

Nous décrivons maintenant les phases de la génération de colonnes. La première est celle du problème maître restreint (*restricted master problem* en anglais, ou RMP). Lors de cette étape, les colonnes qui ont déjà été ajoutées au problème sont prises en compte et le CRP est résolu pour ce sous-ensemble. La deuxième phase est la génération de nouvelles colonnes. La manière de procéder est d'observer la structure particulière du problème à résoudre et de détecter les contraintes qui peuvent être traitées séparément, soit ici les contraintes de flot et de ressources (contraintes (1.3)-(1.8)). Un sous-problème, plus simple que le modèle initial, est alors créé pour ces contraintes, et chaque sous-problème est résolu de façon à générer des colonnes à coût réduit négatif qui peuvent être ajoutées au problème initial. Ce faisant, les deux phases peuvent alors alterner jusqu'à ce qu'une solution satisfaisante soit obtenue.

Le sous-problème pour le pilote k peut être représenté par le sous-graphe G^k. Les coûts sur les arcs, cependant, suivent une structure différente. En effet, pour obtenir de nouvelles colonnes pouvant améliorer l'objectif à la prochaine itération du RMP, il faut que leur coût réduit soit négatif. L'idée du sous-problème est alors de trouver un chemin partant du nœud source $o(k)$ jusqu'au nœud puits $d(k)$ satisfaisant toutes les contraintes de ressources et créant un horaire correspondant à une colonne répondant à ce critère.

Une façon d'appliquer cette méthode est de considérer les variables duales de la solution de l'itération précédente du RMP. En effet, il est alors possible d'exprimer le coût réduit de chaque horaire pour un pilote, $\overline{c}_p^k$, comme le coût c_p^k dont on soustrait une combinaison linéaire des variables duales. Nous donnons davantage de détails sur ce point au chapitre 5. Cependant, l'idée générale est que la différence entre le coût et la combinaison linéaire de variables duales qui équivaut à $\overline{c}_p^k$ peut être décomposée, de sorte que chaque arc du sous-graphe où les horaires de coût réduit négatif sont recherchés peut se voir assigner une valeur.

La somme de ces valeurs équivaut au coût réduit de l'horaire entier.

Trouver de bonnes colonnes additionnelles peut alors être réalisé en résolvant un problème de plus court chemin de façon à minimiser $\overline{c}_p^k$. Étant donné que les horaires ainsi générés doivent être réalisables, il faut cependant tenir compte des contraintes de ressources à l'intérieur de chaque sous-problème. Ce problème modifié est plus complexe et est typiquement appelé le problème de plus court chemin avec contraintes de ressources (souvent abrégé SPPRC, pour *shortest path problem with ressource constraints* en anglais). Il se prête bien à certains types d'algorithmes de programmation dynamique [4].

Les algorithmes de cette nature font appel à la notion d'étiquette. Une étiquette est un $R+1$-tuplet, où R est le nombre de ressources, associé à un nœud i et qui indique la valeur des ressources obtenues en suivant un chemin dans G^k liant le nœud source pour le pilote k, $o(k)$, au nœud i et se terminant au nœud puits $d(k)$. La composante $R+1$ correspond au coût réduit cumulé le long du chemin. On peut exprimer cette étiquette sous la forme (T_i^R, C_i). Chaque algorithme utilise la notion de dominance, où une étiquette est dite en dominer une autre si toutes ses composantes, y compris le coût réduit, sont inférieures à celles de l'autre étiquette. Quel que soit l'algorithme utilisé, les étiquettes qui sont dominées à un nœud sont toujours éliminées afin d'alléger la quantité d'étiquettes qui doivent être conservées durant la résolution.

Si ces méthodes sont implantées efficacement, il est possible d'obtenir de nouvelles colonnes ayant le potentiel d'améliorer l'objectif obtenu à l'itération suivante du RMP. La méthode du SPPRC permet donc en pratique d'implanter l'étape des sous-problèmes et de traiter les contraintes de ressources pilote par pilote, tel qu'est l'avantage de la décomposition de Dantzig-Wolfe. Il est à noter cependant que les solutions obtenues par les RMP ne sont pas forcément entières. Pour y parvenir, les problèmes de création d'horaires d'équipage tels que le CRP emploient souvent un algorithme de type *branch-and-price*.

Une présentation exhaustive des algorithmes de type *branch-and-price* est fournie dans Barnhart *et al.* [5] et Desrosiers *et al.* [6]. Leur fonctionnement est le suivant. D'abord, la relaxation linéaire du RMP initial est résolue jusqu'à ce qu'un critère d'arrêt soit atteint. Il ne s'agit pas nécessairement de l'optimalité, en raison du phénomène du *tail effect*. Un critère fréquent est d'exiger une certaine amélioration au cours des dernières itérations du simplexe réalisées. La solution n'étant pas forcément entière, une étape de branchement est effectuée. Il est alors possible de fixer une ou plusieurs colonnes, ou encore des paires d'activités consécutives. La première méthode est dite le branchement sur les colonnes, alors que la deuxième est le branchement intertâches.

L'idée d'un algorithme *branch-and-price* est donc de continuer à générer de nouvelles colonnes

après chaque étape de branchement. Le résultat est que l'algorithme alterne entre la résolution d'un RMP et la génération de colonnes par les sous-problèmes pendant quelques itérations jusqu'à ce que le critère d'arrêt prédéterminé soit atteint, et une étape de branchement.

1.3 Heuristiques d'amélioration

Agrégation dynamique de contraintes

La génération de colonnes telle qu'intégrée à un algorithme de *branch-and-price* permet souvent de résoudre les problèmes d'horaires d'équipage, dont le CRP, de façon efficace. Cependant, il arrive que mieux soit nécessaire. Cette réalité a mené la communauté scientifique à développer différents moyens d'y parvenir. Leur travail s'étend également au domaine de l'optimisation combinatoire de façon plus générale.

Le travail des chercheurs dans le domaine peut être catégorisé de deux façons différentes. D'abord, il y a les méthodes dites plus traditionnelles dont l'objectif est d'améliorer l'algorithme d'optimisation utilisé par le solveur. Il y a également des approches plus récentes qui utilisent des métaheuristiques ou encore l'apprentissage machine dans le but d'apprendre comment la structure du problème peut être exploitée de manière à aider le solveur à le résoudre plus efficacement, voire à résoudre le problème sans solveur correspondant à l'état de l'art. Pour ce qui est des approches traditionnelles, nous traitons ici les deux méthodes principales qui ont été développées au cours des dernières années : l'agrégation dynamique de contraintes (*dynamic constraint aggregation* ou DCA) et l'approche par fenêtrage ou fenêtres mobiles (ces méthodes sont aussi souvent appelées horizon fuyant ou roulant).

L'agrégation dynamique de contraintes est une méthode regroupant une famille d'algorithmes qui visent à réduire le problème de la dégénérescence lors de l'exécution du simplexe. En effet, lorsqu'un problème est de très grande taille dans un contexte de génération de colonnes, il n'est pas rare que plusieurs des variables basiques soient nulles, menant ainsi à de longs cycles retardant l'amélioration de l'objectif du modèle à optimiser. L'idée sous-jacente à l'agrégation dynamique de contraintes est que lorsque le nombre de contraintes est réduit, la taille de la base du problème l'est également. Ceci permet de diminuer la quantité de variables nulles qui s'y trouvent et la longueur des cycles de dégénérescence.

Pour ce faire, une solution initiale est fournie au solveur qui est utilisé pour résoudre le problème visé. Cette solution peut être produite de différentes façons, y compris à l'aide d'un modèle d'apprentissage machine. Les activités qui figurent ensemble dans cette solution, selon un critère prédéfini, forment alors des agrégats, chaque activité s'y trouvant étant jugée équivalente aux autres. Dans le cas du CRP, l'horaire de chaque pilote individuel trouvé

dans l'ensemble des blocs mensuels fournis initialement sert de point de référence, toutes les rotations dans un même horaire formant un tel agrégat. Lorsque des colonnes sont générées, il est alors possible d'évaluer si elles sont compatibles avec l'ensemble des agrégats, qui est appelé la partition des rotations. En particulier, on dit qu'une colonne est compatible avec cette partition si pour chacun de ces agrégats, toutes les activités qui s'y trouvent sont soit présentes, soit absentes de l'horaire représenté par la colonne. Lorsqu'une partie de ces éléments y figurent et d'autres non, la colonne est jugée incompatible avec la partition, et le degré d'incompatibilité peut être mesuré.

Ce principe permet d'éliminer des contraintes. Dans la version la plus simple de DCA, les contraintes pour les rotations faisant partie d'un même groupe sont agrégées. Quel que soit l'algorithme DCA retenu, seules les colonnes générées qui sont compatibles avec la partition décrite ci-haut peuvent alors être ajoutées au RMP à résoudre, alors que les colonnes incompatibles sont utiles pour déterminer si la partition doit être modifiée (et sont également ajoutées au problème complémentaire dans le cas du simplexe primal amélioré). Par exemple, si certaines colonnes incompatibles ont un coût réduit négatif, l'algorithme peut changer la partition pour que les colonnes incompatibles ne le soient plus.

Chaque type d'algorithme DCA doit aussi inclure un moyen d'obtenir une solution duale complète pour la résolution des sous-problèmes de génération de colonnes, les contraintes en nombre réduit donnant lieu à une solution agrégée ou encore incomplète, selon l'algorithme. Ce peut être fait par exemple à l'aide d'inégalités sur les variables duales créant un problème de plus courts chemins, tel que pour l'algorithme DCA dans [7]. Par contraste, la méthode du simplexe primal amélioré développée par Elhallaoui *et al.* [8] résout un problème complémentaire afin de déterminer les colonnes incompatibles avec la partition actuelle qui devraient être incluses dans le RMP suivant, et simultanément calculer une solution duale complète pour celui tout juste résolu.

Fenêtrage

Le fenêtrage, ou approche par fenêtres mobiles, est une technique visant à accélérer la résolution de problèmes de création d'horaires d'équipage en construisant la solution par parties. Parfois, une solution initiale est fournie à l'aide d'une heuristique simple et rapide. La durée des fenêtres, qui couvrent chacune un certain nombre de jours de l'horizon mensuel, est alors déterminée, ainsi que celle du chevauchement entre elles.

Avec ces conditions en place, l'algorithme de fenêtrage commence par le début du mois, et le problème est résolu de nouveau de façon à ne modifier que le contenu de la première fenêtre. Le reste des fenêtres est figée (à l'exception de la partie de la première fenêtre qui chevauche celle qui suit). La manière d'implanter cette restriction est de modifier la structure du sous-

problème pour chaque pilote en retirant les arcs qui permettraient de trouver un chemin avec contraintes de ressources dont le coût réduit serait négatif et qui écarterait les rotations ou les congés affectés au pilote en dehors de la fenêtre considérée. Le problème est donc techniquement résolu de façon complète, mais la quantité de chemins possibles dans chaque sous-problème est fortement réduite.

Une fois cette étape conclue, la solution obtenue est utilisée pour l'itération qui suit : cette fois, la deuxième fenêtre en ordre chronologique est considérée, alors que le reste de la solution est figé. La procédure décrite ci-dessus est alors répétée, et ainsi de suite pour l'ensemble des fenêtres jusqu'à ce qu'une solution finale soit obtenue.

L'avantage principal de cette méthode de résolution accélérée tient en la taille réduite des problèmes résolus dans chaque fenêtre. Comme dans chaque sous-problème de nombreux sous-chemins sont figés, la quantité des horaires pouvant être générés est grandement réduite. En outre, durant la mise en œuvre de l'algorithme de type *branch-and-price*, les RMP à chaque nœud de branchement sont plus petits, car il y a moins de contraintes. De même, les arbres de branchement utilisés pour obtenir une solution entière sont également moins profonds, car il y a moins de variables fractionnaires. Le résultat est donc que les temps de calculs diminuent plus que linéairement et que le processus de résolution complet est plus court.

Un inconvénient possible, cependant, est que la solution soit de moins bonne qualité que celle obtenue avec une résolution standard. En effet, les activités assignées à l'intérieur d'une fenêtre ne pouvant plus être modifiées aux étapes ultérieures de cette procédure, il est possible que les valeurs des ressources consommées pour leur affectation empêchent celle de tâches faisant partie de la solution optimale, mais survenant plus tard dans l'horizon, sans qu'aucune contrainte ne soit brisée.

Apprentissage machine

Les dernières années ont vu un intérêt croissant se développer envers l'apprentissage machine comme outil pour améliorer la résolution de problèmes d'optimisation combinatoire. L'idée derrière cette méthode est d'utiliser un modèle, tel qu'un réseau neuronal, un arbre de décision, ou la régression linéaire, et d'entrainer ce modèle sur un jeu de données à l'aide d'un algorithme permettant de mettre en évidence des tendances qui peuvent être généralisées à d'autres données. Un résumé exhaustif de l'état de l'art pour cette approche se trouve dans Bengio, Lodi et Prouvost [9].

De manière générale, l'apprentissage machine appliqué au domaine de l'optimisation peut être catégorisé en trois types de méthodes. La première est l'apprentissage de bout en bout. Cette méthode cherche à produire une solution au problème traité comparable à l'état de

l'art, et sans l'utilisation d'un solveur traditionnel. La deuxième approche est l'apprentissage du prétraitement. Ce type vise à apprendre de la structure du problème et des données pour fournir une information utile à un solveur correspondant à l'état de l'art. Sur la base de cette information, le solveur poursuit ensuite la résolution de façon indépendante. Une forme fréquente de contenu fourni au solveur dans ce contexte est une solution initiale pouvant être améliorée par ce dernier. Enfin, l'apprentissage en boucle utilise les résultats de l'entrainement du modèle de façon interactive avec le solveur. Le modèle intervient tout au long du processus de résolution pour aider l'algorithme d'optimisation à faire de meilleurs choix, par exemple lors d'étapes de branchement.

Chaque type d'algorithme ML cherche à apprendre à déterminer de l'information de nature différente. Les méthodes de bout en bout cherchent souvent à prédire de façon précise ce qui se retrouve dans une solution optimale pour que la solution créée par le modèle en soit proche. Les méthodes en boucle cherchent plutôt à détecter des tendances plus générales pour aider le comportement du solveur, sans nécessairement tenter de reproduire une solution particulière. Pour ce qui est de l'apprentissage du prétraitement, ces deux comportements peuvent être observés selon la méthode particulière en fonction, par exemple, de la sensibilité de l'approche à la qualité de la solution initiale fournie. Le type d'information recherchée fait une différence sur le type de problèmes qu'une technique donnée peut traiter de manière satisfaisante.

Dans le cas du CRP tel que traité dans cette thèse, on cherche à apprendre une solution approximative qui peut ensuite être fournie à un solveur standard. Dans certains cas, comme pour l'agrégation dynamique de contraintes, cette solution contient idéalement des parties d'une solution optimale. L'approche utilisée est donc une forme d'apprentissage machine du prétraitement. Comme cette solution peut être fournie en entrée à différentes méthodes visant à accélérer le CRP, elle permet de comparer leur comportement.

Il est à noter que de nombreuses autres heuristiques ont été développées pour essayer de résoudre le CRP de façon plus efficace. Ces techniques sont souvent de bout en bout, mais sans faire un appel à l'apprentissage machine. Il s'agit essentiellement de métaheuristiques, tels différents algorithmes évolutifs ou formes de réoptimisation locale, par exemple.

1.4 Contributions

Nous formulons maintenant les contributions principales apportées par ce travail de recherche. Nous présentons le problème de création des blocs mensuels personnalisés comme une étude de cas approfondie des façons dont l'apprentissage machine peut améliorer la résolution d'un problème de recherche opérationnelle (RO), ou plus généralement d'optimisation combina-

toire. La variété des méthodes testées permet en effet de comparer différentes approches et de déterminer lesquelles donnent lieu à des améliorations ou non. Les résultats obtenus peuvent alors être mis en relation avec ceux observés dans la littérature du domaine. De façon plus exhaustive, nous pouvons considérer les contributions concrètes apportées par ce travail comme étant les suivantes.

Au chapitre 4, nous développons une procédure d'affectation successive (nommée *seqAsg*) entrainée par ML et qui fournit une solution rapide au problème des blocs mensuels personnalisés. Cette méthode simplifie le problème présenté à la section 1.1 en divisant l'horizon mensuel en jours individuels pour lesquels un problème d'affectation est établi pour les pilotes, rotations et autres activités disponibles le jour même. Chaque affectation potentielle se voit attribuer une valeur calculée à l'aide du modèle d'apprentissage machine, ici un réseau de neurones de petite taille avec deux couches cachées. Cette méthode est alors utilisée pour prédire le comportement d'un solveur correspondant à l'état de l'art : la qualité des solutions générées par cette approche en quelques secondes est corrélée avec un R^2 de 0,919 avec celles des solutions du solveur (ici GENCOL version 4.5). Cette relation est plus solide que ce qui est obtenu avec des méthodes alternatives et permet aux planificateurs de déterminer un effectif de pilotes équilibrant les coûts de personnel de la compagnie aérienne avec la satisfaction des pilotes.

Au chapitre 5, la méthode d'affectation successive est réutilisée pour générer des solutions. Une première version d'une approche de résolution par fenêtrage est alors créée dans GENCOL, car contrairement à d'autres problèmes tels que la création de rotations d'équipage ou les tournées de véhicules, le problème des blocs mensuels personnalisés n'a jamais été résolu de cette façon. Ceci constitue une contribution en soi, car il n'était pas évident que cette approche se prêterait bien à un horizon d'un mois et aux contraintes complexes qui y sont associées. De plus, nous observons que les solutions générées par affectation successive, et donc par apprentissage machine, parviennent à résoudre le problème en un temps comparable au fenêtrage sans ML, soit un temps inférieur à 10% de ce qui est requis par GENCOL standard, mais avec un meilleur objectif. En particulier, nous obtenons moins de 1% de perte d'objectif par rapport à GENCOL sur différentes instances de complexité variable, et que cette perte récupère 33,3% de l'erreur due au fenêtrage sans ML.

Finalement, le chapitre 6 traite de l'agrégation dynamique de contraintes. En particulier, il y est montré qu'utiliser l'apprentissage machine dans le cadre de cette approche ne se prête pas bien au CRP. En effet, les solutions initiales données par l'affectation successive ne donnent pas le niveau de précision typiquement associé à l'exécution idéale de cette procédure du point de vue de l'accélération de la résolution. Ceci est notamment dû au fait que la solution initiale

est fortement incompatible avec la partition obtenue à partir d'une solution correspondant à l'état de l'art, même lorsque les objectifs sont proches dans les deux cas. Cinq méthodes diverses sont également testées pour tenter de remédier à ce problème, sans que ce dernier soit éradiqué. À l'aide d'une revue des applications ML à d'autres problèmes dans la littérature, il est montré que ces difficultés sont contournées lorsque certaines particularités du problème peuvent être exploitées, comme dans le cas du problème de rotations d'équipage, où il est plus aisé de prédire le vol suivant si on part du fait que les membres d'équipage suivent généralement le même appareil.

Le chapitre 6 permet donc de contraster les difficultés qui y sont rencontrées et les succès des deux précédents, de sorte que des recommandations peuvent être effectuées. On privilégie notamment l'apprentissage fournissant de l'information plus générale que celui visant la réplication d'une solution optimale, dans la mesure où les propriétés du problème résolu ne se prêtent pas à cette deuxième approche. Ce faisant, nous contribuons à enrichir la littérature où cette tendance semble se dessiner dans divers contextes d'optimisation combinatoire à l'aide d'une étude de cas approfondie pour le CRP qui contient en elle-même différentes méthodes pour en accélérer la résolution.

CHAPITRE 2 REVUE DE LITTÉRATURE

Ce chapitre vise à décrire l'ensemble du travail qui a été réalisé par des chercheurs de différents domaines des mathématiques appliquées, tels que la recherche opérationnelle, l'optimisation combinatoire et l'apprentissage machine et sur lequel cette thèse vise à s'établir. Nous traitons les différents aspects sur lesquels les chapitres 4 à 6 s'appuient séparément. Nous commençons par mentionner la littérature qui décrit les problèmes liés à la création d'horaires d'équipage aérien, y compris les blocs mensuels personnalisés, ainsi que la génération de colonnes. Nous traitons ensuite de ce qui se rapporte aux deux méthodes principales employées, soit l'agrégation dynamique de contraintes et le fenêtrage. Enfin, nous traitons la littérature touchant la question de l'apprentissage machine, notamment l'apprentissage de bout en bout, du prétraitement, ou en boucle, et des métaheuristiques qui visent à améliorer la résolution du CRP.

2.1 Formulations et modélisation

La création d'horaires personnalisés comptant deux étapes, soit la création de rotations et celle de blocs mensuels, nous indiquons la littérature essentielle se rapportant à ces problèmes.

Desrosiers *et al.* [10] ont introduit le solveur d'optimisation mathématique GENCOL, dont l'apport principal est de permettre la résolution de problèmes de création d'horaires de grande taille dans divers contextes, notamment pour le problème des rotations d'équipage. En lien avec cette contribution, Desaulniers *et al.* [11] ont publié les résultats d'une première implantation du problème des rotations pour le compte d'une compagnie aérienne de grande envergure, Air France. Les résultats obtenus sont tels que le coût des solutions est à une fraction de 1 % de l'optimalité, soit bien moindre que pour les rotations obtenues autrement. Barnhart, Hathay et Johnson [12] ont présenté une méthode qui améliore la résolution du problème des rotations d'équipage en sélectionnant de façon efficace des vols de repositionnement, soit des vols où les membres d'équipage sont relocalisés d'un aéroport à l'autre en tant que passagers pour pouvoir assurer un départ ailleurs. Fondée sur la génération de colonnes et un algorithme de type *branch-and-price*, cette approche donne des solutions qui dépassent en qualité les recommandations d'experts avec notamment un écart d'intégralité correspondant à une fraction de 1 %.

Certains travaux ont eu pour but une synthèse des modèles possibles pour la création d'horaires d'équipage. Kohl et Karisch [1] ont présenté les différentes formulations apparaissant

fréquemment pour le problème des blocs mensuels aériens. Ils donnent entre autres les types de critères considérés par l'objectif du modèle d'optimisation, ainsi qu'une catégorisation des contraintes fréquentes comme étant verticales, horizontales ou globales. Gopalakrishnan et Johnson [13] ont quant à eux résumé l'état de l'art en ce qui a trait aux aspects fondamentaux de la création de rotations ou de blocs mensuels aériens.

Nous notons également le travail de Caprara *et al.* [14], qui donnent une formulation générale du problème des blocs mensuels ferroviaires. Il décrivent deux types de contraintes caractérisant le problème, soit les contraintes séquentielles et opérationnelles, et fournissent une heuristique de résolution basée sur l'exploitation de bornes inférieures lagrangiennes.

Types de problèmes de blocs mensuels

Les blocs mensuels peuvent être conçus anonymement ou par création d'horaires personnalisés avec ou sans séniorité. Nous donnons ici des exemples représentatifs de chaque approche.

Boubaker, Desaulniers et Elhallaoui [15] ont traité le cas des blocs anonymes. Cette façon de faire est particulièrement courante chez les compagnies aériennes nord-américaines. Les auteurs définissent un modèle dont l'objectif cherche à minimiser les variations de temps de vol total et de jours de congé entre les horaires fabriqués. Cependant, comme il est habituel pour cette conceptualisation du problème des blocs mensuels, les horaires finalisés sont choisis par les pilotes en ordre décroissant d'ancienneté. En outre, on peut citer les travaux de Jarrah et Diamond [16], Weir et Johnson [17], et Christou *et al.* [18]. Dans les deux derniers cas, l'objectif tient compte d'une mesure de pureté des rotations, où la pureté est définie comme le fait de contenir plusieurs caractéristiques généralement reconnues comme désirables par les membres d'équipage.

Pour ce qui est de la méthode de résolution avec horaires personnalisés, Gamache *et al.* [19] donnent un exemple classique de cette approche. L'objectif de leur modèle maximise la somme des satisfactions pour les agents de bord ainsi que la durée totale des rotations desservies par eux. Leur approche est implantée en tenant compte de la façon de faire d'Air France. En ce sens, leur travail est typique de l'approche européenne, qui privilégie généralement les horaires personnalisés. Un autre exemple phare est le travail de Kasirzadeh, Saddoune et Soumis [20]. Dans ce cas spécifique, le critère d'optimisation retenu est la somme des satisfactions de tous les employés. Leur contribution établit également un jeu de données composé de sept instances différentes. Ces dernières sont devenues une référence fréquente pour comparer l'efficacité de différentes méthodes pour le problème des horaires personnalisés.

De plus, Moudani *et al.* [21] formulent un objectif bi-critère constitué de la satisfaction des pilotes et du coût des solutions pour la compagnie aérienne. On peut également mentionner le

travail de Dawid, König et Strauss [22], qui introduit la notion de rétrogradation permettant à certains membres d'équipage expérimentés d'effectuer des tâches exigeant un niveau moins élevé de qualifications lorsque requis par un problème serré.

La question de la méthode par blocs personnalisés avec séniorité fut traitée par Gamache *et al.* [23]. L'approche qu'ils présentent traite les pilotes par ordre d'ancienneté. Un bloc personnalisé est alors construit pour le pilote ayant le plus d'ancienneté en tenant compte de ses préférences. Le pilote suivant se voit attribuer un problème résiduel qui consiste à créer l'horaire le plus adapté à ses préférences et en s'assurant que le problème reste réalisable pour les suivants. La méthode poursuit ainsi pour tous les pilotes. Achour *et al.* [24], pour leur part, fournissent le premier algorithme de résolution exacte pour cette version du problème des blocs mensuels. Leur algorithme prend en compte la situation où il y a plusieurs solutions de même satisfaction pour un pilote. Ces méthodes utilisent la génération de colonnes et ont été testées sur plusieurs instances réelles.

Affectation d'horaires intégrée

Certains auteurs ont développé des modèles résolvant le problème des rotations d'équipage et celui des blocs mensuels de façon simultanée ou intégrée. Guo *et al.* [25] ont développé un modèle créant des chaînes de rotations intercalées de repos hebdomadaires et prenant en compte toutes les activités préassignées aux pilotes d'une base donnée. Elles sont construites de façon à ce que le temps de vol varie peu entre les chaînes. Les blocs mensuels sont alors construits à l'aide des chaînes obtenues. Saddoune *et al.* [26] utilisent l'agrégation dynamique de contraintes pour résoudre le problème des rotations et celui des blocs mensuels anonymes. Ils trouvent que cette méthode donne lieu à des gains de coût de 3,37 % par rapport à la méthode séquentielle en contrepartie de temps de calculs accrus.

De plus, Zeighami et Soumis [27] de même que Zeighami, Saddoune et Soumis [28] ont créé deux modèles de résolution exploitant la génération de colonnes à l'aide des décompositions de Benders et lagrangienne, respectivement. Ces méthodes permettent d'obtenir un meilleur objectif et, dans le deuxième cas, des temps de calculs moins grands que pour d'autres approches intégrées grâce à DCA. Tous les exemples cités ont également été testés en pratique au service de grandes compagnies aériennes nord-américaines ou européennes.

2.2 Méthodes de résolution

De nombreuses méthodes traditionnelles ont été créées afin de résoudre les problèmes d'horaires d'équipage. On peut mentionner les éléments se rapportant à la génération de colonnes et à la résolution par *branch-and-price*, à l'agrégation dynamique de contraintes et au fenê-

trage. Nous traitons chaque point séparément.

Génération de colonnes

Plusieurs articles ont donné les fondements de la résolution par génération de colonnes de tâches complexes liées à la création de rotations et de blocs mensuels. Par exemple, Desrosiers *et al.* [4] présentent de nombreux problèmes de création tant d'itinéraires que d'horaires. Ils ont donné les algorithmes de programmation dynamique qui peuvent être utilisés pour résoudre les problèmes de plus court chemin avec contraintes de ressources qui sont utiles lors de la résolution des sous-problèmes liés à la décomposition de Dantzig-Wolfe. Ils intègrent aussi la génération de colonnes dans un arbre *branch-and-bound* pour obtenir une solution optimale entière. Desaulniers *et al.* [2] ont développé une formulation unifiée des problèmes de tournées de véhicules et de création d'horaires d'équipage tenant compte des fenêtres de ressources et utilisant des variables portant sur les arcs et le flot qui les traverse. Barnhart *et al.* [5] donnent le nom de *branch-and-price* à l'intégration de la génération de colonnes dans un arbre *branch-and bound*, et Desrosiers *et al.* [6] en donnent une description approfondie.

Lübbecke et Desrosiers [29] ont également répertorié plusieurs considérations pratiques associées à l'implantation efficace d'une approche avec génération de colonnes. Ils traitent entre autres différentes formes de décomposition, d'algorithmes de résolution, des difficultés computationnelles pouvant être réduites avec des techniques de stabilisation, et de l'obtention d'une solution entière.

Agrégation dynamique de contraintes

Sur la base de l'article présentant l'agrégation dynamique de contraintes d'Elhallaoui *et al.* [7], plusieurs améliorations ont été apportées à cette méthode, et les algorithmes DCA ont été utilisés dans différents contextes également.

Dans Elhallaoui *et al.* [30], une version dite multiphase de l'algorithme de base est développée. Elle ajoute une ressource d'incompatibilité qui permet de s'assurer que les colonnes de coût réduit négatif générées par les sous-problèmes n'aient pas un score d'incompatibilité supérieur à celui prédéterminé par la phase en cours. Le score peut ainsi être considéré comme une mesure de distance par rapport à la solution courante. Le résultat est qu'au cours des premières phases, les changements de partition ne sont basés que sur des colonnes légèrement incompatibles avec celle de l'itération courante, réduisant ainsi le risque d'une désagrégation extrême ou précoce de la partition.

Elhallaoui *et al.* [8] ont créé une version améliorée de l'algorithme du simplexe primal où les colonnes incompatibles avec la partition actuelle à intégrer à la base du simplexe sont obtenues en résolvant un problème complémentaire plutôt que des problèmes de plus court chemin.

Zaghrouti, Soumis et El Hallaoui [31] utilisent également un problème réduit et un problème complémentaire. Leur approche peut être considérée comme un algorithme du simplexe primal en nombres entiers : les pivots mènent d'une solution entière à une autre, tandis que le problème complémentaire permet de déterminer un ensemble de pivots permettant de passer à une autre solution entière après quelques itérations de dégénérescence.

Les algorithmes de la famille DCA ont été utilisés avec succès dans différents contextes. On peut notamment mentionner les articles de Boubaker, Desaulniers et Elhallaoui [15] et Saddoune *et al.* [26] mentionnés précédemment. Yaakoubi, Soumis et Lacoste-Julien [32, 33] appliquent également un algorithme DCA multiphase prenant en entrée une solution initiale au problème des rotations d'équipage obtenue par apprentissage machine. Ils parviennent à accélérer la résolution de ce problème par un facteur de 10 en utilisant cet algorithme.

Fenêtrage

De nombreuses applications appartenant à des domaines variés ont utilisé une approche par fenêtrage pour accélérer la résolution du problème visé. Cette approche divise un horizon temporel en fenêtres plus courtes. Le problème est alors résolu une fenêtre à la fois et séquentiellement. Saddoune, Desaulniers et Soumis [34] utilisent le fenêtrage dans le contexte du problème des rotations d'équipage. Ils obtiennent que les temps de calculs et le coût des solutions sont améliorés par cette approche par contraste avec une résolution traditionnelle en trois phases (quotidienne, hebdomadaire, puis à l'échelle d'un mois complet).

Nielsen [35] a également étudié les effets d'une approche par fenêtrage pour le problème de réordonnancement du matériel roulant pour les trains à passagers. Il a trouvé que le fenêtrage permet de résoudre ce problème de façon plus efficace que les modèles traditionnels en cas de perturbations du réseau. De façon analogue, Abdelghany, Abdelghany et Ekollu [36] utilisent le fenêtrage pour la réoptimisation d'horaires d'équipage aérien à la suite de perturbations subites. Ils ont traité la réallocation de ressources en moins d'une minute et en permettant d'obtenir de nouveaux horaires avec le minimum de retards et d'annulations possible. Gkiotsalitis et Van Berkum [37] se concentrent quant à eux sur le problème de l'heure de départ des autobus urbains. À l'aide du fenêtrage, ils arrivent à redéfinir les heures de départ des autobus du réseau en quelques secondes selon les besoins. Jaillet *et al.* [38] introduisent une méthode avec fenêtrage pour le problème de gestion des itinéraires et des stocks, ici de l'huile de chauffage, dont la demande varie quotidiennement. Ces applications soulèvent la rapidité potentielle d'une approche par fenêtrage dans le cas où une réoptimisation est nécessaire.

Hormis ces problèmes, le fenêtrage a entre autres été utilisé pour effectuer la tournée de l'entretien préventif d'avions. C'est notamment le cas pour le travail d'Afsar, Espinouse et Penz [39], ainsi que pour Basdere et Bilge [40]. Qi, Bard et Yu [41] utilisent pour leur part le

fenêtrage pour résoudre le problème de la conception d'horaires de formation pour les pilotes se voyant attribuer un nouveau type de tâche. L'objectif est alors de minimiser la durée totale des cours imposés aux pilotes. Ce processus a été appliqué chez Continental Airlines, aujourd'hui intégré à la compagnie United Airlines. Secomandi et Margot [42] ont aussi traité le problème de tournées de véhicules avec demandes stochastiques.

Le fenêtrage a également été utilisé dans le contexte de diverses industries. Par exemple, Bostel *et al.* [43] se servent de cette technique pour créer de meilleurs horaires pour les visites de techniciens travaillant pour le réseau de distribution de l'eau courante. Millar [44] s'intéresse à l'industrie de la pêche industrielle de manière à expédier les flottes commerciales en tenant compte des fluctuations du marché. Sherali, Al-Yakoob et Hassan [45] appliquent une technique de fenêtrage dans le cadre de l'industrie du pétrole de manière à construire la tournée et les horaires d'équipage. On peut également mentionner le travail d'Alimian *et al.* [46] sur le problème de dimensionnement de lots avec capacité. Nous notons qu'au meilleur de notre connaissance, le problème des blocs mensuels personnalisés n'a pas été traité dans la littérature avec l'aide du fenêtrage. Au minimum, cette méthode n'avait pas été implantée pour le CRP dans le solveur que nous présentons à la section 4.4.

2.3 Apprentissage machine

L'apprentissage machine a connu un véritable essor au cours des 10 dernières années en tant que moyen d'améliorer la résolution de problèmes d'optimisation combinatoire, y compris la création d'horaires d'équipage. Nous traitons ici les trois types d'interaction ML-RO les plus fréquents, soit l'apprentissage de bout en bout, l'apprentissage par prétraitement et l'apprentissage en boucle.

Nous mentionnons les contributions dans la littérature qui sont les plus pertinentes pour notre travail. Cependant, il existe plusieurs articles de synthèse qui donnent une vue d'ensemble de ce qui a été fait dans ce domaine. Bengio, Lodi et Prouvost [9] et Scavuzzo *et al.* [47] donnent un aperçu des normes de pratique pour l'interaction entre ML et RO. Ils mentionnent également certaines considérations qui y sont propres, y compris le besoin de générer des solutions réalisables, de choisir un modèle approprié à des problèmes structurés tels que ceux de l'optimisation combinatoire, la généralisabilité à des problèmes de grande échelle, et la génération de données. Karimi-Mamaghan *et al.* [48] ont dressé la liste des recherches ayant utilisé des modèles ML pour améliorer la capacité de métaheuristiques à résoudre un problème d'optimisation combinatoire. Mazyavkina *et al.* [49] ont fourni une revue exhaustive de l'usage d'algorithmes d'apprentissage par renforcement (*reinforcement learning* ou RL) aux applications d'optimisation combinatoire, les différentes catégories de modèles pouvant

être supervisé, non-supervisé ou par renforcement.

Apprentissage de bout en bout

En ce qui a trait à l'apprentissage de bout en bout, plusieurs contributions peuvent être citées. Un exemple pionnier en la matière correspond au travail de Bello *et al.* [50], où l'article présente un algorithme RL pour créer une solution quasi-optimale au problème du voyageur de commerce (*travelling salesman/salesperson problem* ou TSP) pour des problèmes de petite taille (allant jusqu'à 100 nœuds). Cette solution est obtenue sans l'aide d'un solveur traditionnel. Nazari *et al.* [51] ont étendu ce travail à l'aide d'un mécanisme d'attention, accélérant ainsi la création de solutions pour le problème du voyageur de commerce. Ils ont également généralisé leur modèle au VRP. Le travail de Kool, van Hoof et Welling [52] s'inscrit également dans cet ordre d'idées, où un mécanisme d'attention est utilisé pour les problèmes du TSP avec ou sans tarification, du VRP, et des tournées sélectives.

Khalil *et al.* [53] développent un autre algorithme RL qui représente chaque problème traité par un graphe dont la topologie est alors reformulée sous la forme d'un vecteur obtenu par plongement. Le modèle entrainé par l'algorithme est un réseau de neurones en graphes (*graph neural network* ou GNN). Ils traitent la couverture de sommets minimum, le problème de coupe maximum, et le TSP. Les auteurs obtiennent de bonnes solutions pour ces problèmes en comparaison avec l'état de l'art, bien qu'il y ait un pourcentage d'écart avec la valeur optimale significatif pour le problème du voyageur de commerce. Par exemple, ils obtiennent un tour final d'une longueur excédant le minimum de 10,75 % en moyenne pour les instances allant de 1000 à 1200 nœuds, soit la taille la plus grande considérée. De plus, Nowak *et al.* [54] ont aussi utilisé un GNN dans le cadre du problème d'affectation quadratique.

L'apprentissage machine à l'aide de ce type de modèle est particulièrement pertinent pour les problèmes d'optimisation combinatoire en raison de leur complexité et de leur nature structurée qui se prête souvent bien à une représentation graphique. Dans un contexte d'apprentissage de bout en bout, cependant, la généralisation d'échelle, notamment à des problèmes de plus grande taille, est une préoccupation importante. Les GNN étant complexes, ils peuvent facilement devenir intraitables à grande échelle.

Par exemple, Cappart *et al.* [55] décrivent de façon approfondie l'état de l'art pour cette classe de modèles, et indiquent qu'un défi actuel est la généralisation de leurs capacités pour les graphes denses. Joshi, Laurent et Bresson [56] utilisent un GNN pour résoudre le TSP et dépassent l'état de l'art pour les méthodes d'apprentissage dans des cas allant jusqu'à 100 nœuds, mais n'égalent pas la performance des solveurs traditionnels. Ils expriment aussi l'importance de transférer les capacités de leur approche à des réseaux plus grands. Joshi *et al.* [57] font écho à ce constat, et mettent l'accent sur l'importance de repenser

la généralisation de la résolution du TSP. Ils développent un processus pour évaluer les techniques et heuristiques plus susceptibles de permettre le passage de l'apprentissage réalisé sur de petits réseaux à d'autres plus grands ou plus denses. Ces limites sont également examinées pour l'apprentissage de bout en bout de façon plus générale dans le travail de Glasmachers [58] et Angelini et Ricci-Tersenghi [59].

Apprentissage du prétraitement

L'apprentissage du prétraitement consiste à apprendre à fournir de l'information judicieuse en entrée à un solveur traditionnel avant son lancement. Ce dernier peut alors l'exploiter de façon à résoudre le problème cible plus efficacement, par exemple de manière accélérée en sacrifiant peu ou pas de qualité de l'objectif du modèle d'optimisation. La littérature concernant ce type d'interaction ML-RO est assez peu abondante, mais il existe dans certains cas le potentiel d'améliorer significativement la résolution exécutée par un solveur traditionnel en seulement quelques secondes.

Kruber, Lübbecke et Parmentier [60] ont développé un modèle d'apprentissage supervisé qui inspecte la structure de problèmes mixtes en nombres entiers (*mixed integer program* ou MIP) afin de détecter si l'usage d'une décomposition, telle que la décomposition de Dantzig-Wolfe, peut accélérer la résolution du problème. Leur algorithme permet également de choisir une décomposition particulière quand plusieurs sont détectées comme prometteuses. Les résultats obtenus montrent que le modèle ML prédit de manière fiable si une décomposition est appropriée et laquelle l'est le plus, de sorte que ce sous-ensemble de problèmes sont résolus plus rapidement alors que les autres sont peu affectés.

Bonami, Lodi et Zarpellon [61] ont pour leur part traité le cas de problèmes mixtes quadratiques en nombres entiers (*mixed integer quadratic program* ou MIQP). Dans certains cas, il peut être avantageux de linéariser la partie quadratique de l'objectif. À l'aide de variables explicatives liées à différents aspects de la structure du problème ciblé, ils entrainent un classificateur par apprentissage supervisé afin de déterminer quand la résolution d'un problème peut effectivement être accélérée par cette linéarisation. La performance du classificateur est jugée satisfaisante pour la catégorie de problème testés.

Beulen, Scherp et Santos [62] se sont penchés sur le problème des requêtes de vols spécifiques effectuées par les pilotes dans le cadre du problème des blocs mensuels personnalisés. Ils ont utilisé un modèle d'apprentissage supervisé qui prend la forme d'un réseau de neurones. Cette méthodologie leur permet de prédire les requêtes qui devraient être acceptées. Après que cette information lui ait été transmise, le solveur traditionnel parvient à accorder 22 % de requêtes de plus qu'autrement, sans affecter la qualité des solutions construites.

Quesnel *et al.* [63] ont aussi traité le CRP. Ils ont utilisé un algorithme d'apprentissage supervisé pour entrainer un réseau de neurones qui prédit si une rotation est susceptible de faire partie de la solution optimale d'un pilote. Pour chaque pilote, un sous-ensemble de rotations qui vise à respecter le plus possible ces classifications est alors retenu comme l'ensemble réduit des rotations permises à l'intérieur de son sous-problème. Ces sous-ensembles permettent de réduire la quantité de variables données en entrée au solveur traditionnel, de sorte que la taille du problème est réduite et que sa résolution est accélérée au coût de légères pertes pour la valeur de l'objectif.

Pour leur part, Morabit, Desaulniers et Lodi [64] testent plusieurs modèles d'apprentissage supervisé dans le contexte des problèmes de la création d'horaires de chauffeurs d'autobus et du VRP avec fenêtres de temps. Ils prédisent quels arcs du réseau de chaque sous-problème sont susceptibles de figurer à l'intérieur d'une solution optimale. Uniquement les arcs prometteurs sont conservés, réduisant ainsi la taille des problèmes et accélérant leur résolution par 27 % et 40 %, respectivement.

On peut également noter les travaux de Yaakoubi, Soumis et Lacoste-Julien [32, 33]. Ces contributions fournissent une solution initiale à un solveur qui utilise l'agrégation dynamique de contraintes pour accélérer la résolution du problème des rotations d'équipage. La qualité de la solution initiale fournie au solveur permet de résoudre le problème mieux qu'une solution réalisable générique.

Apprentissage en boucle

L'apprentissage en boucle cherche à interagir avec un solveur traditionnel afin de lui apporter de l'information utile. La littérature dans ce domaine est plutôt abondante, et un certain nombre d'applications ont été traitées avec succès.

Gasse *et al.* [65] ont développé un modèle utilisant un GNN où le graphe représente un MIP. Les graphes sont bipartis et représentent les variables et contraintes ainsi que la façon dont elles sont liées. Le modèle est entrainé à sélectionner quelles variables fixer par branchement tout au long du processus de résolution. Les résultats obtenus ont été démontrés comme généralisables à des problèmes de plus grande taille que ceux utilisés pour l'entrainement.

Balcan *et al.* [66] ont montré comment l'apprentissage machine peut être utilisé pour pondérer différentes stratégies de branchement afin d'explorer l'arbre de décision plus efficacement pour les problèmes en nombres entiers. Le domaine d'applicabilité fait en sorte que le poids accordé à chaque stratégie varie. Le travail de Gupta *et al.* [67] s'inscrit également dans cette lignée. Pour un résumé plus approfondi de l'usage de techniques d'apprentissage machine pour le branchement lors de la résolution de problèmes en nombres entiers en général, le travail de

Lodi et Zarpellon [68] est une ressource plus complète.

Tang, Agrawal et Faenza [69] se sont également intéressés à la création d'un algorithme RL dont le but est d'aider l'obtention d'une solution entière à l'aide de coupes plus judicieuses. Leur travail porte sur différents problèmes et s'ajoute donc au corpus des contributions étudiant les capacités de l'apprentissage en boucle à faire du branchement plus efficace.

Pour ce qui touche plus directement aux problèmes de recherche opérationnelle liés aux transports, on peut également citer plusieurs exemples d'applications de ce type d'apprentissage. Furian *et al.* [70] ont développé une heuristique qui permet de prédire quels nœuds de l'arbre de branchement doivent être explorés, ainsi que les variables qui doivent être fixées dans le cadre du VRP. Morabit, Desaulniers et Lodi [71] utilisent l'apprentissage supervisé dans le contexte des problèmes de la création d'horaires de chauffeurs d'autobus et du VRP avec fenêtres de temps. Ils apprennent à sélectionner des colonnes prometteuses lors de la résolution de chaque RMP et réduisent ainsi le temps de résolution de 30 % pour ces problèmes. Pereira *et al.* [72] ont appliqué au problème des rotations d'équipage l'apprentissage pour branchement amélioré comme dans les contributions ci-dessus. Ils obtiennent de meilleures solutions en des temps comparables à ce qu'une heuristique de branchement myope produit. On peut également citer le travail de Václavík *et al.* [73] ainsi que celui de Nillius [74].

2.4 Métaheuristiques

Nous traitons dans cette section des contributions qui ont cherché à améliorer la résolution du problème des blocs mensuels personnalisés ou d'autres qui lui sont similaires à l'aide de métaheuristiques n'appartenant pas au domaine de l'apprentissage machine. Ces heuristiques sont essentielles à prendre en compte en ce qu'elles donnent souvent de bons résultats et doivent servir de point de comparaison aux méthodes d'apprentissage machine. On peut ainsi vérifier si l'amélioration observée grâce au ML dépasse celle d'heuristiques d'autres natures ou non.

Une des premières contributions dans ce domaine fut présentée par Ozdemir et Mohan [75]. Leur approche visait à établir le potentiel de l'usage d'un algorithme génétique pour la résolution du problème des rotations d'équipage. Elle est comparée à une heuristique gloutonne et à un autre algorithme génétique. La méthode proposée donne des solutions avec une répartition plus égale du travail, nécessitant moins de membres d'équipage et moins de vols de repositionnement (*deadhead*). Elle est suggérée comme une alternative possible à la génération de colonnes.

Lucic et Teodorovic [76, 77] ont également été parmi les premiers à examiner l'effet d'utiliser

différentes métaheuristiques, telles que le recuit simulé, un algorithme génétique, ou une recherche taboue, sur la résolution du problème des blocs mensuels avec horaires personnalisés. Ils obtiennent leurs meilleurs résultats avec le recuit simulé en un temps de calcul moyen de 4 minutes pour des instances allant jusqu'à 65 pilotes.

Armas *et al.* [78] ont considéré le problème des blocs mensuels personnalisés. Ils utilisent une heuristique randomisée à plusieurs départs (*multistart*) sur de petites instances en cherchant à équilibrer la charge de travail des pilotes. Leur travail s'inspire de celui de Juan *et al.* [79], qui présentent des heuristiques similaires pour des instances du VRP allant jusqu'à 200 nœuds. À l'aide de cette heuristique, ils parviennent à générer des solutions au CRP à l'intérieur de quelques secondes tout en tenant compte d'une plus grande variété de contraintes verticales et horizontales que pour l'état de l'art. Les solutions obtenues peuvent donc être utilisées par les compagnies aériennes avec peu d'ajustements en comparaison avec les métaheuristiques ou la génération de colonnes utilisées dans le cas où les types de contraintes du problème sont particulièrement nombreux.

Souai et Teghem [80] ont traité le problème de la création d'horaires d'équipage intégrée à l'aide d'un algorithme génétique hybride. Ils ont présenté trois heuristiques différentes pour améliorer la qualité de cet algorithme et ont testé leur méthode sur trois instances fournies par Air-Algérie.

Deng et Lin [81] ont implanté une heuristique d'optimisation par colonie de fourmis. Ils appliquent cette méthode sur de petites instances du problème de création d'horaires d'équipage et montrent que les temps de calculs requis sont plus faibles en moyenne que pour un algorithme génétique. Deveci et Demirel [82] comparent différents algorithmes évolutifs à un algorithme génétique pour le problème des rotations d'équipage et trouvent qu'un algorithme mimétique offre la meilleure performance. Saemi *et al.* [83] ont également réalisé une analyse similaire pour le CRP et ont trouvé qu'une heuristique avec optimisation par essaims particulaires donnait de meilleurs résultats qu'un algorithme génétique.

Maenhout et Vanhoucke [84] ont présenté un algorithme de recherche par échantillonnage pour la création de blocs mensuels. Leurs instances s'inspiraient de celles observées chez Brussels Airlines et comptaient de 100 à 140 pilotes ainsi que 700 activités (dont 600 rotations) pour une période de deux semaines. Ils ont trouvé des solutions en moyenne avec un écart d'optimalité de 1,67 % pour les solutions optimales trouvées par un algorithme de type *branch-and-price* standard. Les temps de calculs sont évalués en moyenne à 127,4 s.

Certaines heuristiques ont considéré des objectifs tenant compte de plusieurs critères. C'est le cas de Boufaied *et al.* [85], qui ont testé un algorithme génétique pour la création de blocs mensuels. Ils ont utilisé des données fournies par Tunisair et ont obtenu des solutions dont

la qualité est proche de la borne supérieure sur l'objectif utilisée comme point de comparaison. De même, Sargut, Altuntas et Tulazoglu [86] présentent une heuristique de recherche taboue adaptée aux problèmes de la création d'horaires et de l'assignation de véhicules pour les autobus urbains. Ils testent leur approche sur 10 objectifs multicritères différents et démontrent l'efficacité de leur méthode. Zhou *et al.* [87] ont aussi traité le problème des blocs mensuels personnalisés. L'objectif est constitué de deux critères, soit l'équité de la charge de travail des pilotes ainsi que leur satisfaction. Ils présentent une heuristique utilisant deux colonies de fourmis, chacune étant concentrée sur un critère de l'objectif, et montrent que leur méthode dépasse différents algorithmes myopes ainsi que deux autres métaheuristiques. Zhou *et al.* [88] abordent le même problème à l'aide d'une métaheuristique d'optimisation par essaims particulaires.

Doi, Nishi et Voß [89] ont considéré le problème des blocs mensuels personnalisés à l'aide d'une méthode d'optimisation partielle. Le critère de leur modèle d'optimisation est l'équité de la charge de travail. Ils utilisent en outre la métaheuristique POPMUSIC, qui permet d'obtenir de meilleures solutions au cours de la résolution en se concentrant sur certaines parties spécifiques de la solution actuellement disponible.

Plusieurs autres exemples apparaissent dans la littérature et les métaheuristiques sont variées. On peut mentionner l'évolution différentielle utilisée par Santosa, Sunarto et Rahman [90] pour le CRP, une heuristique de recherche locale variable exploitée par Legrain, Omer et Rosat [91] pour améliorer la résolution du problème des horaires personnalisés pour infirmières sur de grandes instances, ainsi que différentes contributions où les auteurs créent leur propre métaheuristique. C'est le cas par exemple de Chan *et al.* [92] et de Er-Rbib *et al.* [93], ces derniers ayant présenté deux métaheuristiques originales pour le problème de l'affectation des chauffeurs d'autobus avec préférences.

Enfin, on peut noter qu'il y a parfois eu chevauchement entre l'apprentissage machine et les métaheuristiques dans un contexte d'optimisation combinatoire. Un exemple est celui de l'utilisation du ML dans le but d'améliorer la bonne marche des différentes étapes de la métaheuristique. Les contributions de Lu, Zhang et Yang [94] et Wu *et al.* [95] s'inscrivent dans cet ordre d'idée. Les problèmes traités sont le TSP ainsi que le VRP avec capacités. Ce type de travail peut être vu sous l'angle des métaheuristiques ou encore sous celui de l'apprentissage machine en boucle, la métaheuristique interagissant avec l'algorithme d'apprentissage tout au long de la résolution.

Stratégies d'évolution (ES)

Certaines métaheuristiques que nous traitons plus en profondeur dans ce travail appartiennent à la famille des stratégies d'évolution. Nous décrivons leur nature de façon plus détaillée au

chapitre 4. De manière générale, une stratégie d'évolution modifie itérativement un d-tuplet dans le cas d'un problème d'optimisation dans $\mathbb{R}^d$. Chaque itération, plusieurs mutations sont apportées aléatoirement à ce d-tuplet de sorte qu'il y en ait différentes versions légèrement modifiées. Celles-ci sont évaluées par la fonction à optimiser. Les meilleurs candidats sont alors utilisés pour effectuer la mise à jour du d-tuplet initial. La distribution des mutations est parfois mise à jour également, selon la méthode.

Une des percées dans le domaine des stratégies d'évolution est le travail de Hansen et Ostermeier [96]. Ils y présentent l'algorithme de stratégie d'évolution avec adaptation de la matrice de covariance (*covariance matrix adaptation evolution strategy*, ou CMA-ES). Comme pour la plupart des algorithmes ES, les mutations suivent une distribution gaussienne. Ce qui distingue cette approche des précédentes est que la matrice de covariance permettant de définir cette distribution est modifiée à chaque itération en tenant compte des mutations qui ont le mieux fonctionné par le passé. L'algorithme CMA-ES a ensuite donné lieu à de nombreux algorithmes dérivés. Ceux-ci, ainsi que plusieurs aspects essentiels à l'application des stratégies d'évolution, figurent dans Bäck, Foussette et Krause [97].

Les stratégies d'évolution et algorithmes évolutifs plus généralement ont été étudiées afin de déterminer leur potentiel dans le cadre de différentes tâches. Par exemple, Moriarty, Schultz et Grefenstette [98] ont étudié les implications d'utiliser un algorithme évolutif dans le cadre de différents problèmes RL, comme certaines tâches de robotique. Yi *et al.* [99] ont testé une nouvelle approche, les stratégies d'évolution naturelles (*natural evolution strategies* ou NES), et ont observé que les résultats obtenus sur différents problèmes classiques de RL étaient comparables à l'état de l'art pour les métaheuristiques analysées. Salimans *et al.* [100] ont pour leur part démontré que les stratégies d'évolution étaient compétitives avec les méthodes standard de descente de gradient pour l'ensemble des problèmes d'apprentissage par renforcement considérés, et même qu'elles pouvaient parfois les dépasser.

Le travail de Salimans *et al.* [100] peut être considéré sous l'angle de la neuroévolution, où un algorithme évolutif est utilisé pour entrainer les poids d'un réseau de neurones artificiel. Il s'agit d'une certaine façon de l'approche contraire à celle utilisée par Lu, Zhang et Yang [94] ou Wu *et al.* [95] ci-dessus et où l'apprentissage machine permet d'améliorer un algorithme évolutif. Stanley, D'Ambrosio et Gauci [101] ont porté ce concept plus loin en développant l'algorithme HyperNEAT. En effet, leur algorithme permet de faire évoluer la topologie des réseaux de neurones en plus des poids sur leurs connexions. L'idée principale est de repérer des symétries ou tendances dans la structure des réseaux afin d'en réduire la dimensionnalité.

CHAPITRE 3 ORGANISATION DE LA THÈSE

Les différents travaux cités dans la revue de littérature ont mis en évidence à la fois la complexité du problème des blocs mensuels personnalisés et certaines tendances qui se dégagent des interactions de type ML-RO. Ce premier constat est attesté par la multiplicité des techniques qui ont été développées pour améliorer la résolution de la famille des problèmes dont le CRP fait partie, qu'il s'agisse de l'approche DCA, du fenêtrage, de modèles ML ou encore de métaheuristiques. Quant aux interactions ML, l'apprentissage de bout en bout donne de bons résultats pour de plus petites instances de divers problèmes d'optimisation combinatoire ou lorsque le problème a une structure spéciale, mais rencontre plus d'obstacles pour généraliser ces résultats à des problèmes de plus grande taille. Il s'agit d'une avenue de recherche active. Par contraste, l'apprentissage en boucle a connu plusieurs succès marquants concentrés dans le développement de techniques de branchement plus intelligent.

En contraste, l'apprentissage du prétraitement, dont le but est d'apprendre à mieux configurer un solveur avant son lancement, a rarement été étudié. Un contexte où cette forme d'apprentissage est appliquée est la résolution de MIP ou MIQP. Cependant, des exemples d'application plus récents comprennent l'apprentissage du prétraitement pour réduire la taille des sous-problèmes utilisés lors de la génération de colonnes. Or, ce type d'algorithmes ML peut prendre différentes formes selon le prétraitement requis et le type d'informations fournies au solveur traditionnel. De façon générale, l'apprentissage de bout en bout requiert de prédire avec précision certaines parties d'une solution optimale, alors que l'apprentissage en boucle cherche plutôt à apprendre de l'information plus générale mais utile à la bonne marche du solveur. L'approche du prétraitement peut nécessiter l'un ou l'autre de ces éléments, selon l'application. Elle se prête donc particulièrement bien à un examen plus structuré de la nature de l'information apprise par ML qui aide à la résolution d'un problème comme le CRP.

La contribution principale de cette thèse se décline donc en deux parties essentielles. La première est de développer une méthode pouvant améliorer la planification et la résolution du CRP à l'aide d'un algorithme ML. La seconde est de tester différentes techniques d'apprentissage du prétraitement pour synthétiser ce qui est adapté au CRP et aux problèmes qui y sont apparentés. Chacune des contributions spécifiques s'aligne avec cette analyse plus approfondie.

Au chapitre 4, nous nous concentrons sur la planification du CRP. Il est essentiel, avant de lancer la résolution du problème des blocs personnalisés, de déterminer certains paramètres propres à la tâche. Ceci inclut le nombre de pilotes affectés à l'horizon mensuel, et par le

fait même la productivité moyenne en heures de vol qui en est attendue au cours du mois, ainsi que le nombre de pilotes en réserve. La qualité de cette paramétrisation est directement liée à la valeur optimale (et donc à la satisfaction totale) qui peut être accordée aux pilotes, et il arrive que ces paramètres doivent être décidés en temps réel. Pour cette raison, nous fournissons une procédure d'affectation successive dont les coûts sur les arcs sont calculés par un réseau de neurones dont les poids sont appris par ML. La valeur de l'objectif de la solution obtenue est fortement corrélée avec la valeur du CRP une fois résolu, même lorsque que cette solution diffère notablement de celle qui est optimale. La solution par affectation successive sert donc de prétraitement pour s'assurer qu'on ait un problème bien calibré et dont l'optimalité est satisfaisante pour les pilotes. Les détails de l'approche utilisée sont présentés plus amplement au chapitre 4.

Au chapitre 5, nous élargissons la portée de la procédure d'affectation successive pour accélérer la résolution du CRP. Nous mettons en place une méthode par fenêtrage, ce qui est nouveau en soi pour le problème des blocs mensuels personnalisés. Nous montrons ensuite que démarrer l'algorithme de fenêtrage avec une solution initiale obtenue par affectation successive donne de meilleures solutions finales. Bien que dans les deux cas, le temps de résolution est moins de 10 % de celui observé sans fenêtrage, l'approche ML diminue de 33,3 % la détérioration de la valeur de l'objectif, de sorte que pour les deux instances de grande taille testées, les solutions sont à moins de 1 % de l'optimalité en moyenne. La solution ML fournit dans ce cas l'information requise par le solveur traditionnel pour que l'optimisation de chaque fenêtre n'entre pas en conflit avec la bonne information fixée à l'intérieur des autres.

Le chapitre 6 vise à déterminer le potentiel de l'affectation successive à accélérer le CRP à l'aide de l'agrégation dynamique de contraintes. Cette catégorie d'algorithmes est souvent utile lorsqu'une caractéristique du problème assure que plusieurs suites de tâches présentes dans la solution optimale peuvent être déterminées avec un haut degré de confiance. En l'absence de telles conditions, comme pour le problème des blocs mensuels personnalisés, cette prédiction devient plus difficile, et construire une solution initiale satisfaisante (telle que requise en entrée par DCA) représente un défi. L'affectation successive est testée dans ce contexte et est associée à cinq méthodes différentes visant à en améliorer le fonctionnement. Le constat qui ressort de ces expériences est que même une solution initiale avec une valeur de la fonction objectif proche de la solution optimale ou quasi-optimale fournie par le solveur est fortement incompatible avec cette dernière.

Enfin, le chapitre de discussion qui suit l'ensemble de ces contributions spécifiques récapitule ce qui a été observé, la significativité des constats qui en émergent, et certaines pistes de recherche pour le futur.

CHAPITRE 4 ARTICLE 1 : GAINING INSIGHT INTO CREW ROSTERING INSTANCES THROUGH ML-BASED SEQUENTIAL ASSIGNMENT

Authors : Philippe Racette, Frédéric Quesnel, Andrea Lodi, and François Soumis

Submitted on March 2, 2023 and published on June 25, 2024 in TOP, Volume 32 Issue 3, pages : 537-578, DOI : 10.1007/s11750-024-00678-8

Abstract

Crew scheduling is typically performed in two stages. First, solving the crew pairing problem generates sequences of flights called pairings. Then, the pairings are assigned to crew members to provide each person with a full schedule. A common way to do this is to solve an optimization problem called the crew rostering problem (CRP). However, before solving the CRP, the problem instance must be parameterized appropriately while taking different factors such as preassigned days off, crew training, sick leave, reserve duty or unusual events into account. In this paper, we present a new method for the parameterization of CRP instances for pilots by scheduling planners. A machine learning-based sequential assignment procedure (*seqAsg*) whose arc weights are computed using a policy over state-action pairs for pilots is implemented to generate very fast solutions. We establish a relationship between the quality of the solutions generated by *seqAsg* and that of solutions produced by a state-of-the-art solver. Based on those results, we formulate recommendations for instance parameterization. Given that the *seqAsg* procedure takes only a few seconds to run, this allows scheduling workers to reparameterize crew rostering instances many times over the course of the planning process as needed.

4.1 Introduction

Crew scheduling is a complex problem crucial to the operations of airline companies. From an economic point of view, it is the second source of expense for airline carriers, after fuel costs [102]. As such, it has typically garnered interest from researchers seeking to improve its

efficiency.

Crew scheduling is typically solved in two steps. First, it is necessary to solve the *crew pairing problem* (CPP). This problem involves creating pairings, defined as sequences of flights (also called airlegs) and layovers starting and ending at the same airport. They represent a few days of work for a crew member. The next step is to assign pairings to pilots (the case we consider) or the cabin crew. One possibility is to solve the *crew rostering problem* (CRP). The outcome is a full schedule per pilot over the month considered, and the entire collection of schedules makes up the roster for the month.

When solving the CRP, the scheduling worker must plan the CRP instance. This process involves a very large number of decisions of two types. First, there are decisions made sequentially and related to granting pilot requests. Requests made by pilots can be of many kinds and include asking for certain days off, further training (e.g., training to fly airplanes of different types), medical leave or work-related health screenings. Next, it is necessary to determine, of all available pilots at the company, how many will be on reserve for the upcoming month. Reserve duty refers to the situation where a pilot is on call for the month to fill in when pairings cannot be assigned. This is judged an undesirable situation for industry-specific reasons, such as the salary given for few hours worked. The number of pilots on reserve duty may change during planning as new requests are made by pilots.

Making these decisions occurs as the scheduling worker also aims to take pilot preferences regarding time off and pairing assignments into account. Note that those preferences do not need to be satisfied. They also consider special events on certain months such as Thanksgiving or Christmas holidays. This requires determining a set of parameters indicating what percentage of the requests should be granted as preassigned days off (i.e., specific days off explicitly requested by pilots in advance) and how many pilots should be on reserve.

Adequately parameterizing the instance for the upcoming month means finding good values for these parameters such that a maximum of requests is granted and a minimum of pilots is required to carry out flights on regular duty (i.e., not on reserve) while not affecting the latter's satisfaction or the feasibility of the schedules generated. Granting too many requests or putting too many crew members on reserve forces pilots actively covering pairings during the month to serve unwanted pairings. It may also mean that some pairings will remain unassigned during peak flight periods for the airline. This situation must be avoided as a priority while still taking pilot satisfaction into account. Overall, this state of matters means that planning a CRP instance is not a trivial task. Indeed, the parameterization process can involve hundreds of manual decisions when determining how to create a definitive instance. In practice, it may also have to be repeated many times until an instance is deemed satisfactory.

The impact of an incorrectly parameterized instance becomes obvious when many pilots become unavailable for one reason or another. Typically, such situations would include having many pilots gone for training or on personal leave, or an unusually high number of pilots having been granted time off during the same period, such as yearly recurring holidays. In those problematic cases, the CRP can be very difficult to solve, or even infeasible.

To prevent this situation, one option is to adjust the parameters each time a decision is made and to solve the CRP instance to see if pilot satisfaction or feasibility are affected. However, this method quickly becomes extremely time-consuming, as the mathematical programming solver used must generate a full roster each time the necessity of a change to the instance becomes apparent. For that reason, insight into the CRP instance allowing to change parameters without solving the entire CRP could be very useful. Unfortunately, obtaining this kind of reliable, instance-specific insight can be hard, especially for inexperienced planners.

In recent years, machine learning (ML), including reinforcement learning (RL), has proven to be useful in extracting valuable information from existing data, including data pertaining to highly constrained problems like the CRP. In this paper, we propose integrating ML tools with optimization techniques to analyze CRP instances and improve their parameterization.

This paper proposes a new way of aiding scheduling planners to parameterize CRP instances. In particular, we do the following :

— propose an ML-based sequential assignment procedure called *seqAsg* that quickly builds a feasible roster. The *seqAsg* procedure relies on pairing assignment weights computed via an RL model ;

— establish the relationship between the quality for both the solution generated by *seqAsg* and the solution obtained by a state-of-the-art solver ;

— make recommendations for scheduling planners based on the quality of the solutions generated by *seqAsg*.

Together, these different contributions create a comprehensive method for checking iteratively, each time the scheduling worker makes a decision, if some changes need to be made to the instance. The *seqAsg* procedure takes only a few seconds to run even on large instances, making it suitable for instance analysis.

We now present the structure of the paper. A literature review on the CRP and the integration of ML and operations research (OR) is presented in Section 4.2. In Section 4.3, we give a formal definition of crew scheduling in general and of the CRP especially. In Section 4.4, we present the data used in the computational experiments. In Section 4.5, we describe the *seqAsg* algorithm, as well as the function approximators used to generate its weights. In Section 4.6, we measure the quality of the solutions generated by our algorithm, demonstrate

the relationship between them and those obtained by the solver, and develop guidelines for scheduling workers. We conclude with a general discussion in Section 4.7.

4.2 Literature review

In this section, we review certain techniques that have been used to solve the CRP and crew scheduling problems more generally. We also cover the literature on several aspects of ML, particularly where ML methods have been relevant to solving combinatorial optimization (CO) problems.

4.2.1 Crew scheduling and column generation

The crew scheduling literature includes a large variety of different methods aimed at solving crew scheduling problems successfully. For a general overview of the main approaches involved, see Gopalakrishnan and Johnson [13].

One well-studied problem in the crew scheduling literature is the crew pairing problem described in Desaulniers *et al.* [11]. This problem concerns building feasible pairings of high quality while covering all flights planned for the month. Once pairings are built, it is possible to form high-quality schedules using crew rostering. There is also the possibility of solving the crew bidding problem described in Boubaker, Desaulniers and Elhallaoui [15]. In that context, schedules are built from the provided pairing list, and pilots and flight attendants bid on them based on seniority.

In many crew scheduling problems, there are too many feasible pairings or schedules and it is impossible to list them all explicitly. For that reason, column generation is often used. Column generation involves iteratively solving a master problem and several subproblems (e.g., one per pilot), with columns here referring to feasible schedules. The available columns at an iteration form the *restricted* master problem (RMP) that is solved to find the optimal solution for this restricted set of columns. This provides a dual solution whose variables are used to define the reduced costs provided to the subproblems. The schedules found by solving the subproblems can then be added to the RMP. When there are none, the current solution is optimal. See Lübbecke and Desrosiers [29] for more on column generation.

Additional methods have been developed to tackle these scheduling problems as well. Saddoune *et al.* [26] suggest an approach that solves both the crew pairing and crew rostering problems simultaneously with column generation. Similarly, Zeighami and Soumis [27] bring forward an integrated model using Benders' Decomposition. In Zeighami, Saddoune and Soumis [28], the integrated model focuses on Lagrangian relaxations. This family of approaches

can be advantageous to solving pilot and copilot pairing and rostering problems with the objective of creating similar pairings even if rosters are different. These integrated modelling approaches can better accommodate personnel preferences.

The CRP has been the focal point of a growing body of literature as well. There are many variants of the CRP, and each favors different objective criteria. Some common criteria are the solution cost, crew satisfaction, fairness, seniority, or a combination of these points of consideration. While we present in this section some examples of CRP formulations applied to research problems, a more comprehensive review of the different CRP approaches can be found in Kohl and Karisch (2004) [1].

With the work of Caprara *et al.* [14], we can see that the authors use Lagrangian relaxations to solve the CRP in the context of the railway industry. They use the relaxations to assign all the fixed set of required pairings with the smallest number of crew members possible. Gamache *et al.* [19] present a column generation approach and show that they can solve the CRP substantially faster (up to three orders of magnitude) while remaining within 0.6% of the optimal solution objective. Cappanera and Gallo [103] feature a multicommodity formulation of the CRP for airline crew members and show the effectiveness of their method with instances of various sizes and natures using CPLEX. Another paradigm is introduced in Moudani *et al.* [21], where the authors consider a bi-criterion objective including both the cost and crew satisfaction as important variables to optimize. Using a genetic algorithm, they provide a set of solutions of non-inferior quality that can help with the full resolution of the CRP.

We also mention the aspect of the CRP we cover in more depth in this paper, i.e., the parameterization process that occurs when scheduling planners create CRP instances. This process has little published literature addressing it, as the sort of work that it entails is often highly customized depending on the airline and even individual planners. This being said, a notable exception is Beulen, Scherp and Santos [62]. In their work, the authors train a neural network to determine when crew members' flight requests should be granted. These decisions occur during CRP planning. They obtain that their method grants 22% more requests than had previously been the case for a major European airline. Nonetheless, the existing work on instance parameterization is very scarce and we draw attention to it with this paper.

4.2.2 Machine learning in operations research

When it comes to the different ways of aiding the resolution of CO problems with ML, there are three different paradigms. The first one is called *end-to-end* learning and involves the entire resolution of the problem through an ML solution without the help of a CO solver.

For example, Vinyals, Fortunato and Jaitly [104] introduce the concept of pointer networks, which are inspired by the attention mechanism for natural language processing described in Bahdanau, Cho and Bengio [105]. They use pointer networks to solve various CO problems. Bello *et al.* [50] use a policy gradient RL algorithm to solve the traveling salesman problem (TSP), while Nazari *et al.* [51] expand on this contribution to apply a deep RL algorithm to the vehicle routing problem (VRP). Attention mechanisms are also used in Kool, van Hoof and Welling [52], where the authors develop a model with attention layers to improve the existing RL heuristics applied to the TSP and the VRP. For a more comprehensive review of the use of end-to-end learning on OR problems, including a large variety of different tasks, we refer the reader to Kotary *et al.* [106] and Karimi-Mamaghan *et al.* [48].

The next type of ML-OR integration is *sequential* learning. In this case, a trained ML model produces information that is then passed to the solver before resolution. For example, in Bonami, Lodi and Zarpellon [61], the ML algorithm determines whether a mixed integer linear program should be linearized in CPLEX v. 12.10.0 before solving starts. Using a support vector machine model, they reduce computing times by a percentage ranging from 28% to 92% depending on the problem's size. In crew scheduling, sequential algorithms have primarily focused on giving elements of initial solutions to the solver. Yaakoubi, Soumis and Lacoste-Julien [32] use supervised learning to provide initial clusters of airlegs likely to go together in a pairing. They improve the CPP solution time by a factor of 10 with a 0.2% improvement of the objective. Yaakoubi, Soumis and Lacoste-Julien [33] use a similar approach with an integral primal simplex algorithm presented in Elhallaoui *et al.* [8]. As to crew rostering, Quesnel *et al.* [63] use supervised learning to learn which pairings are likely to be in a flight attendant's final schedule and to reduce the size of the subproblem network. They obtain an objective of similar quality for the CRP in less than half the time.

The last type of ML application to OR algorithms is *integrated* learning, also known as online learning. In this framework, the ML algorithm intervenes throughout the OR algorithm. Among other examples, Furian *et al.* [70] suggest a branch-and-price algorithm that makes use of online learning to determine binary decision variables for node selection and make predictions on fractional variables during branching. Their method makes significant gains in computing time for the VRP. In Morabit, Desaulniers and Lodi [64] and Morabit, Desaulniers and Lodi [71], the authors use ML to improve column generation by selecting promising columns or arcs among those generated by or present in the subproblems, respectively, at each iteration of the resolution of a subproblem. They save 40% and 30% of the computing time for the CPP, respectively. A similar approach is developed in Tahir *et al.* [107], where only high-probability flight connections are included in the subproblems to be solved. They outperform state-of-the-art benchmarks for the CPP both in terms of objective and speed.

With regard to the CRP, some work has been done from an end-to-end point of view in work trying to address the problem with ML, while it has more often been treated from a sequential or integrated learning paradigm as in some of the contributions cited above. Nonetheless, some research, mostly using different kinds of heuristics, is worth mentioning. The work in Armas *et al.* [78] builds a multi-start heuristic alternating random and local searches to find fast high-quality solutions on small, industry-realistic CRP instances. For this purpose, they draw inspiration from Juan *et al.* [79]. Zhou *et al.* [87] seek to optimize a CRP objective taking into account both fairness and satisfaction, and they present an ant colony algorithm that is then compared against other heuristics on the instances found in Kasirzadeh, Saddoune and Soumis [20]. Zhou *et al.* [88] expand on this work and test their method on a small instance comprised of 6 pilots. These different contributions also build on the work, among others, found in Lučic and Teodorović [76], Lučic and Teodorović [77], Boufaied *et al.* [85] and Maenhout and Vanhoucke [84]. These contributions all consider different heuristics to solve different variants of the CRP more efficiently.

However, in light of the previous work mentioned and to the best of our knowledge, ML techniques have not yet been used to compare the solutions of such methods with state-of-the-art CRP solvers on large, realistic instances from an industry angle, and while also providing an ML solution in a few seconds only on such instances. Typically, we observe that either the instances considered are small, that they are not realistic in a real-life context, or, if the instances are both large and realistic, that they take several minutes to solve, even when the technique cuts the solving time in half or more. This paper can then be seen as presenting a way to obtain very fast solutions on large, difficult, and industry-realistic instances through ML so as to infer useful information regarding the CRP's planning phase.

We note that the work presented in this paper can be viewed as belonging to both the end-to-end and integrated learning paradigms. ML solutions are generated without the use of a traditional CRP solver : however, the ML weights learned by the algorithm we present in Section 4.5 intervene at each step of the *seqAsg* procedure.

4.3 Crew rostering problem

We now describe the variant of the crew rostering problem considered in this paper. It is inspired by the work presented in Kasirzadeh, Saddoune and Soumis [20] with some adaptations to fit the current version of the CRP coded in the solver we use in Section 4.6. Consider the set of pairings W and the set of pilots K. The goal is to assign the pairings in W to pilots in K to build one feasible schedule per pilot. The schedules are built over a monthly horizon. Also, each pilot in K is attached to one of d bases, and the problem is solved separately by

base.

Schedules must comply with collective agreements and airline regulations. In this paper, we consider the following schedule restrictions :

— each pairing must be assigned to exactly one pilot,
— each pilot must receive a full schedule (complete sequence of pairings and days off),
— each pilot must be assigned a minimum of T^{off} days off, which may include preassigned days off,
— pilots cannot work over $T^{credits}$ hours in a month,
— pilots cannot work more than T^{work} consecutive *duties* (calendar days with at least one ongoing pairing),
— there must be at least T^{min} hours between two consecutive pairings, and
— all preassigned days off must be respected.

The objective function for the CRP, given below by (4.1), maximizes an expression with three terms. The first term maximizes pilot satisfaction. The satisfaction a pilot derives from their schedule is a function of the pilot's preferences. Prior to solving the CRP, pilots state their preferences for certain flights (that can be part of any pairing) and certain vacations (three consecutive days off). Furthermore, they assign a weight to each preference. A schedule's satisfaction score for a given pilot is the weighted sum of the pilot's preferences that the schedule meets. The other two terms subtract penalties for failing to cover some pairings, or for not respecting preassigned days off. The penalty for an unassigned pairing (i.e., a pairing that has not been assigned) increases by C^{flight} points for each airleg included in the pairing (e.g., the penalty for an unassigned pairing with six airlegs is $6 * C^{flight}$ points). Uncovered preassigned days off are heavily penalized, at C^{day} points per day. This value is selected to be sufficient to ensure that all such preassigned days off are covered when solving the CRP.

We note that, due to the penalties described above, while the requirements related to pairing assignments (which are also referred to as coverage requirements) and preassigned days off are part of the CRP, the violation of these constraints does not make a schedule (or the corresponding roster) infeasible and is instead penalized. However, any other violated restriction makes a schedule infeasible.

4.3.1 Mathematical formulation

The CRP is defined as follows. Let Ω and Ω^k be the set of feasible schedules, and feasible schedules for pilot $k \in K$, respectively. Let x_p^k be a variable equal to 1 if schedule $p \in \Omega^k$ is assigned to pilot k, and 0 otherwise. Let Q^k be the set of preassigned days off for pilot $k \in K$. Let y_w and y_q be the slack variables ensuring the coverage of pairing $w \in W$ and preassigned

day off $q \in Q^k, k \in K$, respectively. Let c_p^k be the cost (satisfaction score) of schedule $p \in \Omega^k, k \in K$. Let $n_w, w \in W$ be the number of airlegs in unassigned pairing $w \in W$. Finally, let a_{wp} and a_{qp} be binary constants indicating whether pairing w or preassigned day off q are in schedule $p \in \Omega^k, k \in K$, respectively. The CRP reads as

$$\max \sum_{k \in K} \sum_{p \in \Omega^k} c_p^k x_p^k - C^{flight} \sum_{w \in W} n_w y_w - C^{day} \sum_{k \in K} \sum_{q \in Q^k} y_q \tag{4.1}$$

subject to :

$$\sum_{k \in K} \sum_{p \in \Omega^k} a_{wp} x_p^k + y_w = 1, \quad \forall w \in W \tag{4.2}$$

$$\sum_{p \in \Omega^k} a_{qp} x_p^k + y_q = 1, \quad \forall q \in Q^k, \forall k \in K \tag{4.3}$$

$$\sum_{p \in \Omega^k} x_p^k = 1, \quad \forall k \in K \tag{4.4}$$

$$x_p^k \in \{0, 1\}, \quad \forall k \in K, \forall p \in \Omega^k \tag{4.5}$$

Equations (4.2) and (4.3) ensure that each pairing and preassigned day off is covered exactly once, respectively. Equations (4.4) ensure that each pilot has exactly one schedule.

Formulation (4.1) – (4.5) typically contains a large number of variables. Therefore, it is often solved using a heuristic branch-and-price algorithm. In that framework, relaxations of (4.1) – (4.5) are solved using column generation. The column generation algorithm is embedded in a heuristic branch-and-bound algorithm. Such a commonly used heuristic fixes some pairing variables at each node until an integer solution is obtained.

4.4 Data

Our data is derived from the data sets published in Kasirzadeh, Saddoune et Soumis [20]. It consists of seven instances of different sizes for which the airlegs (i.e., flights), pilot bases, and airports are given. The instances are labeled I1 to I7 in increasing number of flights, and each instance has $d = 3$ bases (see Table 4.1). For each instance, a state-of-the-art CPP solver builds pairings (see Quesnel, Desaulniers and Soumis [108]). This data set was originally chosen due to the diversity of the seven instances. It includes instances with different proportions of short-haul and medium-haul pairings lasting from 1 to 5 days. These proportions are such that the difference in composition between instances is very large. For example, for instance I2, the percentage of pairings of length ≥ 4 days is 1.8% for the base with the highest percentage. For instance I1, this goes up to 66.7%. Given this fact, each instance being quite

different from the others, an ML algorithm trained on one instance and generalizing well to the other six shows robustness on a variety of instance types.

As the data set does not contain pilot-related information, pilot-related data was randomly generated using a procedure adapted from Quesnel, Desaulniers and Soumis [109] that aims to create preferences similar to that of an international airline based in North America. First, a percentage of pilots (uniformly chosen between 5% and 15%) receive randomly four, six, or seven consecutive preassigned days off. Then, each pilot is given three or four three-day preferred vacations. These vacations are generated uniformly over the month with the requirement that they be non-overlapping with preassigned days off or with each other. As to flight preferences, pilots in real-life scenarios tend to choose airlegs that mark the beginning or the end of a pairing, as mentioned in Quesnel, Desaulniers and Soumis [109]. Each pilot is given between zero and ten such flights, inclusively.

Tableau 4.1 Description of the instances used

Instance	Nb pilots	Nb airports	Nb airlegs	Nb pairings
I1	27	26	1013	157
I2	27	35	1500	331
I3	37	41	1854	359
I4	117	49	5613	1038
I5	239	34	5743	1222
I6	180	52	5886	1080
I7	322	54	7765	1473

The weights in the objective that are associated with pilot preferences are as follows. For each preferred airleg or vacation, a weight ranging between 10 and 400 points is added to the objective. Preference weights are multiples of 25 points, with the exception of some flights worth 10 points. Preferred vacations must also be worth at least 75 points. All preference weights are generated randomly following two uniform distributions : one for flights and the other for vacations. All possible values for a flight or preferred vacation have equal probabilities. As mentioned in Section 4.3.1, the total satisfaction score of a pairing-pilot assignment is the weighted sum of all preferences for flights included in the pairing.

Preference weights are generally below 150 points for specific airlegs, whereas they often go to the upper end of the weight range for three-day vacations. Given that pairings are made on average of over six airlegs (and almost always have at least two), this means the penalty for missing a pairing is almost always considerably higher than not assigning a preferred pairing to a pilot. The cost of not assigning a pairing to anyone is also usually higher than the loss

of satisfaction from not assigning a three-day vacation. The opposite case occurs only rarely, i.e., when a very short pairing is not assigned to allow for the allocation of a highly prized vacation. On these few occurrences, this is considered an acceptable compromise so as to avoid taking away what may be a pilot's greatest source of satisfaction for the month.

As we aim to learn from data sets with an ML-based procedure, we need more than seven instances to train the ML model. As such, we build a new method for obtaining more data. For each instance, we create different scenarios with each representing a different set of pilots and associated preferences. We generate 30 different scenarios for each instance. The scenarios are obtained with the help of software that takes a fixed set of pairings, a fixed number of pilots, possible preferred flights, and preassigned days off as input (i.e., those of the original instance). A file containing the parameters for the different uniform probability distributions is also provided, as shown in Table 4.2. Omitted from this table is information related to the distributions for preference weights, as their software implementation is less straightforward. Drawing from the possible flights, preferred vacations, and sequences of preassigned days off, as well as the parameter file, the software then creates a random scenario by assigning certain flights and vacations as preferred to each pilot and giving a percentage of them preassigned days off. In practice, this means that scenarios are close to being permutations of each other. However, this is not quite the case, as not all preferences appearing in one scenario do in the other and vice versa due to sampling.

Tableau 4.2 Parameters for scenario generation

Parameter	Possible values
Fraction of pilots with preassigned days off	$\{0.05, 0.10, 0.15\}$
Number of consecutive preassigned days off	$\{4, 6, 7\}$
Number of preferred vacations	$\{3, 4\}$
Number of days per preferred vacation	3
Maximum number of legs per employee	10

Once the scenarios are generated, we can build an input file for each one. Each input file is passed to the GENCOL software (version 4.5), where we implement a branch-and-price algorithm for the CRP. GENCOL is a state-of-the-art solver similar to the ones used in commercial planning software. The solutions produced by GENCOL are used as a point of reference in Sections 4.6.2 and 4.6.3. The CRP constants used in our experiments, as defined in Section 4.3, are given in Table 4.3.

Tableau 4.3 CRP constants

Constant	Value	Description
T^{off}	10	Min. number of days off
T^{credit}	85	Max. credited hours per pilot
T^{work}	6	Max. number of consecutive duties
T^{min}	12	Min. postpairing rest time (hours).
C^{flight}	100	Penalty for unassigned flight
C^{day}	1,000,000	Penalty for preassigned days off

The penalty for preassigned days off is high to the extent shown as preassigned days are enforced by collective agreement regulations. A high penalty ensures that these coverage constraints are met.

In a real-life application, data could be obtained by looking at historical rosters over several months or years for different instances. In such a situation, the rosters would be more closely related than the scenarios we generate since the majority of the pilots remain from one month to another. For this reason, our prediction task is likely harder than it would be in real life. Additionally, in cases where no historical data exists and where ML weights learned at a previous time as in Section 4.5 are not available, it may be possible to use the scenario generation procedure outlined above on a single instance. We explore this possibility further in Section 4.6.2.

4.5 Algorithm

Our method gains insight into a CRP instance by quickly generating a CRP solution and then analyzing it. The solution generation procedure has two steps. First, a sequential assignment procedure (called *seqAsg*) quickly produces a schedule for each pilot. This solution may be infeasible. In that case, a heuristic correction procedure is applied. We describe the sequential assignment procedure in Section 4.5.1 and the feasibility heuristic in Section 4.5.2. The algorithm *seqAsg* relies on a set of weights that are produced by an ML model trained on the data described in Section 4.4. We describe the ML procedure used for this purpose in Section 4.5.3.

4.5.1 Sequential assignment

The sequential assignment procedure aims at providing a schedule for each pilot quickly (i.e., within a few seconds). Here, an assignment is defined as adding a specific pairing, day off (whether preassigned or not), or preferred vacation, to the schedule of a pilot ready to work on a certain day. An assignment problem seeks to assign all required pairings and preassigned days off on a certain day to pilots and to give all pilots required to receive an assignment on that day exactly one pairing, day off, or preferred vacation. Some pilots available to work, but not required to, may also be assigned an activity. The sequential assignment procedure iteratively solves an assignment problem for each day of the month, in chronological order. Each problem assigns pairings and days off to pilots. The sequential assignment procedure can thus be viewed as a constructive heuristic.

The assignment problem for day t can be described as follows. Let W^t be the set of pairings to be covered. Let P^t be the set of pilots able to resume work on day t and whose last assignment ends on day t. Let Q^t be the set of pilots able to resume work on day t and whose last assignment ends on the day just before t. Let A^t be the set of possible assignments. Days off and preferred vacations are not part of W^t as it is not strictly required to cover them, with the exception of preassigned days off, which we address later. However, they can still be assigned to pilots : these assignments are part of A^t. The distinction between P^t and Q^t is important, as the pilots in the former may receive an assignment on day t but are not obligated to, while the pilots in the latter must receive an assignment. Pilots that are not in either set are excluded from the problem.

To build the set A^t of valid assignments, we consider the following criterion. The assignment of activity (pairing, day off, three-day vacation) j to pilot i, denoted (i, j) is valid if :

- the pairing starts after the pilot has completed their last assignment (with sufficient postpairing rest in the case of a pairing arrival if the previous activity and j are both pairings),
- if j is a pairing, it does not cause the pilot to work more than T^{work} consecutive working days, and
- it does not overlap with a preassigned day off, in the case of pairings or preferred vacations.

Let assignment variable $x_{ij}, (i, j) \in A^t$ be equal to 1 if pairing, day off, or three-day vacation j is added to the partially built schedule for pilot i. Let w_{ij} be the weight of such an assignment, corresponding to a measure of its desirability. Let slack variable y_j be equal to one if pairing $j \in W^t$ remains unassigned.

The assignment problem at day t then reads as

$$\max \sum_{(i,j)\in A^t} w_{ij}x_{ij} \tag{4.6}$$

subject to :

$$\sum_{i:(i,j)\in A^t} x_{ij} + y_j = 1, \quad \forall j \in W^t \tag{4.7}$$

$$\sum_{j:(i,j)\in A^t} x_{ij} \leq 1, \quad \forall i \in P^t \tag{4.8}$$

$$\sum_{j:(i,j)\in A^t} x_{ij} = 1, \quad \forall i \in Q^t \tag{4.9}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in A^t \tag{4.10}$$

Here, constraints (4.7) ensure that each pairing is covered, while constraints (4.8) and (4.9) ensure that pilots available at day t are assigned to exactly one pairing, day off or preferred vacation or remain idle for those pilots whose last arrival was on day t, as expressed by the inequality in constraints (4.8). Note that the slack variables y are not penalized in the objective function.

In our tests, we solve an assignment problem that differs slightly from (4.6) - (4.10). Indeed, we add constraints to ensure that as many pairings as possible are assigned. Let G be a bipartite graph representing the feasible assignments of pilots to pairings, and let there be a maximum matching in G. The additional constraint then reads :

$$\sum_{j\in W^t} y_j = |W^t| - \nu(G) \tag{4.11}$$

where $\nu(G)$ is the size of the maximum matching discussed here.

It is worth observing that the weights w_{ij} are learned through ML, which we justify in Section 4.5.3. These weights should reflect not only the pilot preferences, but also the need to assign each pairing to a pilot, as well as the need for each pilot to have a valid schedule. For example, a bigger weight may be given to a pairing that is harder to assign. The ML model trained to learn these weights is described in detail in Section 4.5.3.

Another way to represent the sequential assignment problem for day t is to set up a bipartite graph for each day. In that graph, arcs link pilots to the pairings or days off that can be

assigned to them, with arc weights determining the desirability of a pilot-pairing assignment. In that case, W^t, P^t and Q^t represent pairing nodes, nodes for pilots available to work after the previous assignment ended on day t and pilots available to work after the previous assignment ended the day before t, respectively, while A^t represents possible assignment arcs. Pilots that are not available on day t are also represented by nodes that are not linked to any arc, and arc capacities are all exactly one. As this formulation is an example of a max (network) flow problem whose demands are all integers, solving the linear relaxation of the problem always provides an integer solution. This considerably simplifies its resolution. Figure 1 represents this conceptualization of the sequential problem solved by the *seqAsg* procedure. To be complete, each graph would need additional flow arcs : input arcs for pilots and output arcs with flow set to 1 exactly for pairing nodes. As input arcs vary by pilot, we do not represent input or output arcs. In Figure 1, we also represent imaginary pilots to account for pairings that are left unassigned. Satisfying the requirement that these pairings be assigned by selecting arcs associated with virtual pilots is equivalent to setting the slack variables for the unassigned pairings to 1. The input flow on nodes for those virtual pilots is equal to the number of pairings that cannot be assigned on the day considered.

Each assignment problem is solved with the goal of maximizing the weighted sum of the selected assignments. Once this is done, the pairings, days off, or preferred vacations associated with the selected assignments are added to the schedules of the corresponding pilots. Then, the procedure moves on to the following day and solves the next assignment problem with the pilots available on that day (i.e., those not currently carrying out a previously assigned pairing or preferred vacation or still recovering during postpairing rest). Those steps are then repeated until the end of the month, at which point each pilot has a complete schedule.

The *seqAsg* procedure does not strictly enforce the limit for the total number of credit (i.e., working) hours per pilot over the entire month, as assignment decisions are done on a day-to-day basis. We choose this approach as enforcing a hard limit on credit hours in the sequential assignment procedure could prevent assigning a highly desirable pairing in a later graph. The drawback of this is that following the *seqAsg* procedure, some schedules may exceed T^{credit} hours of flight time. We employ two strategies to prevent this. First, the weights computed by the ML model consider the pilot's accumulated work time so as to discourage assignments that exceed the credit limit or cause future high-quality assignments to do so. Second, a feasibility heuristic later corrects any remaining violations.

Producing a schedule with the *seqAsg* procedure is much faster than solving the CRP using branch-and-price. The quality of the obtained solution is highly dependent on the performance of the learning algorithm presented in Section 4.5.3.

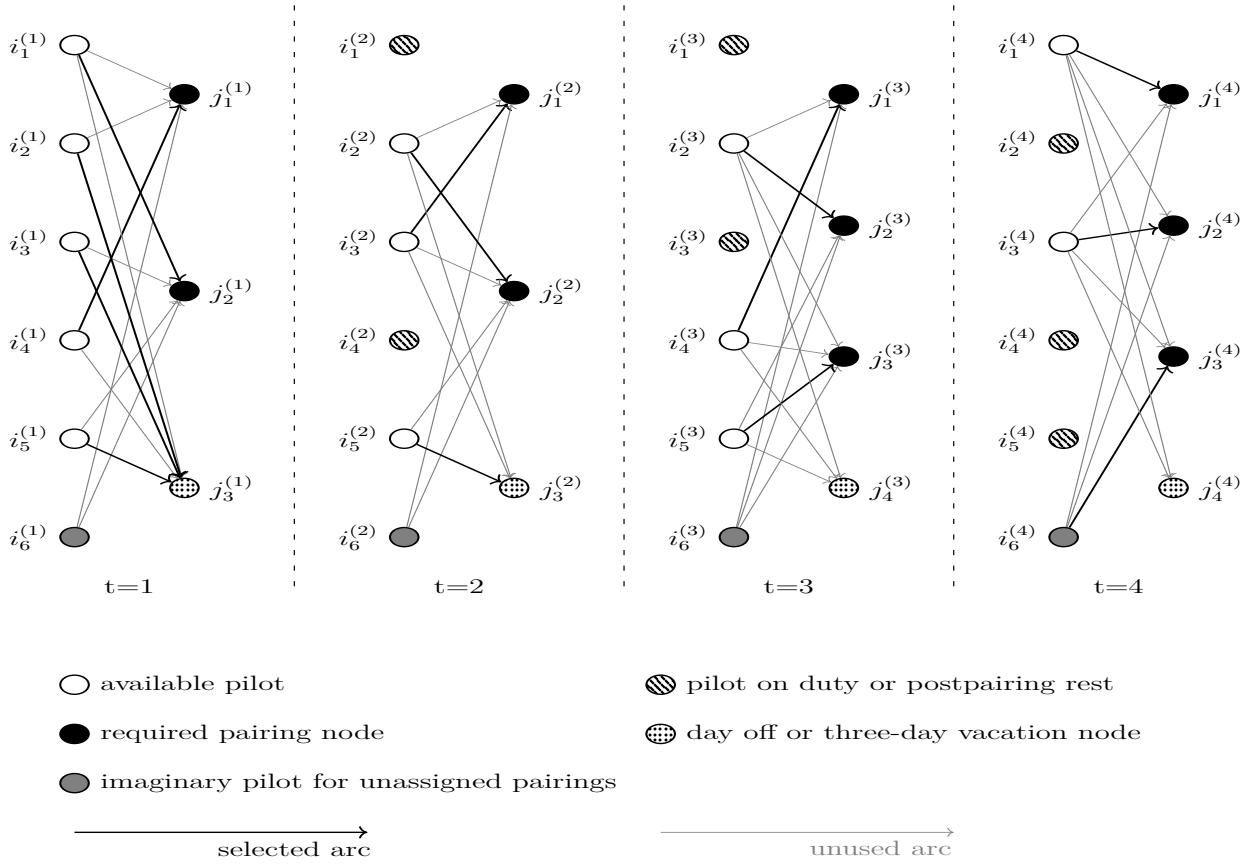


FIGURE 4.1 Visual representation of *seqAsg*. Sequential assignment of pairings and days off for available pilots

4.5.2 Feasibility heuristic

The next component of the methodology presented in this section is the heuristic that ensures each schedule within an ML-generated solution is feasible. While the majority of such schedules will generally be feasible, a minority of them are expected to require some corrections. There are two possible types of infeasibility. First, some pilots may have been assigned too many credit hours. Second, it is possible that not enough days off have been assigned to some pilots.

In order to make each schedule feasible, each source of infeasibility is addressed in the following order. First, the algorithm tackles pilots with too many credits. To correct the schedules for these pilots, pairings are removed until there is no excess credit. Specifically, the algorithm first tries to repair the schedule by removing one or two pairings in a way that minimizes the corresponding penalty for leaving a pairing unassigned (i.e., $n_w * C^{flight}$, as presented in Section 4.3.1). If that fails, it instead removes pairings one by one, in increasing order of

penalty, until there remains no excess credit. Next, if after these operations, some schedules still contain too few days off (i.e., fewer than T^{off}), pairings are removed from those schedules, in increasing order of penalty, until there are enough days off allocated to the pilot. After this step, the schedule for each pilot is feasible, as is the entire roster. An example of pseudo-code for the heuristic can be found in Section 4.8.

Also, at the end of this process, it may be possible to re-assign some unassigned pairings that have been excluded from a pilot's schedule. Indeed, in a minority of cases, such pairings may fit within another schedule and be directly inserted into it. This can happen when both steps of the heuristic have been applied to make a schedule feasible so that short, unassigned pairings can fit. However, this is an uncommon situation, especially for instances that were well planned, where very few pairings are left unassigned. When some short pairings can in fact be reinserted into a schedule, there are too few of them to get a significantly better solution. For these reasons, this step has not been implemented.

4.5.3 ML procedure

The *seqAsg* procedure relies on good values of the weights w_{ij} to correctly estimate the quality of the assignment matching pilot i with pairing, day off, or preferred vacation j. Those weights are computed using either a linear or neural network function approximator. The function approximator makes it possible to define an RL policy whose parameterization is learned by an evolutionary algorithm named CMA-ES. We present the general framework for the ML model and RL policy, detailed information about the ML models selected, the reward function used during learning, and describe the CMA-ES algorithm.

ML framework

In order to learn a policy estimating the quality of the assignment of a pairing, day off, or preferred vacation to a pilot and to determine the parameters of the sequential assignment problem presented in Section 4.5.1, we define a learning environment. We also show the sequence of steps to complete one ML training iteration within it.

The environment is defined by states representing the assignment graph at the current time step and a matrix of feature vectors. These feature vectors provide information about the partially completed schedule for each pilot. Actions are defined by selecting one solution for the assignment problem on the current day : the action space is then the set of all possible solutions for that problem. State-action transitions set up the next assignment problem by making feature vector updates based on the pairings, days off, and preferred vacations assigned to the pilots, adding allowable assignment arcs to the new graph and computing their

weights. The new state is then an updated assignment graph and feature matrix. As the assignment problem on day t is fully determined by the state of the environment, the action on that day will always be to select the solution produced by *seqAsg*, as it maximizes the total quality for the assignments made.

To perform an action within the environment, we need a function approximator to compute assignment weights on the arcs of the graph for day t. This function approximator intervenes during state-action transitions as new assignment graphs are set up and ensures that the *seqAsg* procedure can solve the assignment problem, thus performing the action needed. Function approximators must be trained and their parameters learned.

We draw a distinction between the arc weights in each assignment problem and the function approximator's parameters. The arc weights are recomputed at each step of the sequential assignment procedure, i.e., each time we set up the graph for the next assignment problem. The function approximator's parameters, on the other hand, are only updated after a batch of parallel *seqAsg* runs (see below), and only during training iterations. They are used to compute the arc weights.

We also distinguish the function approximator's parameters from the CRP instance's parameters. The CRP instance's parameters, which we want to help scheduling workers adjust as is appropriate and possibly a very large number of times, are related to the number of pilots who are granted specific requests or placed on reserve. They are separate from the function approximator's parameters.

Once the learning environment is defined as above and a function approximator has been chosen, training the function approximator follows a well-defined sequence of steps. Before we introduce these steps, we clarify that we call a *parallel run* a group of *seqAsg* runs for different ML weight vectors tested simultaneously on one scenario ; a batch is the set of parallel runs done on each training scenario. Then, we have that

— First, as per the CMA-ES algorithm, a fixed number of noisy parameter vectors are generated by introducing noise into the approximator's parameters at the current time step.
— Next, each set of noisy parameters is associated with one full run of the *seqAsg* procedure. During each run, one set of noisy parameters is used with the relevant feature matrix to compute assignment weights. This builds a distinct solution associated with each parameter set.
— Then, a reward function scores each roster. The best noisy parameter vectors are those that, through *seqAsg*, generated the rosters ranked the highest by the reward function. This completes a parallel run of *seqAsg*, which is repeated for each training scenario.

— Finally, the original approximator's parameter vector is updated by recombining the best noisy parameters averaged over the entire batch. The search space for the next iteration is updated as well.

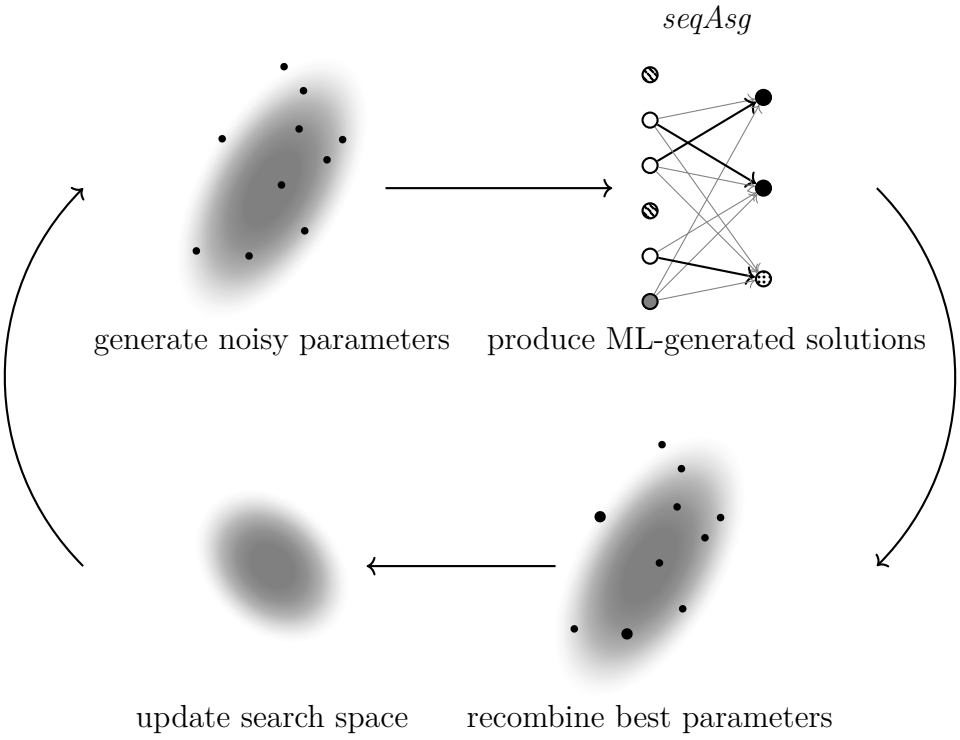


FIGURE 4.2 ML steps per training iteration

The overall sequence is depicted in Figure 4.2. We consider that one full pass over the data for the instance takes m iterations, or one per scenario. Additionally, this sequence of steps only applies to ML training. When testing the models learned during training, only the *seqAsg* procedure comes into play by taking the models to generate solutions.

ML models

We consider two types of function approximators. One is based on a so-called linear function approximator, where a weighted recombination of features scaled to be between zero and one is used. The other is a neural network approximator. Whichever function approximator is used, it is called each time the quality of the assignment of a pairing, day off, or preferred vacation to a pilot must be calculated. However, the architecture and parameters of the function approximator are fixed during a run of *seqAsg* : only the input given to the approximator varies between possible assignments.

Neural networks are made of nodes (i.e., neurons) and arcs connecting them. There are

parameters on the arcs that determine the signals that the nodes on each layer receive. The signals received by the nodes on each layer are processed by an activation function and the output (final) layer gives an action to be made in the environment. The parameters must be learned, for example using stochastic gradient descent or an evolutionary algorithm. See Higham and Higham [110] for an in-depth discussion of neural networks.

For the neural network approximator, we use a hyperparameter search to determine the number of hidden layers and hidden units on each layer. The rectified linear unit (ReLU) activation function was chosen to process the weighted input signal at each neuron of the hidden and output layers. This function is defined as : $f : \mathbb{R} \to \mathbb{R}$, $f(x) = \max(0, x)$ and thus keeps the positive part of the signal.

Each function approximator requires features as input. Traditionally, ML input is a collection of data points. For the purpose of the ML model, a data point is a pilot-pairing pair, with information about the partially built schedule for the pilot at time step t.

Features are related to pilot preferences, pairings, days off, and the overall composition of the partially built schedule for each pilot. The features passed as input are as follows :

- the cumulative number of days off assigned to pilot i before day t (numerical),
- the number of consecutive days off or work days for pilot i before day t, respectively (numerical),
- the satisfaction brought to pilot i by assignment j (numerical),
- the number of credit hours or duties associated with an assignment (both zero for non-pairing assignments), respectively (numerical),
- a binary feature checking if the assignment is a day off (preassigned or not) or preferred vacation (categorical),
- two binary features testing if there are enough credits left for an assignment in the schedule for pilot i at day t, and for future preferred pairings of weights ≥ 75, with or without a 15-hour buffer, respectively (categorical),
- two binary features checking if an assignment conflicts with the next preference for pilot i, or ends the day just before it (categorical), and
- a feature assessing whether a pairing, day off, or preferred vacation is valid within pilot i's schedule, i.e., no conflict or the assignment is worth at least as much as the next preference (categorical).

A bias term is also included as an additional feature.

Reward function

Let r_k for pilot $k \in K$ be the value of the total satisfaction accumulated over a full schedule and let v_k be the penalty incurred if the schedule of pilot $k \in K$ is infeasible. Then, the total

reward R for a roster generated by *seqAsg* during an ML training iteration can be expressed as

$$R = \sum_{k \in K} (r_k - v_k), \quad k \in K, \text{ the set of pilots for a given instance.} \tag{4.12}$$

In our tests, we used $v_k = 1000$, which is a good compromise between favoring improvements in satisfaction and reducing the infeasibility of the generated rosters. Another possibility would be to penalize infeasibility by removing pairings using the correction heuristic presented in Section 4.5.2 and subtracting the penalties associated with them. Reward function (4.12) is generally more conservative with respect to infeasibility, as most infeasible schedules can be corrected by excluding pairings with a total cost below 1000. However, we apply the correction included in the feasibility heuristic in Section 4.5.2 on test CRP instances to see how the function approximators perform in reality. This means that reward function (4.12) is only used during training.

There are two other reasons why unassigned pairings are not taken into account by this reward function. First, if the training instances are adequately parameterized, meaning that the right balance has been found between asking some pilots to report for duty and placing others on reserve or granting them special requests, unassigned pairings are unlikely to happen during training. This is because there is then a sufficient number of pilots to cover all required pairings. On the other hand, if some of the training data comes from instance variations that are in fact unbalanced (in the sense that too many pilots have been excused from regular duty), then the number of unassigned pairings can be simultaneously high and variable from one iteration to the next and one scenario to another, causing a significant unpredictability of the learning process during the first few training iterations. In practice, we have found that (4.12) strikes a good balance between limiting the infeasibility of the generated solutions and maintaining a high level of satisfaction for pilots.

One could wonder why learned weights are necessary for the *seqAsg* procedure to produce good-quality solutions. Indeed, we can imagine solving *seqAsg* with arc weights corresponding to the CRP assignment values. We first note that most of those weights would be zero since most possible assignments do not contribute to fulfilling a pilot's wishes. It is then likely that *seqAsg* would act in a myopic way, choosing assignments that directly fulfill pilot's wishes without considering future possibilities. However, it is often advantageous to consider possible future rewards. For instance, it may be advantageous to select a less valued but shorter pairing if it allows the pilot to take a preferred vacation directly afterwards. These considerations are taken into account by our ML features. Furthermore, the ML updates implemented at the end of each training iteration are based on reward function (4.12), which means that the RL weights are improved based on the quality of an entire roster. In other

words, while assignments are made one day at a time and in sequence, they take into account future information.

CMA-ES algorithm

This section describes the CMA-ES algorithm in more detail and justifies its choice among other RL algorithms. An exhaustive description can be found in Hansen et Ostermeier [96], while different adaptations of the algorithm are detailed in Bäck, Foussette and Krause [97]. The covariance matrix adaptation evolution strategy (CMA-ES) is a type of evolutionary algorithm. This means that, at each iteration, it generates λ real-valued vectors obtained from the vector at the current iteration. These modified vectors are obtained by adding noise following a multinormal distribution. The multinormal distribution and a global variance acting as a step size change dynamically at each iteration, forming a new covariance matrix each time. This means that the search space for better real-valued vectors evolves dynamically. Then, the modified vectors are evaluated with a reward function like (4.12).

Once this is done, the best μ sets of modified parameters are kept and their weighted average is taken as the new parameter vector. The covariance matrix and global step size are also changed. In order to do so, the concept of evolution path is useful. The global step size is adjusted while taking into account both its previous updates (as accounted by one evolution path) and whether the mutations were unusually small or large, as measured by a χ_n distribution for an isotropic normal distribution. The covariance matrix is adjusted based on previous changes for that matrix (as accounted by another evolution path) and the sample noisy covariance matrix at that iteration.

The CMA-ES algorithm requires setting the value of certain hyperparameters, including constants determining the magnitude of updates made to the covariance matrix, the initial global step size and the initial evolution paths. All values for the hyperparameters have been empirically validated in Hansen and Ostermeier [96]. The only constant we set ourselves is the dimension of the search space : it is determined by the architecture of the function approximator chosen for the ML model.

While it is more traditional to learn an RL policy's parameters using some version of stochastic gradient descent, the evolutionary algorithm presented above was chosen for two reasons. First, evolutionary algorithms have been found to be competitive on a variety of RL problems and are therefore a sensible option [100]. Also, in order to do gradient updates using standard actor-critic methods for RL, the policy needs to be stochastic. If it is deterministic, as in our case, there is the possibility of either using natural gradients (e.g., with natural evolution strategies, an alternative to CMA-ES quite close to standard policy gradients but doing gradient updates over the parameters generated by a probability distribution) or with

methods such as those presented in Silver *et al.* [111]. We tested NES and did not get better results than with CMA-ES ; furthermore, our evolutionary algorithm is a simpler alternative to deterministic actor-critic methods.

4.6 Results

In this section, we present results for the *seqAsg* method. We show that the solutions generated with the algorithm described in Section 4.5 can help scheduling planners gain insight into CRP instances and parameterize them more appropriately. In Section 4.6.1, we give details of the implementation of our ML model and the optimization solver used for the CRP. In Section 4.6.2, we consider the quality of the solutions produced by the model learned at the end of the ML training process. In Section 4.6.3, we establish the relationship between the quality of an ML-generated solution and the parameterization of a CRP instance. We do so by observing how the CRP objective in the optimization solver is associated with the quality of the ML-generated solution. Finally, in Section 4.6.4, we compare the *seqAsg* procedure with ML weights to another very fast heuristic created for this purpose.

4.6.1 Methodology

We now provide the methodology used to train the ML model presented in Section 4.5.3. For each instance, 25 out of the 30 available scenarios are kept for training, while the remaining 5 are set aside for testing after the entire ML process is complete. Training is performed on each instance separately to get the ML model for that instance, but we test the model obtained on all 7 instances. For example, if we train an ML model on instance I1 on the 25 first scenarios for that instance, we test the learned model by running the *seqAsg* procedure on the 5 last scenarios for all 7 instances. We do the same for each instance. In the absence of undue overfitting, all generalizable ML models should yield similar results on the test scenarios.

For both approximators, we use early stopping to limit overfitting. In our experiments, we stopped training after 120 iterations. For the neural network approximator, it was found that the algorithm learned best with a 2-layer net. The hidden layers had 13 and 10 neurons, respectively, for a total of 320 parameters. The *seqAsg* procedure was implemented in Python v. 3.7.1 through the package *pulp* v. 2.6.0. The *pulp* module in turn calls the integrated CBC solver v. 2.10.3 to solve the assignment problem defined by (5.1)-(5.5). CPLEX 12.9 was also considered and exhibited similar performance as CBC.

To compare different CRP solutions, we define the following quantities. Let S_k^{best} be the

best feasible schedule for pilot $k \in K$. S_k^{best} is obtained by solving the column generation subproblem for each pilot, without dual information (all dual variables equal to 0). In other words, it is found by considering the cost of each arc in the pilot's subproblem rather than the reduced cost as would typically be the case. Let $S^{best} = \sum_{k \in K} S_k^{best}$ be an upper bound on the total satisfaction, and thus, on the CRP optimal value. We use S^{best} as a reference point to compare different solutions of the same instance. In addition, S^{best} is easily computed (in less than one second) and is independent of any solution method, making it ideal to compare different CRP solutions.

In practice, S^{best} is almost never achieved because of conflicting preferences among employees. However, we observe that for well-parameterized problems, GENCOL finds solutions whose satisfaction is very close to S^{best}. In fact, it has been our experience that when a CRP instance is well parameterized, the overwhelming majority of the pairings and days off accounting for pilot satisfaction in the schedules with total satisfaction S^{best} are retained for each pilot when the CRP is solved by GENCOL. However, this is not the case for poorly parameterized instance variations.

Consider a CRP solution of value $z = S - U - V$, with S its satisfaction (first term of 4.1), U the penalty due to missed coverage (second term of 4.1), and V the penalty due to missed preassigned days off (third term of 4.1). We define the *satisfaction loss* as

$$L^{sat} = 1 - \frac{S}{S^{best}}$$

Likewise, we express the *coverage loss* and the *preassignment loss* relative to S^{best} :

$$L_{ML}^{cover} = \frac{U}{S^{best}}$$

$$L^{preasg} = \frac{V}{S^{best}}$$

Note that L^{preasg} does not appear in the results because it is always zero in our observations. Furthermore, L_{ML}^{cover} is defined here only for ML solutions. While one could also define a variable for the loss of objective due to pairings unassigned to any pilot in standard GENCOL rosters, this measure would be more useful to get a more detailed breakdown of any objective loss in GENCOL solutions, for example by examining any correlation between the cover loss in ML solutions and GENCOL rosters. Since scheduling planners are primarily interested in predicting the global loss of objective to choose whether to re-determine how many pilots

work regular duty, are on reserve, or are assigned special activities and requests, we focus on using L_{ML}^{cover} to infer about GENCOL's behavior.

Let S_{ML} and L_{ML}^{sat} be the total satisfaction and the satisfaction loss for ML-generated solutions. Likewise, let S_{GENCOL} and L_{GENCOL}^{sat} be the total satisfaction and the satisfaction loss for solutions generated by GENCOL.

4.6.2 ML-generated solution quality

We now report on ML-generated solution quality. We do so with the intent of verifying if a relationship between L_{ML}^{sat} and L_{GENCOL}^{sat} is apparent for the instances described in Section 4.4. A more detailed and expanded analysis of the conclusions found in this section is presented next in Section 4.6.3 with many different instance variations.

Table 4.4 shows the average value of L_{ML}^{sat}, which includes the penalties applied for pairings unassigned to any pilot, for each validation instance and each ML model trained. Each row corresponds to a different ML model (one for each training instance) and each column to a different set of validation scenarios (split by instance). In each case, the ML model learned over the training instance is used in the *seqAsg* method to generate five different solutions per validation instance (one for each validation scenario). This means that columns represent the results obtained for different ML models, trained on different instances, for a same validation instance. We report these results for both the linear and the neural network function approximators. For each validation instance, we also report the mean value of L_{ML}^{sat}, as well as the mean value of L_{GENCOL}^{sat} for all seven instances. Table 4.4 also shows the average pairing-to-pilot ratio. What constitutes a high ratio is instance-specific. Finally, we report the training time in minutes to learn the function approximator's parameters, per instance. Smaller instances allow training the approximator faster and are therefore preferred to larger instances during training when performance across instances is comparable during testing. All values in the table are accompanied by the standard deviation for the sample of scenarios involved in their computation.

From the results in Table 4.4, a few observations stand out. First, in all but one instance (where the difference is negligible), the neural net performs better than the linear approximator. Next, there is little variation in performance for any given validation instance. This suggests there is little overfitting in general, as the performance on a validation instance is generalizable no matter which ML model is used, i.e., no matter which instance was used for training. Following up on the discussion brought up in Section 4.4, we see how in the absence of historic data, scenarios could be generated on a single instance and how a model as we describe could be trained on them. There is generally little variation between performance

on different scenarios (within an instance variation), just as is the case between instances. This means that the average values reported in Table 4.4 are close to those for individual validation scenarios. Additionally, since our instances are varied, Table 4.4 suggests that the *seqAsg* procedure is consistent even on instances whose compositions, as understood from the lens of instance size and the length of its pairings, differ substantially.

TABLEAU 4.4 Average L_{ML}^{sat} (%) for each training set/test set combination (two approximators)

Training instance	Validation instance							Pairings/pilot	Time (min)
	I1	I2	I3	I4	I5	I6	I7		
Linear function approximator									
I1	49.3 (11.3)	71.0 (19.2)	68.4 (6.0)	54.9 (3.8)	19.6 (1.7)	31.5 (3.2)	14.8 (1.5)	5.81	70
I2	44.1 (3.6)	65.7 (13.7)	68.4 (10.9)	56.0 (4.1)	14.6 (2.2)	29.0 (3.8)	13.6 (1.2)	12.26	82
I3	47.1 (5.4)	62.3 (4.6)	62.9 (6.5)	52.5 (4.1)	22.0 (1.6)	27.2 (3.5)	16.4 (2.3)	9.70	80
I4	46.2 (1.8)	61.8 (8.5)	60.8 (7.2)	50.7 (5.0)	15.6 (1.7)	24.2 (3.1)	14.6 (1.2)	8.87	220
I5	43.8 (5.3)	64.1 (2.8)	60.1 (4.3)	56.5 (6.2)	12.2 (1.8)	22.9 (2.5)	11.0 (1.0)	5.11	579
I6	43.1 (4.4)	65.9 (9.3)	64.4 (8.8)	55.3 (7.0)	12.2 (1.6)	24.2 (1.8)	11.0 (1.2)	5.98	310
I7	41.4 (3.3)	63.0 (11.5)	61.5 (4.1)	61.2 (6.7)	14.6 (2.3)	23.9 (2.6)	10.7 (1.0)	4.57	717
Mean (L_{ML}^{sat})	**45.0 (5.8)**	63.4 (10.6)	62.4 (7.6)	55.3 (6.6)	15.8 (4.1)	26.1 (4.0)	13.2 (2.5)		
Mean (L_{GENCOL}^{sat})	17.4 (2.3)	17.2 (5.2)	17.6 (2.4)	10.0 (1.6)	2.4 (0.9)	4.2 (0.9)	2.1 (1.6)		
Neural function approximator									
I1	50.3 (3.5)	55.8 (8.9)	57.8 (14.1)	53.8 (3.7)	16.4 (0.9)	26.3 (1.8)	13.6 (0.9)	5.81	127
I2	46.2 (2.0)	53.0 (11.8)	59.2 (7.2)	53.7 (9.8)	14.8 (1.3)	26.5 (3.4)	12.2 (1.5)	12.26	134
I3	44.1 (3.9)	51.8 (8.4)	55.2 (8.5)	45.8 (6.3)	17.5 (1.8)	24.0 (1.4)	12.8 (1.3)	9.70	132
I4	40.6 (4.5)	51.6 (9.9)	56.7 (6.6)	53.7 (6.4)	14.4 (1.8)	23.3 (2.5)	13.3 (1.0)	8.87	345
I5	46.7 (9.8)	51.5 (10.5)	58.7 (6.5)	53.2 (6.3)	11.4 (1.2)	20.0 (2.8)	10.5 (1.3)	5.11	794
I6	43.8 (6.5)	51.9 (7.4)	66.5 (6.7)	52.5 (11.7)	12.4 (1.0)	20.2 (2.4)	11.1 (1.2)	5.98	492
I7	44.7 (4.9)	56.8 (8.6)	65.7 (7.8)	54.1 (3.2)	11.9 (1.4)	21.5 (2.8)	11.3 (1.2)	4.57	1174
Mean (L_{ML}^{sat})	45.2 (10.0)	**53.2 (8.8)**	**58.5 (9.9)**	**52.4 (6.0)**	**14.1 (2.5)**	**23.1 (3.5)**	**12.1 (2.0)**		
Mean (L_{GENCOL}^{sat})	17.4 (2.3)	17.2 (5.2)	17.6 (2.4)	10.0 (1.6)	2.4 (0.9)	4.2 (0.9)	2.1 (1.6)		

There is a drastic difference in performance between validation instances. In particular, instances I1 to I4 are associated with a loss of satisfaction that is greater than for instances I5 to I7. In the case that performance variability is due to the quality of an instance's parameterization, this would mean that good instances (i.e., balanced in terms of pilots assigned for regular duty as opposed to special tasks and requests, well-parameterized) are associated with solutions in the GENCOL solver whose objective is close to S^{best}. These results confirm the interest in trying to figure out a relationship between L_{ML}^{sat} and L_{GENCOL}^{sat}.

One possibility is that some of our instances are poorly parameterized and do not include enough pilots ready to carry out pairings. This may cause uneven performance across instances after the ML-based sequential assignment procedure is done. In that case, the higher the number of pairings each pilot must cover on average in their schedule, the more difficult the problem is. However, testing for this hypothesis requires more than the data presented

here, and the notion of instance variation introduced in Section 4.4 is useful to cover a larger span of ML solutions than is available in Table 4.4, as mentioned above.

Another interesting fact is made explicit by the pairing-to-pilot ratio by instance. It can be seen from Table 4.4 that if two instances have different values for that quantity, the instance with the lower ratio does not necessarily have ML solutions with lower values of L^{sat}_{ML}. This is made evident by comparing the results for instance $I1$ to instances $I5$-$I7$. This is consistent with industry situations, as the ideal number of pilots to schedule for duty will vary based on the situation and with such factors as the length of the pairings or the presence of bottlenecks (i.e., certain days of the month when the pilots available are particularly solicited). In any case, the unevenness in the association between pairing-to-pilot ratio and ML solution quality between instances suggests that L^{sat}_{ML} is better than the pairing-to-pilot ratio as an indicator of the behavior observed in L^{sat}_{GENCOL}.

With regard to training time, there is a positive association between instance size and computing time. This observation combined with the reusability of weights learned on small instances for larger instances suggests that if training time is an issue, learning on smaller instances may be more advisable. Nonetheless, an important caveat is that training only needs to happen once, while the *seqAsg* procedure can be run an unlimited number of times as needed by the scheduling workers : for that reason, it is less of an issue if training takes time on larger instances and this method may also be preferred by certain companies.

4.6.3 Linking ML-generated solutions with solver performance

Based on the observations in Section 4.6.2, we now seek to establish a robust relationship between the solution generated with the *seqAsg* procedure and the performance of the GENCOL solver. This can give insight into a CRP instance's parameterization. We do so by considering many instance variations, aimed at providing a good range of different levels of difficulty (i.e., we vary how many pilots are available for a fixed set of pairings) for each instance. The results for these experiments are shown in Table 4.5. This table shows, for each instance variation, how many pilots are part of it as well as the corresponding pairing-to-pilot ratio. We also report the sum of best possible satisfactions (S^{best}), the ML-generated solution total satisfaction (S_{ML}) and mean satisfaction loss (L^{sat}_{ML}), the GENCOL total satisfaction (S_{GENCOL}) and mean loss (L^{sat}_{GENCOL}) and the loss due to unassigned pairings (L^{cover}_{ML}). Finally, we report the average computing time for GENCOL over the five scenarios. For instance variations associated with a computing time in GENCOL greater than 8 hours, we have sometimes used less than five scenarios (e.g., only one if the computing time exceeds one week). We have not noticed major differences in the variability of the solutions in those cases.

Also, we note that the speed to obtain the ML solutions is not mentioned. This is because, in our experience, the *seqAsg* procedure has always taken less than 15 seconds (under 10 seconds for smaller instances). It also increases only slightly with instance size. Given that for large instances especially, such a speed approaches immediate resolution, we consider the time taken by *seqAsg* to provide an ML solution negligible in those cases.

From Table 4.5, a number of observations can be made. First, predictably, both ML-generated and GENCOL solution objectives decrease as the total number of pilots available for an instance is reduced. On the other hand, in the same situation, computing times and the loss of satisfaction due to unassigned pairings increase. This is remarkable as a more limited number of pilots means that there are fewer subproblem networks during column generation, which reinforces the idea that not scheduling enough pilots for regular duty increases the complexity of crew rostering. Additionally, at first glance, it seems like the decrease for the GENCOL objective follows the decrease in ML solution quality in an approximatively linear fashion, and this is true across all 7 instances. In particular, the values for L^{sat}_{ML} and L^{sat}_{GENCOL} range from 3.4% (I5-01) to 86.1% (I6-07) and from 1.3% (I5-01) to 33.8% (I6-07), respectively, and seem to covary.

Furthermore, computing time increases rapidly as fewer pilots are available, while L^{cover}_{ML} starts exceeding zero for most instances once enough pilots have been removed from scheduling duty. This is significant, as when the problem increases in complexity, it becomes harder to generate a solution that meets all pairing coverage constraints. We also observe that when the number of pilots scheduled for regular duty is sufficient, L^{cover}_{ML} is zero and only increases slowly at first, but can dominate quickly (for example as in the cases of instance variations I3-07, I6-07 and I7-07). The extent to which this finding is significant depends on the scheduling planner's needs. In the case that a planner is less interested in getting a precise estimate of the value for L^{sat}_{GENCOL} as the instance is currently parameterized, then a condition on L^{cover}_{ML} could enforce that it be zero to keep L^{sat}_{GENCOL} high. Indeed, it seems that when our sequential assignment procedure generates a fully feasible roster and all CRP constraints are met, L^{sat}_{GENCOL} is close to as low as it can possibly get.

To further explore the relationship between L^{sat}_{ML} and L^{sat}_{GENCOL}, we use the data in the table and fit a regression model with $\log(L^{sat}_{ML})$ as an explanatory variable and $\log(L^{sat}_{GENCOL})$ as a predicted variable. It is then possible to draw conclusions from the results of this analysis, which we perform for the four largest instances (I4-I7). Results are presented in Table 4.6. The first row of the table gives general information about the linear regression model fit to the data. It shows the explanatory and predicted variables, the regression method (ordinary least squares, also written OLS), the number of data points part of the model fit, the degrees

Tableau 4.5 Relationship between ML-generated and final solutions in the solver by instance and pairing-to-pilot ratio

Variation	Nb pilots	Pairings/pilot	S^{best}	S_{ML}	L^{sat}_{ML} (%)	S_{GENCOL}	L^{sat}_{GENCOL} (%)	Time (s)	L^{cover}_{ML} (%)
I1-01	52	3.0	39603	36736	7.2	38861	1.9	0.5	0.0
I1-02	45	3.5	34173	31075	9.1	33379	2.3	0.7	0.4
I1-03	39	4.0	29820	26013	12.7	28795	3.4	0.7	0.5
I1-04	35	4.5	26350	21737	17.5	24844	5.7	1.1	1.8
I1-05	31	5.0	23521	17345	26.3	21248	9.7	1.5	4.4
I1-06	29	5.5	22106	14900	32.6	19824	10.3	2.0	6.9
I1-07	26	6.0	19721	10126	48.7	15596	20.9	3.2	14.3
I2-01	66	5.0	55333	52352	5.4	53959	2.8	7.1	0.0
I2-02	55	6.0	45323	42347	6.6	44336	2.2	8.5	0.4
I2-03	47	7.0	40013	37226	7.0	39210	2.0	9.3	0.0
I2-04	41	8.0	34653	32051	7.5	33562	3.1	16	0.5
I2-05	37	9.0	30320	27180	10.4	28984	4.4	19	0.9
I2-06	33	10	27670	22757	17.7	25977	6.1	31	4.6
I2-07	30	11	25289	17769	29.7	22796	9.7	51	11.0
I3-01	90	4.0	70239	64223	8.6	69050	1.7	16	0.0
I3-02	72	5.0	56660	51393	9.3	55449	2.1	19	0.0
I3-03	60	6.0	47369	42136	11.0	46301	2.3	24	0.2
I3-04	51	7.0	40729	35001	14.1	39492	3.0	37	1.4
I3-05	45	8.0	35157	28540	18.8	32749	6.8	108	4.2
I3-06	40	9.0	31878	20727	33.9	27755	12.9	128	13.7
I3-07	36	10	28106	10057	64.2	21250	24.4	807	33.2
I4-01	260	4.0	198915	186702	6.1	195627	1.7	223	0.0
I4-02	208	5.0	159531	149490	6.3	156993	1.6	268	0.0
I4-03	173	6.0	133135	121948	8.4	131448	1.3	291	0.0
I4-04	160	6.5	124436	113766	8.6	121972	2.0	438	0.0
I4-05	148	7.0	112619	99242	11.9	110526	1.9	556	0.5
I4-06	138	7.5	103774	88277	14.9	101282	2.4	767	1.4
I4-07	130	8.0	99654	77259	22.5	94912	4.8	1268	4.6
I4-08	122	8.5	93092	59246	36.4	86432	7.2	2474	12.2
I4-09	115	9.0	89097	35563	60.0	77428	13.1	4047	29.5
I5-01	407	3.0	300987	290631	3.4	297201	1.3	324	0.0
I5-02	349	3.5	260584	251506	3.5	256942	1.4	365	0.0
I5-03	306	4.0	226636	216858	4.3	223568	1.4	421	0.0
I5-04	272	4.5	203401	191348	5.9	200160	1.6	467	0.1
I5-05	244	5.0	182711	164968	9.7	177850	2.7	656	0.6
I5-06	222	5.5	165992	136962	17.5	159451	3.9	936	3.5
I5-07	204	6.0	150899	95930	36.4	135540	10.2	> 8 hrs	16.2
I6-01	269	4.0	204453	194498	4.9	201366	1.5	124	0.0
I6-02	240	4.5	180873	171003	5.5	178278	1.4	150	0.0
I6-03	215	5.0	161289	147172	8.8	158692	1.6	190	0.0
I6-04	196	5.5	146884	129457	11.9	143157	2.5	244	0.0
I6-05	179	6.0	135290	106414	21.3	128947	4.7	470	2.4
I6-06	166	6.5	125294	65755	47.3	112185	10.4	> 8 hrs	21.4
I6-07	154	7.0	117058	16282	86.1	77483	33.8	> 8 hrs	50.5
I7-01	491	3.0	367439	352302	4.1	361987	1.5	455	0.0
I7-02	421	3.5	316157	300478	5.0	311402	1.5	516	0.0
I7-03	368	4.0	274031	257204	6.1	269206	1.8	707	0.0
I7-04	327	4.5	244622	220567	9.0	239787	2.0	897	0.1
I7-05	295	5.0	218413	177926	18.5	209831	3.9	1552	2.5
I7-06	268	5.5	201112	128838	35.9	182775	9.1	> 8 hrs	14.9
I7-07	246	6.0	184138	63781	65.4	136215	26.0	> 8 hrs	36.8

of freedom of the model used to compute the F-statistic and t-statistics elsewhere in the table, the number of degrees of freedom in the model, and the covariance type (i.e., robust to heteroscedasticity or not). The second row includes information about the quality of the model fit : the R-squared and adjusted R-squared coefficients, the F-statistic and associated probability that such a statistic would be reached if the model coefficients were all zero, as well as other goodness of fit statistics such as the log-likelihood and the AIC and BIC

(Aikake information criterion and Bayesian information criterion, respectively). The next two rows give information about the regression parameters : their values, standard errors, t-statistics, the probability of reaching such values of the (absolute value of the) t-statistic if the coefficients are zero, and the 95% confidence intervals for the coefficient values.

By analyzing Table 4.6, certain observations stand out. First, the values for R-squared and adjusted R-squared are similar and very high : this R-squared value suggests that 91.9% of the variance in $\log(L^{sat}_{GENCOL})$ is explained by $\log(L^{sat}_{ML})$. In other words, the loss of satisfaction in our ML solutions explains almost all the variance in loss of satisfaction in GENCOL according to a linear relationship. Furthermore, the F-statistic suggests the non-trivial model is statistically significant, with both t-statistics suggesting that the intercept and the explanatory variable coefficients are each different from zero. Finally, the association between $\log(L^{sat}_{ML})$ and $\log(L^{sat}_{GENCOL})$ is positive, as would be expected. Altogether, this summary of the regression analysis in Table 4.6 allows us to conclude that a strong relationship between L^{sat}_{ML} and L^{sat}_{GENCOL} exists and that the model's fit is excellent.

Tableau 4.6 Regression analysis for OLS fit between $\log(L^{sat}_{ML})$ and $\log(L^{sat}_{GENCOL})$, instances I4-I7

Expl.	**Pred.**	**Method**	**Obs.**	**Df (residuals)**	**Df (model)**	**Cov.**
$\log(L^{sat}_{ML})$	$\log(L^{sat}_{GENCOL})$	OLS	30	28	1	nonrobust
R^2	**Adj. R^2**	**F-stat.**	**Pr(F-stat.)**	**Llk**	**AIC**	**BIC**
0.919	0.916	315.7	8.82e-17	-13.359	30.72	33.52
Variable	**Coef.**	**Std err.**	**t-stat**	**Pr(t-stat)**	**LC**	**UC**
intercept	-1.7971	0.206	-8.713	0.000	-2.220	-1.375
$\log(L^{sat}_{ML})$	0.9486	0.053	17.768	0.000	0.839	1.058

To better visualize the effect of an increasing value of L^{sat}_{ML}, we represent the observations from Table 4.5 with Figure 4.3. In this figure, for each instance variation presented in Table 4.5, we associate one data point in the scatterplot. Each point represents the $(L^{sat}_{ML}, L^{sat}_{GENCOL})$ coordinate pair for the instance variation. The horizontal axis represents the value of L^{sat}_{ML} in percentage for the solution generated by *seqAsg* and for that instance variation. The vertical axis gives the corresponding value of L^{sat}_{GENCOL} for the same data point. Both axes are on a base 10 logarithmic scale to remain consistent with the model variables and for better visibility. We then fit the data using the model presented in Table 4.6 and trace the regression line for the best OLS fit. The best OLS fit shows a clear positive and very strong association between $\log(L^{sat}_{ML})$ and $\log(L^{sat}_{GENCOL})$, thereby reinforcing the conclusions drawn from the regression analysis summary presented in Table 4.6. Another important point of consideration is that, based on the observed data, the variance of the model seems mostly uniform across the range of different values of L^{sat}_{ML} for the data. This suggests that there is

no heteroscedasticity, as previously assumed, and that the data was fit properly.

Both Table 4.6 and Figure 4.3 seem to support the notion of a linear association between $\log(L^{sat}_{ML})$ and $\log(L^{sat}_{GENCOL})$ that informs as to the expected loss of satisfaction when solving the CRP in GENCOL. We now evaluate the usefulness of obtaining information in such a way. As mentioned in the introduction, CRP scheduling planners need to make many, sometimes hundreds of decisions throughout a month when creating the instance for the next month. Absent any suitable analysis tool, they must check whether their newly modified instance is likely to give a good objective after resolution in GENCOL. However, Table 4.5 suggests the time required to do that becomes prohibitive for large instances once L^{sat}_{ML} becomes too large. This means that building an instance that includes enough pilots scheduled to fly (i.e., not granting too many requests or placing too many pilots on reserve) takes an unreasonable amount of time for instances that are not adequately parameterized. In this way, the *seqAsg* procedure allows testing the adjustments made by the scheduling workers within seconds while giving them insight into whether the instance is overly difficult to solve. For example, it allows them to check if it can be solved to most pilots' satisfaction while granting as many requests as possible, including during special events.

Another aspect worth reiterating is the speed at which we produce feasible CRP solutions fit to help with schedule planning. It is unparalleled in previous research so far, as past ML research (see Section 4.2) has mostly focused on solving the CRP either by enhancing the work of a solver or by using complex heuristics. While these methods often match or even slightly exceed objective quality for well-known benchmark instances, they always take at least several minutes to run (a duration that may be shorter than standard resolution but still far exceeds the 15 seconds or less for our method). Furthermore, it has been our observation that the 10 to 15-second duration does not increase much with instance size. There are also instances larger than even our largest instances (up to 50000 flights, see Yaakoubi, Soumis and Lacoste-Julien [33]) that may take hours to solve through traditional means or heuristics. The ML weights only having to be learned once, the *seqAsg* procedure is largely immune to this issue and it would likely be of similar speed in that context as well, therefore allowing scheduling planners to do their work in a fraction of the time it would normally take. For this reason, while our solutions are not perfect and are not taken as an end product in themselves, they are very fast and are still strong enough to provide useful information : this is not something addressed by any alternative method yet.

With these considerations in mind, a rigorous evaluation of our ML-based method's usefulness needs to take into account potential methods that would also be very fast in providing important information to scheduling planners. We do so in Section 4.6.4.

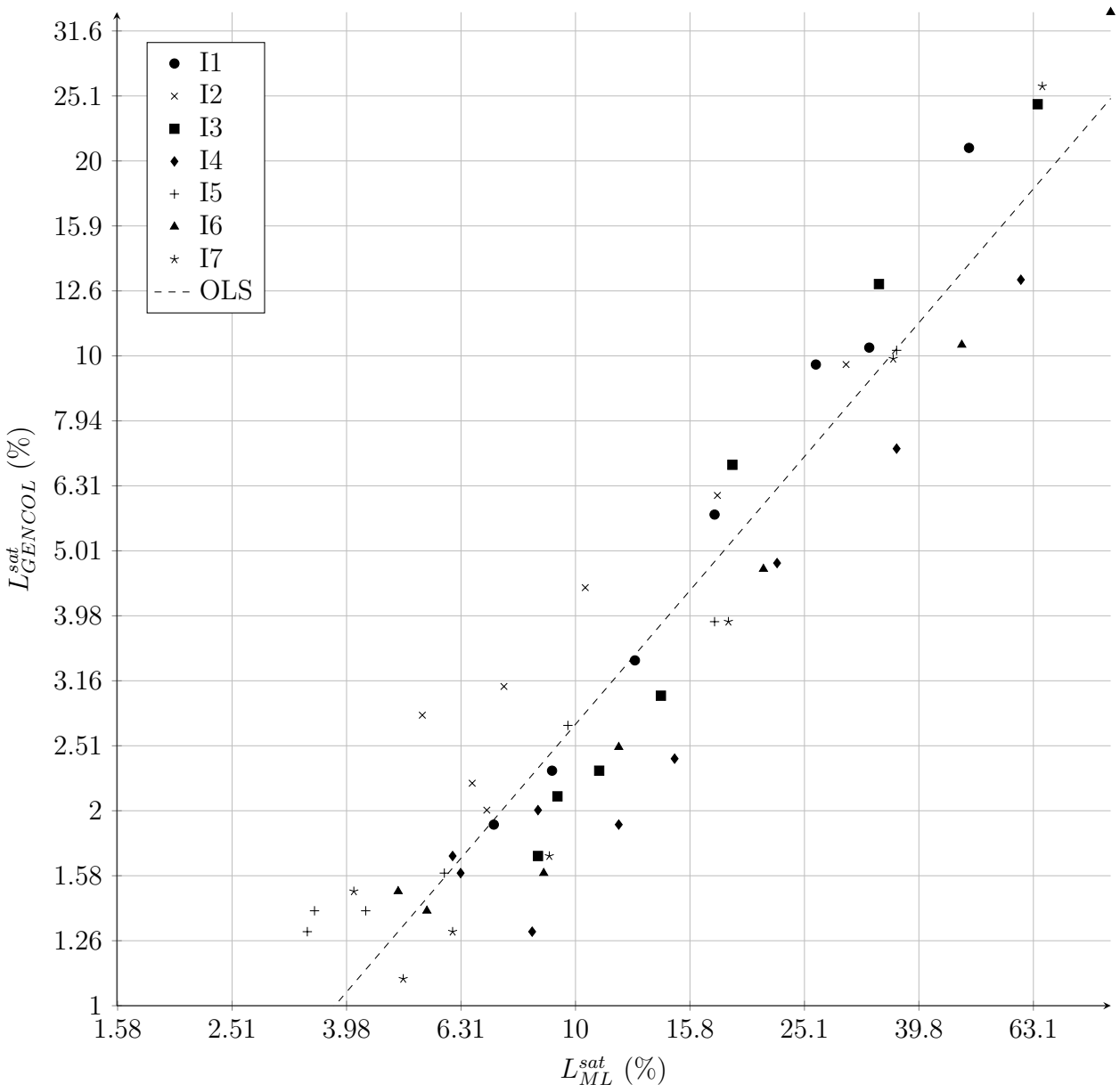


FIGURE 4.3 Log-log relationship between L^{sat}_{ML} and L^{sat}_{GENCOL} for many instance variations

4.6.4 Fast GENCOL heuristic

We now seek to assess the appropriateness of our method, especially with regard to the use of ML as opposed to other types of algorithms. Not having found in the literature any approach similar to *seqAsg* in terms of speed, as a comparison point, we provide a simple heuristic that can be directly implemented in GENCOL. We define this heuristic as follows.

First, we focus on changing certain parameters in the GENCOL solver to converge towards a solution very quickly. Since the goal is to offer a heuristic as an alternative to *seqAsg*, we seek

to bring computational time below 60 seconds and in most cases, below 30 seconds for the linear relaxation of the problem (in comparison to *seqAsg*'s 10-15 seconds for the full solution depending on the instance). To achieve this goal, we change the value of certain parameters in GENCOL. The parameters changed in this way fall into three categories : parameters related to resource dominance during the subproblem phase of column generation, parameters related to stopping criteria (e.g., number of rolling iterations and relative improvement of the objective over these iterations), as well as parameters related to branching. Once the correct configuration of parameters is found, we run the input file for one scenario for each instance variation associated with the four largest instances. We then note the objective value and time taken by the solver once the final solution is obtained. We also take note of the objective value and computing time once the linear relaxation is solved, i.e., before branching starts. We refer to these values as those obtained once branching node zero is solved.

The results for the GENCOL heuristic are shown in Table 4.7.

Let L^{sat}_{heur} be the satisfaction loss for solutions produced by the accelerated GENCOL heuristic. Table 4.7 shows the loss of satisfaction for the heuristic L^{sat}_{heur} against S_{best} for the solutions obtained after solving node zero or after the full resolution of the integer problem, as well as the loss of satisfaction L^{sat}_{GENCOL} for the full solution obtained without the heuristic (i.e., with the standard configuration of the solving parameters in GENCOL). The first column shows the instance variations considered. We present the objective value, solving time and loss of satisfaction L^{sat}_{heur} both for the relaxed and the full solution in the six following columns, while the last two show L^{sat}_{GENCOL} and S_{best} by instance variation.

A closer examination of the data reported in Table 4.7 suggests the value of L^{sat}_{heur} for the relaxed heuristic solutions is associated more closely with L^{sat}_{GENCOL} than the loss of satisfaction observed in the final heuristic solutions. The relaxed solutions are also considerably faster to get. For example, in the final heuristic solution for instance variation I4-06, we observe a value of L^{sat}_{heur} of 30.4% while L^{sat}_{GENCOL} stands at 1.49%, whereas for instance I7-07 we have that L^{sat}_{heur} and L^{sat}_{GENCOL} are 32.8% and 26.0% respectively. For this reason, we focus on comparing the goodness of fit for OLS models based on the *seqAsg* predictions and the heuristic applied to relaxed CRP solutions.

The observations made above were a surprising finding, given that one might expect that a full solution would convey more useful information to scheduling workers than a relaxed solution. This initially made the accelerated resolution in GENCOL the natural candidate to consider. Once we noticed that the relationship between L^{sat}_{heur} and L^{sat}_{GENCOL} seemed weak in that specific case, we realized that the change in branching parameters put in place to converge towards a full solution very quickly in GENCOL caused a massive drop of performance

TABLEAU 4.7 Comparison between standard resolution and a fast heuristic in GENCOL

Variation	Node zero			Final solution			L^{sat}_{GENCOL}	S^{best}
	Objective	Time	L^{sat}_{heur} (%)	Objective	Time	L^{sat}_{heur} (%)		
I4-01	167439	13.0	16.3	183575	63.5	8.27	1.87	200120
I4-02	133925	20.4	16.1	142515	82.1	10.0	1.04	158405
I4-03	111861	19.3	16.1	111370	97.6	16.4	1.45	133305
I4-04	105662	20.7	15.4	97770	118	21.8	4.36	124965
I4-05	94681	28.3	16.9	84000	127	26.3	1.83	113995
I4-06	91139	45.2	12.2	72260	145	30.4	1.49	103810
I4-07	77084	29.9	22.6	68315	150	31.4	5.89	99575
I4-08	72734	50.9	19.9	50575	162	44.3	6.75	90765
I4-09	62532	38.7	30.9	43825	211	51.6	12.1	90515
I5-01	259385	18.6	13.7	288545	127	4.05	1.52	300725
I5-02	214201	21.4	18.1	246455	129	5.81	1.27	261650
I5-03	193413	28.6	14.7	213435	146	5.85	1.06	226695
I5-04	166305	28.6	15.9	177670	148	10.2	1.56	197795
I5-05	161103	39.7	13.0	163495	156	11.7	3.03	185250
I5-06	133506	36.8	19.4	141090	188	14.8	3.87	165595
I5-07	112967	52.7	25.9	122205	300	19.8	10.2	152445
I6-01	178789	10.7	11.7	185770	42.8	8.27	1.49	202525
I6-02	155475	9.6	14.7	170075	47.0	6.65	1.47	182200
I6-03	130926	10.4	18.8	142050	53.7	11.9	0.82	161230
I6-04	121072	14.1	17.7	130410	69.1	11.4	3.44	147130
I6-05	99175	13.1	27.0	115890	85.2	14.7	5.44	135815
I6-06	85966	20.3	31.7	94540	142	24.8	10.4	125780
I6-07	59519	22.6	48.6	66595	130	42.4	33.8	115700
I7-01	312847	28.6	14.9	353900	163	3.77	1.23	367775
I7-02	258663	34.7	17.6	295685	187	5.81	1.34	313915
I7-03	227988	40.0	16.7	251965	197	7.94	1.19	273695
I7-04	197805	44.0	18.5	223450	219	7.93	1.68	242695
I7-05	174219	49.8	21.3	197500	250	10.8	3.72	221375
I7-06	144591	66.9	28.1	163185	590	18.8	9.1	201030
I7-07	100131	55.8	46.3	125185	563	32.8	26.0	186310

between the values of the objective after node zero and the final objective. This means that a full solution, far from adding information, requires sacrificing performance to such an extent that the resulting objective is less informative about L^{sat}_{GENCOL}. This explains why we have also included the version of the heuristic concerned with the objective for the linear relaxation of the problem, and why it performs better.

We also performed a regression analysis on the results of Table 4.7. Results are presented in Table 4.8, in the same fashion as Table 4.6. Comparing the two tables allows us to make several observations. First, while the heuristic model captures most of the variance in the explained variable, $\log(L^{sat}_{GENCOL})$, our model does much better by capturing most of the variance unexplained by the GENCOL heuristic. This is a meaningful difference, as the

precise prediction of GENCOL's behavior allows us to detect more accurately the small losses in pilot satisfaction that can be expected for the instance under consideration. Given that even small differences in satisfaction can substantially impact pilots, this is not trivial. Furthermore, both models are significant and both the intercept and the explanatory variable are significant in each case as well. However, the goodness of fit for the *seqAsg* OLS model is better than for the model based on the heuristic. The R-squared is 0.919 for the former and 0.754 for the latter ; all three values for the log-likelihood, the AIC, and the BIC favor the model based on the ML solutions as the best fit on prospective new data.

TABLEAU 4.8 Regression analysis for OLS fit between $\log(L^{sat}_{heur})$ (node zero) and $\log(L^{sat}_{GENCOL})$, instances I4-I7

Expl.	**Pred.**	**Method**	**Obs.**	**Df (residuals)**	**Df (model)**	**Cov.**
$\log(L^{sat}_{heur})$	$\log(L^{sat}_{GENCOL})$	OLS	30	28	1	nonrobust
R^2	**Adj. R^2**	**F-stat.**	**Pr(F-stat.)**	**Llk**	**AIC**	**BIC**
0.754	0.745	85.87	5.03e-10	-32.075	68.15	70.95
Variable	**Coef.**	**Std err.**	**t-stat**	**Pr(t-stat)**	**LC**	**UC**
intercept	-8.8831	1.138	-7.809	0.000	-11.213	-6.553
$\log(L^{sat}_{ML})$	2.4509	0.264	9.267	0.000	1.909	2.993

It is worth detailing some of the reasons that may explain why our sequential assignment procedure performs better than the GENCOL heuristic that we have implemented. In particular, we have that the columns generated by the heuristic are not guided by any auxiliary information. When a time constraint is added to the entire process, this means that the solver must largely find a solution that is based on a random and limited set of schedules. Consequently, as discussed previously, with branching in place to reach an integer solution, there is a great amount of loss of quality due to this element of randomness. Regarding the variant of the heuristic leveraging the linear relaxation of the problem, we have not only that the set of available columns is restricted, but also that the value of the objective does not account for the inevitable fluctuations in loss of quality due to branching during the standard form of resolution in GENCOL. In contrast, *seqAsg* provides an integer solution for the same CRP instance as the one given as input to GENCOL : it is then not surprising that it can capture some of the fluctuations related to the transformation from a relaxed solution into an integer solution. This makes it unlikely that even extensive efforts would lead to a configuration of the GENCOL parameters such that the heuristic produces solutions comparable to *seqAsg* in terms of predictions.

Based on these observations, it is possible to see that while a simple heuristic can to some extent quickly give an approximate idea of L^{sat}_{GENCOL} for a given CRP instance, such a heuristic lacks in precision and is less strong than a more elaborate approach such as the analysis of

solutions generated by an ML procedure like *seqAsg*. Another point of consideration is that such an ML procedure is more robust to changing instance conditions or even problem tasks : one can see how different CO tasks would be amenable to an approach that builds a solution fast and sequentially with the correct ML sequential assignment procedure. It is not clear, on the other hand, that a simple heuristic of the sort that we have shown would necessarily be available or reliable for these other tasks. These remarks point to the fact that the method we have introduced in this paper shows not only promise for an understudied problem (the CRP), but may also extend to other scheduling or more general OR tasks wherever it is most practical to implement. It is then possible to gain insight into final solutions obtained through standard resolution at a much discounted time.

4.7 Discussion

In this paper, we have presented an innovative way of gaining insight into CRP instances. After reviewing how ML and OR problems are interrelated, we have put forward an algorithm built upon a sequential assignment procedure, *seqAsg*, whose purpose is to generate quick solutions rather than solving the full CRP in a standard solver. We showed how this procedure relies on an RL policy defined by parameters updated through an evolutionary algorithm, CMA-ES. We have then shown, using seven data instances and a number of their variations, how the quality of the solutions generated by this procedure can be used to gain insight into the solutions produced by the GENCOL solver. By generating many variations of the data instances with gradually decreasing numbers of pilots, we showed the existence of a positive linear association between two types of loss of satisfaction. These were L^{sat}_{ML} and L^{sat}_{GENCOL}, in ML and solver-generated solutions, respectively. In both cases, the loss of satisfaction was measured against the sum of satisfactions for the best feasible schedules for pilots, S^{best}. We have then established that as L^{sat}_{ML} increases, the quality of the objective for a CRP instance handled by the solver quickly decreases while computing time dramatically increases. In that sense, the loss of satisfaction in the ML solutions generated by *seqAsg* directly translates into information as to the behavior of the GENCOL solver.

Next, we demonstrated how the relationship between ML and solver loss of satisfaction could be used in practice to help scheduling workers make adjustments to CRP instances as needed without prohibitive time requirements. The robustness of our method was also contrasted to other methods, such as handmade heuristics developed by scheduling workers and a heuristic yielding fast solutions in GENCOL itself. Indeed, we have seen that our method is generalizable to many different contexts and outperforms the insights that can be derived from simply obtaining solutions in GENCOL in a few seconds. This allowed us to

provide tangible recommendations for the use of the *seqAsg* procedure during the iterative construction of a CRP instance.

Overall, our contribution provides a concrete way of solving the CRP more efficiently, while using ML and RL techniques in a unique manner. Indeed, our procedure allows us to solve the CRP quickly, thus producing a fully feasible solution for suitable numbers of pilots while also allowing us to gain insight into the relevant instance. This allows for crew scheduling workers to reparameterize the problem at a low time cost, therefore obtaining a more sophisticated solution in the GENCOL solver when it is invoked at the end of the parameterization process.

Our method is also unique in that it provides very fast solutions, i.e., solutions obtained in an order of magnitude less time than other existing heuristics. While this involves some loss of ML solution quality, this loss is limited so that the solutions are strong enough to provide useful information almost immediately. This constitutes a unique contribution.

We conclude by mentioning that while this paper offers a way to improve CRP instances, there is still much to be gained by speeding up the resolution of the problem in an optimization solver like GENCOL. As *seqAsg* can also be used to determine feasible schedules for each pilot, this approach lends itself well to the dynamic constraint aggregation approach (DCA). Indeed, DCA procedures require clusters of pairings for each pilot, such as provided by an ML-generated solution, to launch. This solution can be generated by *seqAsg*. Thus, the solutions given by our ML-based procedure could not only build insight into the CRP instance, but also facilitate its resolution once the instance is deemed parameterized in a satisfactory way. Another possibility would be to exploit the information given by schedules with total satisfaction S^{best} in combination with the analysis tool provided by this paper. In this way, well-parameterized CRP instances can be readily identified, and valuable information regarding likely pairings and days off for each pilot can be provided to a learning algorithm. As seen in Section 4.6.2, this information is closely related to what is observed in GENCOL solutions for the CRP, potentially making it highly valuable.

Finally, it might be interesting to see under which conditions very fast solutions such as those provided by a procedure similar to *seqAsg* might be not only good enough to gain certain types of useful insights, but also to be used as final solutions in themselves. With the increasing popularity of GNNs that learn about an entire graph's topology, it might be possible to provide better features to quickly train algorithms of the sort presented in this paper in such a way as to bridge the gap in objective performance compared to other heuristics. These possibilities could be explored in future work.

4.8 Pseudocode for the feasibility heuristic

In this section, we detail further the feasibility heuristic presented in Section 4.5.2. We give the following pseudocode to describe the concrete steps taken to make one pilot's schedule feasible.

Pseudocode for the feasibility heuristic

```
pairings ← pairings.sortbylength()
short_pairings ← pairings.keep_shorts() // keeps pairings with few duties only
short_pairs ← short_pairings.make_combinations() // all pairs of pairings with few
duties
credits ← pairings.lengths.sum()

Function trim_schedule(pairings, rests) // removes pairings until credits do not exceed
the limit
for pairing in short_pairings do // if one short pairing is enough, remove it and make
updates
    if credits − pairing.length ≤ T^credit then
        pairings.remove(pairing)
        rests ← rests + pairing.duties and return
for pair in short_pairs do // if one pair of short pairings is enough, remove them and
make updates
    if credits − pair.length.sum() ≤ T^credit then
        pairings.remove(pair)
        rests ← rests + pair.duties.sum() and return
while credits > T^credit do // otherwise, remove pairings until below limit
    pairing ← pairings.next_long_enough()
    if pairing is not NULL then // remove first pairing that is long enough, if any
        pairings.remove(pairing)
        rests ← rests + pairing.duties
    else // if none, remove the last (and longest) pairing instead
        pairing ← pairings.last()
        pairings.remove(pairing)
        rests ← rests + pairing.duties
return

Function add_rest(pairings, rests) // removes pairings until there are enough days off
pairings ← pairings.sortbyduties()
while rests < T^off do
    pairing ← pairings.next()
    pairings.remove(pairing)
    rests ← rests + pairing.duties
return

if credits > T^credit then // call trim_schedule if too many credits
    pairings, rests ← trim_schedule(pairings, rests)
if rests < T^off then // call add_rests afterwards if still not enough days off
    pairings, rests ← add_rest(pairings, rests)
```

CHAPITRE 5 ARTICLE 2 : ACCELERATED WINDOWING FOR THE CREW ROSTERING PROBLEM WITH MACHINE LEARNING

Authors : Philippe Racette, Frédéric Quesnel, Andrea Lodi, and François Soumis

Submitted on February 27, 2025 in Computational Optimization and Applications

Abstract

The crew rostering problem (CRP) for pilots is a complex crew scheduling task assigning pairings, or sequences of flights starting and ending at the same airport, to pilots to create a monthly schedule. In this paper, we propose an innovative solution method for the CRP that uses a windowing approach. First, using a combination of machine learning (ML) and combinatorial optimisation (CO), we quickly generate an initial solution. The solution is obtained with a sequential assignment procedure (*seqAsg*) based on a neural network trained by an evolutionary algorithm. Then, this initial solution is reoptimized using a branch-and-price algorithm that relies on a windowing scheme to quickly obtain a CRP solution. This windowing method consists of decomposing the optimization horizon into several overlapping windows, and then optimizing each one sequentially. Although windowing has been successfully used in other airline applications, it had never been implemented for the CRP, due to its large number of horizontal constraints involving the whole planning horizon. We test our approach on two large real-world instances, and show that our method is over ten times faster than the state-of-the-art branch-and-price CRP solver GENCOL while providing solutions on average less than 1% away from optimality. We show that our windowing approach greatly benefits from being initialized with good-quality ML-based solutions. This is because the initial solution provides reliable information on the following windows, allowing the solver to better optimize the current one. For this reason, this approach outperforms other naive heuristics, including stand-alone ML or windowing.

5.1 Introduction

The crew rostering problem (CRP) is a form of crew assignment concerned with creating sets of schedules for pilots and flight attendants called rosters. Taking a set of pairings (i.e., sequences of flights and layovers starting and ending at the same airport) as input, the CRP forms schedules by assigning them to pilots over a predetermined time horizon. This is done while taking into consideration crew preferences regarding specific flights, days off and special requests made by crew members. A number of airline and collective agreement regulations

must also be observed.

Developing efficient methods to solve crew scheduling tasks in general is of prime importance to airline carriers, as problems such as the CRP take a long time to solve. One class of methods that has gained ground over the last few years is the use of machine learning (ML) techniques in the context of combinatorial optimization (CO) and operations research (OR) problems. In this paper, we build upon the previous work in Racette *et al.* [112] and develop a method that quickly generates good-quality CRP solutions. They propose a sequential assignment procedure named *seqAsg*, an ML-based approach that generates complete rosters to facilitate schedule planning. Although those rosters could be used to provide insight to planners, they were not of a good enough quality to be used in practice. We expand this contribution by showing that these rosters can be leveraged to obtain different near-optimal solutions produced faster than in the existing literature, and better than what would be obtained by different heuristics.

To do so, we propose a windowing approach that can take ML-generated rosters as input and improve them with a branch-and-price algorithm. Windowing is a solution method that decomposes the horizon into several overlapping time windows, and sequentially optimizes each window. Although windowing was applied to great success to the crew pairing problem (CPP) such as in Saddoune, Desaulniers and Soumis [34], it has not been attempted for the CRP. This is perhaps because the CPP has more localized horizontal constraints (each pairing lasting a few days). By contrast, the CRP has horizontal constraints spanning the whole horizon (schedule validity constraints). As the results of this paper show, windowing can cope with this difficulty well, and especially so if a good-enough initial solution is provided. This initial solution helps by providing an estimate of the horizontal contributions outside the current optimization window.

The benefits of windowing lie in the fact that by only enhancing part of a solution and freezing the rest, the size of the problem treated in each window decreases considerably. However, one possible drawback of windowing is the fact that the method is no longer optimal. The question is then to determine if any loss in the CRP objective is minimal enough to justify significant gains in computational time.

Given this context, we make the following contributions :

- We propose a windowing approach to the CRP for pilots and prove its effectiveness in obtaining rosters within 1% of optimality. Our method is over 10 times faster than the state of the art.
- We show that our windowing method greatly benefits from an initial solution, even if this solution is far from the optimum. Such a solution can be obtained using the

ML-based method proposed by Racette *et al.* [112].

We also provide a brief discussion of the future implications of this work. In particular, we mention the application of fast ML methods to CO problems, and how this research adds support to the use of ML to enhance solving methods able to exploit even limited additional information.

We now give an outline of the paper. In Section 5.2, we present a literature review of the topics relevant to this paper. In Section 5.3, we provide a detailed description of the CRP. In Section 5.4, we lay out the details of the windowing procedure. In Section 5.5, we present our experimental protocol and provide some implementation details. In Section 5.6, we present the main results supporting our contributions. We conclude in Section 5.7 with a discussion of the work presented in this paper and of some of its implications.

5.2 Literature review

In this section, we review the relevant literature for the work presented in this paper. We start with Section 5.2.1 by giving some context relevant to crew scheduling in general and the CRP more specifically. In Section 5.2.2, we provide more information about column generation, a technique often used to solve the CRP, and windowing. We describe the literature regarding solving OR problems with ML algorithms in Section 5.2.3. Finally, we provide the essential details related to the sequential assignment procedure we use in Section 5.2.4.

5.2.1 Crew scheduling and the CRP

Crew scheduling is a rich field that tackles many different problems. Gopalakrishnan and Johnson [13] describe the main tasks addressed by airline scheduling as a sequential process. The last two parts of the airline scheduling process are the CPP and the crew assignment problem, which includes crew bidding, crew rostering, and preferential bidding. As together these two steps make up crew scheduling, we describe them in more detail.

The CPP is concerned with building pairings, i.e., sequences of flights starting and ending at the same airport base, of minimal cost while serving all flights planned in the previous scheduling steps. Once pairings are made, they must be assigned to crew members, whether pilots or flight attendants, while also observing many additional, often complex regulations. Airlines have different ways to do so. One way is to have crew members bid on pre-made schedules, called bidlines, and to assign them while holding these bids in consideration. Another possibility is crew rostering, where an entire set of personalized schedules for the pilots is called a roster. In crew rostering, crew members state preferences over certain flights, days

off and sometimes other special activities (e.g., training for flying certain kinds of aircrafts), and the schedule is optimized for a certain criterion while ensuring that every pairing is carried out in a way that respects all rules. This is the approach considered in this paper. A hybrid approach, called preferential bidding, creates schedules much as in the CRP, but by also taking seniority into account in the objective. We note that each of these crew scheduling problems may use different formulations with one key difference lying in the criteria taken into account by the model objective. A review of these formulations is found in Kohl and Karisch [1].

5.2.2 Column generation and windowing

Column generation is a technique that is often used to solve complex CO problems. It decomposes the resolution process in two alternating phases : the restricted master problem (RMP) phase, where a model considering only a subset of possible schedules is solved, and the subproblem phase, where new promising columns are generated and added to the next RMP. Column generation is often embedded into a branch-and-bound tree where branching steps occur throughout the solving process. This ensures that an integer solution is found. Such algorithms are called "branch-and-price" algorithms and are described in Barnhart *et al.* [5] and Desrosiers *et al.* [6]. This method is now quite standard in commercial solvers.

While the branch-and-price framework presented above works well to solve many CO tasks in an efficient manner, there are some situations where we might want a faster solution than these methods can allow. Some examples include real-time optimization or planning which may have to be performed in a few seconds, or the reoptimization of a solution. One approach attempting to do this is windowing. With windowing, the planning horizon is broken into time windows of a few days that are treated separately. Windowing has been used in contexts as varied as flight and maintenance planning, the fishing industry, bus dispatching, the railway industry, and the CPP [34,35,37,39,44]. In particular, Saddoune, Desaulniers and Soumis [34] have found that the resolution of the CPP is accelerated by an average of 23.1% on 7 instances when compared to a sequential three-phase approach. However, windowing has not been attempted for the CRP, due to the presence of constraints extending for one month, or the entire horizon's length.

5.2.3 ML, metaheuristics and crew scheduling

ML methods have gained traction in recent years and have been increasingly applied to help solving OR problems, as have metaheuristics. Both also often serve as comparison points for each other. For an in-depth description, see Bengio, Lodi and Prouvost [9] and Scavuzzo *et*

al. [47].

With respect to crew scheduling, several contributions have been made. Yaakoubi, Soumis and Lacoste-Julien [32, 33] use supervised learning to make clusters of flights that are likely to belong to the same pairing to speed up the resolution of the CPP with dynamic constraint aggregation. Pereira *et al.* [72] use supervised learning to outperform strong branching during the resolution of the CPP while using properties of the branching tree as features, a type of approach often called “learning to branch". Quesnel *et al.* [63] consider the CRP and learn which pairings are most likely to be part of a crew's schedule, reducing the size of the problem and improving computing time by a factor of up to 10. Similarly, sometimes only promising columns, arcs, or flight connections, respectively, are considered as part of the column generation algorithm to enhance computing time [64, 71, 107]. Finally, Maenhout and Vanhoucke [84] solve the CRP with a scatter search heuristic for two-week time horizons, 100 pilots and 600 pairings and obtain solutions 1.67% away from optimality in an average time of 127 seconds. Their work stands out in the literature attempting to solve the CRP with metaheuristics as the instances used are particularly large, and the solutions obtained are of good quality.

5.2.4 Sequential assignment

In order to provide ML-generated solutions as input to the windowing implementation of our CRP solver, we use the sequential assignment procedure presented in Racette *et al.* [112]. The *seqAsg* procedure is a constructive heuristic that solves an assignment problem for each day of the CRP horizon (i.e., one month) in sequential order. Once this process is completed, a full roster, or set of schedules, is available. On each day, there is a set of pilots who are available to receive pairing assignments, single days off, or vacations. There is also a fixed set of pairings that must be assigned. Each assignment is given a utility (i.e., a measure of how desirable it is to make this specific assignment) intended to estimate the value of giving a pilot the activity specific to that assignment.

The utility function mentioned above must be learned. Racette *et al.* [112] do this using reinforcement learning or RL. They learn a policy that gives an approximation of the value of an assignment given certain conditions in the learning environment. This environment includes a reward function that accounts for satisfaction and feasibility. The ML weights for the neural network used to approximate the utility of an assignment are learned with the CMA-ES evolutionary algorithm [96].

5.3 Crew rostering problem

In this section, we describe the version of the CRP that we use in this paper. Since the data we present in Section 5.5.1 was originally found in Kasirzadeh, Saddoune and Soumis [20], the structure of the problem shown here follows essentially the same pattern. In particular, in this work, we consider the creation of personalized rosters, where pilots express preferences before scheduling starts and where these preferences are taken into account so as to maximize the overall crew satisfaction. Pilots declare weighted preferences, with a weight indicating the extent of the satisfaction given by its fulfilment. Each pilot receives a similar budget for the allocation of preference weights. Additionally, some pilots have prescheduled off period requests that must be honored.

Let W be a set of pairings, K a set of pilots, and D the set of possible bases. Each pilot and each pairing is associated with one base $d \in D$. The pairings are spread over an horizon of exactly one month. The CRP is then concerned with assigning all pairings to the pilots while ensuring each pilot has a complete schedule and in such a way as to maximize the sum of satisfaction scores for the whole set of pilots.

Schedules are constrained by various rules and regulations. We have the following rostering restrictions. Each pairing must be assigned to exactly one pilot and each pilot included must receive a feasible schedule. Pilots may not fly more than T^{flight} hours in a month, must not be assigned more than T^{work} consecutive duties, i.e., calendar days with at least one flight departure, and must receive at least T^{off} days off. All preassigned days off must be included, and, lastly, there must also be at least T^{min} hours of uninterrupted rest between two consecutive pairings.

Concerning the objective, the main criterion is pilots' satisfaction with their schedules, which we define as follows. Pilot preferences fall into two categories : preferred flights (as opposed to pairings) and preferred three-day vacations. Then, let us consider pilot $k \in K$. Let F_k be the set of preferred flights for pilot k and let $f \in F_k$ be such a flight with weight v_f^k. Likewise, let O_k be the set of preferred vacations for pilot k and let $o \in O_k$ be such a vacation with weight v_o^k. Let Ω and Ω^k be the set of feasible schedules, and feasible schedules for pilot $k \in K$, respectively. Let $p \in \Omega^k$ be such a schedule. Let a_{fp}^k be a constant with value 1 if flight f is in one of the pairings in schedule p assigned to pilot k, and 0 otherwise. Finally, let a_{op}^k be a constant with value 1 if preferred vacation o is in schedule p assigned to pilot k, and 0 otherwise. Then, we can define the satisfaction score brought to pilot k by schedule p, c_p^k, in the following way :

$$c_p^k = \sum_{f \in F^k} a_{fp}^k v_f^k + \sum_{o \in O_k} a_{op}^k v_o^k. \tag{5.1}$$

As it can be seen from Equation (5.1), a pilot's satisfaction with their schedule is the sum of all the weighted preferences granted to them. These scores are used in the objective formulation for the CRP model.

5.3.1 Mathematical formulation

In this section, we present the model we use to solve the CRP. Let Q^k be the set of preassigned days off for pilot $k \in K$. Let $n_w, w \in W$ be the number of flights in pairing $w \in W$. Let a_{wp} be a constant taking value 1 if pairing w is in schedule $p \in \Omega^k, k \in K$, and 0 otherwise. Let a_{qp} be a constant taking value 1 if preassigned day off $q \in Q^k$ is in schedule $p \in \Omega^k, k \in K$. Let c_p^k be the cost (satisfaction score) of schedule $p \in \Omega^k, k \in K$, as defined by Equation (5.1). Let x_p^k be a variable equal to 1 if schedule $p \in \Omega^k$ is assigned to pilot k, and 0 otherwise. Finally, let s_w and y_q be the slack variables ensuring the coverage of pairing $w \in W$ and preassigned day off $q \in Q^k, k \in K$, respectively.

Also, within this model, it is possible for certain pairings not to be assigned at the cost of a penalty. Let this penalty be C^F per unassigned flight in the pairing. In practice, such pairings would be assigned to reserve pilots, a situation judged undesirable due to the necessity of paying reserve pilots high wages for few hours worked. Additionally, to ensure feasibility, we allow not assigning prescheduled days off at the cost of a different penalty. Let its cost be C^D for each such day. C^D is set at a prohibitively high value so no preassigned days off are left unassigned unless absolutely necessary.

Then, the CRP reads as

$$\max \sum_{k \in K} \sum_{p \in \Omega^k} c_p^k x_p^k - C^F \sum_{w \in W} n_w s_w - C^D \sum_{k \in K} \sum_{q \in Q^k} y_q \tag{5.2}$$

subject to :

$$\sum_{k \in K} \sum_{p \in \Omega^k} a_{wp} x_p^k + s_w = 1, \quad \forall w \in W \tag{5.3}$$

$$\sum_{p \in \Omega^k} a_{qp} x_p^k + y_q = 1, \quad \forall k \in K, \forall q \in Q^k \tag{5.4}$$

$$\sum_{p \in \Omega^k} x_p^k = 1, \quad \forall k \in K \tag{5.5}$$

$$x_p^k \in \{0,1\}, \quad \forall k \in K, \forall p \in \Omega^k \tag{5.6}$$

$$s_w \in \{0,1\}, \quad \forall w \in W \tag{5.7}$$

$$y_q \in \{0,1\}, \quad \forall k \in K, \forall q \in Q^k \tag{5.8}$$

Objective function (5.2) maximizes the sum of three terms : one that takes into account the total satisfaction provided to the whole set of pilots ready to work for that roster, and two more terms that add penalties to the objective for unassigned pairings and missing preassigned days off, respectively. Constraints (5.3) ensure that each pairing is assigned. Constraints (5.4) ensure that each preassigned day off is assigned. Constraints (5.5) ensure that each pilot is given a schedule. Constraints (5.6) ensure that assignment variables are binary (i.e., an assignment is selected or not). Constraints (5.7) and (5.8) ensure that all slack variables are binary. We note that constraints related to the number of consecutive work days, total flight time, number of days off and postpairing rest are not included as they can all be assessed on an individual schedule basis. By only generating schedules that meet these requirements at each iteration of column generation, we do not need to state these constraints explicitly in the model.

Typically, the number of elements in Ω is too large for them to be enumerated explicitly, and in practice, there is no set of feasible schedules readily available. In order to solve the problem efficiently, one approach is to embed the mathematical formulation stated in this section in a branch-and-price scheme, where linear relaxations of model (5.2)-(5.8) are solved with column generation inside a branch-and-bound tree.

5.4 Methodology

In this section, we describe resolution by windowing in more depth. In Section 5.4.1, we describe the structure of the subproblem networks that are used to generate new columns during column generation. In Section 5.4.2, we show how these networks are adapted to implement a windowing procedure. For a detailed review of the different kinds of optimization strategies used in crew scheduling and many of their examples, see Desaulniers, Desrosiers and Solomon [113].

5.4.1 Subproblem networks

To implement windowing as described in Section 5.4.2, it is essential to understand the subproblem phase of column generation mentioned in Section 5.2.2. As a reminder, column generation is divided in two alternating phases. During the RMP phase, the relaxation of problem (5.2) - (5.8) is solved with a restricted set of feasible schedules, named Ω'. Next, the subproblem phase generates new schedules (i.e., columns) that are added to the restricted set of schedules considered when solving the RMP. For this to be possible, one needs a way of generating new feasible schedules and a method to check their reduced cost.

To compute the reduced cost of a candidate schedule, we use the dual solution of the RMP. Let us consider pilot $k \in K$. Let α_w denote the dual value for constraint (5.3) for pairing w. Let β_q denote the dual value for constraint (5.4) for pilot k's preassigned day off q. Let γ^k denote the dual value for constraint (5.5) for the scheduling requirement of pilot k.

The reduced cost of schedule p for pilot $k \in K$ is then

$$\overline{c}_p^k = c_p^k - \sum_{w \in W} \alpha_w a_{wp}^k - \sum_{q \in Q^k} \beta_q a_{qp}^k - \gamma^k. \tag{5.9}$$

Since Equation (5.9) is expressed as a sum of terms involving the total satisfaction provided to pilot $k \in K$ by schedule p, we model the subproblems as constrained shortest-path problems in an acyclic network. In each network, nodes represent different events such as the beginning or end of a pairing, and where arcs represent the selection of a certain action imposed to the pilot. Such a network is depicted in Figure 5.1. The costs placed on the arcs of the network are then chosen so that the sum of the arc costs along a path from the source to the sink, representing a schedule, is equivalent to the reduced cost of the entire schedule. Paths are further constrained by resource windows on each node, which are described below.

We now describe the structure of the network for pilot k's subproblem. The network includes five types of nodes. First, there is a *source* node $o(k)$ for the pilot at the start of their schedule. There is also a *sink* node $d(k)$ which marks the end of a schedule. Next, for each pairing in W, there is a *beginning-of-pairing* node and an *end-of-pairing* node. Finally, there is one *midnight* node per day. These mark the beginning of a new calendar day. Every day off starts and ends with such a node.

Let A^k be the arc set linking the nodes for in the subproblem of pilot $k \in K$. We note that we remove all arcs that make it possible to avoid assigning preassigned days off, although the associated penalties are still technically implemented in the solver used as a basic feature and remain part of objective (5.2). In that case, not granting pilot k's preassigned days off is equivalent to not assigning them a schedule. This is done with a static column for that purpose.

A^k includes the following types of arcs. First, there is a *roster start* arc which represents the start of the schedule. It links the source node for pilot k to the midnight node for the first day of the month. *Roster end* arcs mark the end of the schedule. There is a roster end arc between the end node of each pairing ending on the last day of the horizon and the sink. There is also a roster end arc between the last midnight node of the month and the sink.

Two types of arcs represent days off or three-day vacations. *Rest* arcs represent one single

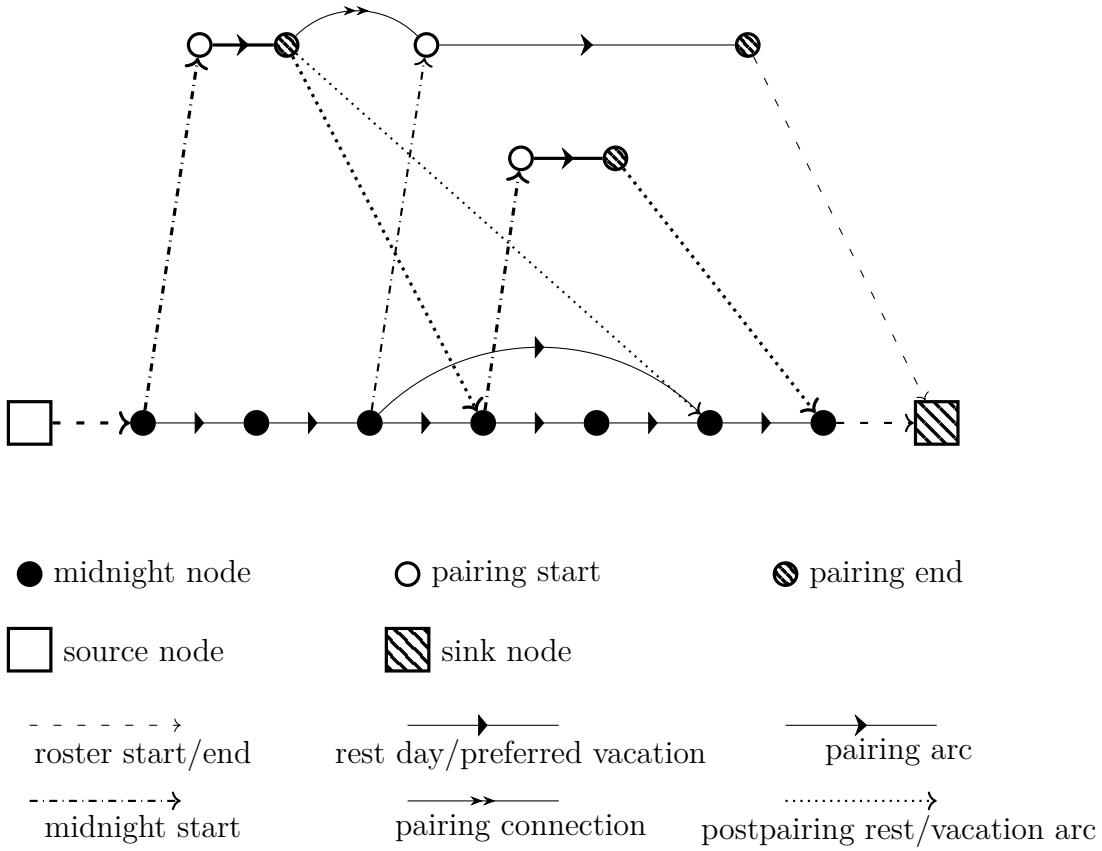


FIGURE 5.1 Example of subproblem network for a pilot

day off and link the midnight node marking the start of the day off to the midnight node marking the beginning of the next day. *Vacation arcs*, on the other end, are used to assign three-day vacations. These arc goes from the midnight node at the beginning of the first day of the vacation to the midnight node three calendar days later.

Pairing arcs represent carrying out pairings. The arc for a specific pairing links the pairing node at its beginning to the pairing node at its end. *Midnight start* arcs are used to assign a pairing after one or several days off or on the first day of the horizon. They link the midnight node at the start of the day to the beginning-of-pairing node of the first pairing assigned on that day. *Pairing connection* arcs ensure the direct transition between two pairings, one of which ends on a given day while the other starts on the same or following day. They go from the end-of-pairing node for the first pairing in the connection to the beginning-of-pairing node for the other pairing. Only those connection arcs that allow for sufficient postpairing rest time T^{min} between pairings are included.

Finally, some arcs are related to both pairing and rest periods. They ensure that time off is granted to the pilot following a pairing arrival. *Postpairing rest* arcs are used to assigned a

day off after a pairing arrival. The arc links the end-of-pairing node at the pairing arrival to the midnight node at the end of the next day. *Postpairing vacation* arcs allow the assignment of a three-day vacation after a pairing arrival. The arc links the end-of-pairing node at the pairing arrival to the midnight node at the end of the next three days.

Arc costs depend on the arc type. Let i and j be two nodes linked by an arc in A^k and let c_{ij}^k be its cost. Rest and postpairing rest arcs may have a positive cost depending on the pilot's preferences, while all other costs are zero. The schedule associated with selecting arcs forming a full path in the network has a cost equal to the sum of the arc costs, which is equal to costs (5.1). Then, using the dual variables defined above, we set reduced arc costs as follows and where i and j here refer to either the node identifier or the corresponding activity depending on context :

$$\bar{c}_{ij}^k = \begin{cases} c_{ij}^k - \alpha_i & \text{for pairing arcs,} \\ c_{ij}^k - \beta_i & \text{for rest arcs that ensure respecting a preassigned day,} \\ c_{ij}^k - \gamma^k & \text{for roster start arcs, i.e., } i = o(k), \\ c_{ij}^k & \text{otherwise.} \end{cases}$$

Taking the sum of all reduced costs for the arcs in a schedule gives the same result as Equation (5.9).

In addition, there are resource windows on each node. Resources are values that are consumed when traversing arcs. Resource windows require that at each node, each resource fall within a given range. These ranges ensure that the schedules generated are feasible. Moreover, the lower bounds on the resource windows are soft : it is possible to extend a schedule even if it violates the lower bound of the arrival node. In this case, the corresponding resource values are set to the lower bound.

We define the resources as follows. The *time* resource is the chronological time, in minutes, elapsed since the beginning of the month. Each assigned arc increases this resource by the amount of time needed to perform the activity encoded by the arc. The *days off* resource is the number of days off left to grant to the pilot to meet the monthly minimum for time off. It is initialized at T^{off}, and rest, vacation, postpairing rest and postpairing vacation arcs decrease its value by the number of days off taken. To ensure that pilot k receives enough days off, the resource window on the sink node is $[0, 0]$ days off left. The *flight time* resource is the flight time cumulated by the pilot so far, in hours. Pairing arcs increase the value of this resource by the number of hours of work included in the flights in the pairing and the briefing and debriefing at the start and end of the pairing. This resource is initialized

at 0 at the source node and must respect a resource window of $[0, T^{flight}]$ at each node in the network. The last resource is the *number of consecutive days worked* (i.e., consecutive days where the pilot serves at least one departing flight). It is initialized at 0, and its value increases when selecting pairing arcs by the number of days in the pairing where at least one flight departs, while rest arcs, including postpairing rest arcs, reset it to zero. Other arc types have no effect on this resource, and its range is $[0, T^{work}]$ consecutive working days at every node.

When no new columns are generated by the subproblems, we perform a branching step. However, we note that practical implementations of the CRP do not always solve each linear relaxation to optimality, but rather set a heuristic stopping criterion. In this paper, we require that the objective improve by a percentage of at least m_{iter} over the last N_{iter} iterations (i.e., we impose a common stopping criterion for the CRP). When this is not the case, a branching step takes place, whether the linear relaxation is solved to optimality using this criterion or not. We use two branching methods : column fixing and intertask fixing.

Column fixing sets a schedule assignment variable to 1 permanently. Intertask fixing imposes or forbids that two activities appear consecutively in a solution. Schedules that do not respect this condition are not included in the next RMP. In each case, the branching we do is heuristic with a depth-first search that continues until an integer solution is found (i.e., there is no backtracking or branching to infeasibility). We start with column fixing of one to three variables whose fractional value is above a certain threshold (*CfixSelectThreshold*). Intertask decisions are scored next based on fractional flow, which is then standardized based on the scores for all such possible decisions. One to three intertask decisions with a score above a minimum threshold (*ItimposeSelectThreshold*) are then selected.

5.4.2 Windowing method

We now describe the main approach used to leverage ML in this paper : windowing. When applying windowing, the horizon is broken into several overlapping windows. For example, if windows are taken to be 10 days in length with an overlap of three days, and the month chosen has 31 days, we would have that the windows range from

— day 1 - day 10,
— day 8 - day 17,
— day 15 - day 24, and
— day 22 - day 31.

Once these windows have been defined, windowing can proceed in two ways. If no initial solution has been provided as input to the windowing procedure, we build a solution from

scratch. The solution is built one window at a time and in sequence while holding fixed the elements found in previous windows. We call this basic windowing, *win-basic* for short as used in Section 5.5.2. Basic windowing is the approach that has been used so far in the literature, such as in the papers cited in Section 5.2.2.

In the case where an initial solution has been provided as input, the whole solution is reoptimized one window at a time. Windowing improves each part of the roster successively and in sequence, while the elements included in the other windows and outside the overlap period remain fixed. We call this windowing with ML, *win-ML* for short as used in Section 5.5.2. In order to implement both forms of windowing (i.e., *win-basic* and *win-ML*), we solve the CRP with the same branch-and-price scheme that would otherwise be standard, with the difference that the subproblem network for each pilot is modified to impose that the content of some windows remain fixed.

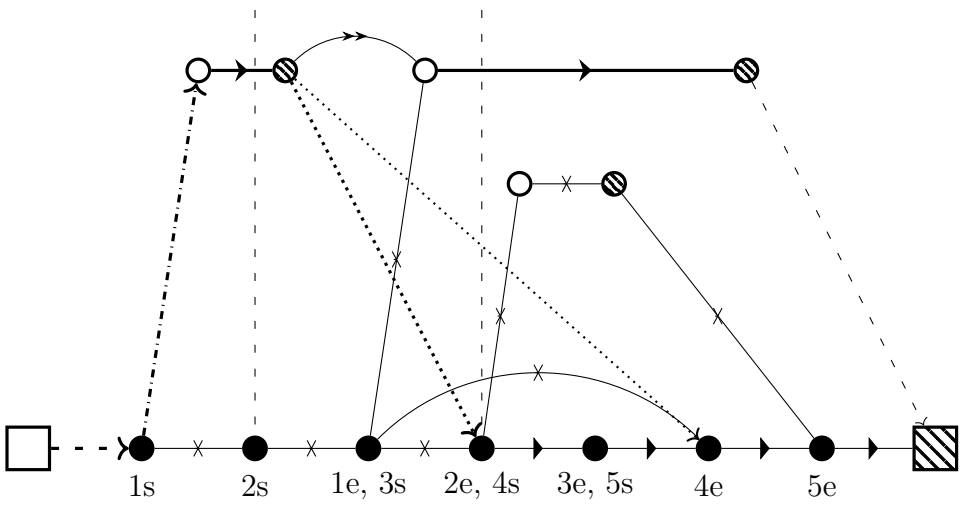
With *win-basic* (no initial solution), no pairing can be assigned prematurely in future windows. The subproblem networks for the pilots are therefore changed to account for this, and the arcs that would allow assigning pairings starting in future windows, unlike rest arcs, are removed. Furthermore, pairings assigned in previous windowing steps must be selected again (unless the departure occurs in the overlapping period between the previous and current windows), and arcs that are only in paths allowing to bypass this restriction are excluded as well.

With *win-ML* (an initial solution is provided), pairings assigned both in a previous windowing step (with the exception of overlap as above) or future windows of the initial solution, must be respected. For this reason, in each subproblem network, arcs that are in conflict with the pairings in the fixed windows are removed from the network, forcing the solver to select pairings departing in them. We also note that for both types of windowing, nodes that are not part of any allowed path are removed.

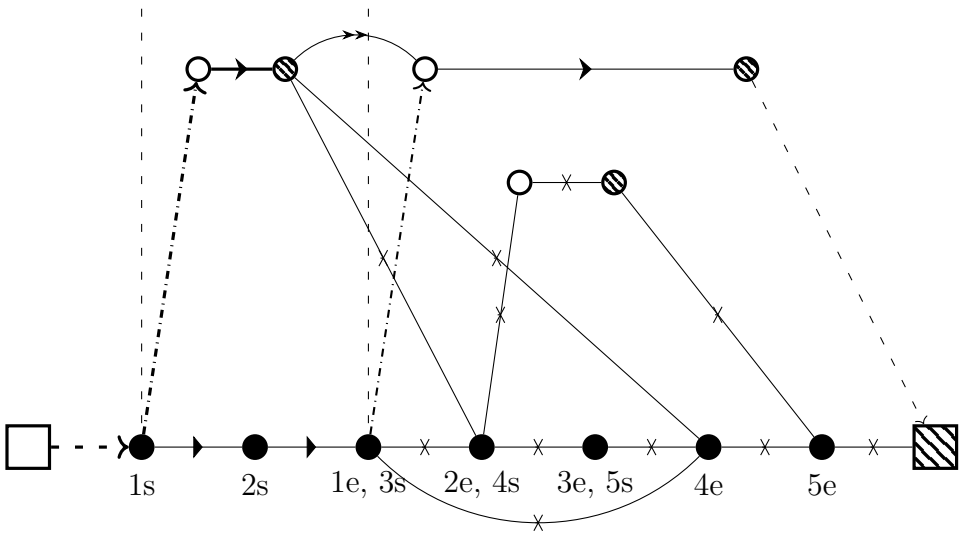
Figure 5.2 illustrates the windowing method for windows with a length of 2 days and an overlap of 1 day. It represents two modified subproblem networks based on Figure 5.1, one for each form of windowing. A window start or end is represented by its number followed by the letter **s** or **e**, respectively (e.g., **2s** and **2e** for the second window). In this example, we focus on the second window for windowing without an initial solution (Figure 5.2a), and on the first window for windowing with an initial solution (Figure 5.2b).

In the case without an initial solution (Figure 5.2a), we consider the first pairing of the month. The network excludes arcs that conflict with this pairing, as that pairing was assigned at the previous windowing step. We allow to choose whether to assign the pairing starting on the third day, as this falls within the second window (i.e., the one under treatment). The network

(a) windowing without initial solution (current window : 2)



(b) windowing with initial solution (current window : 1)



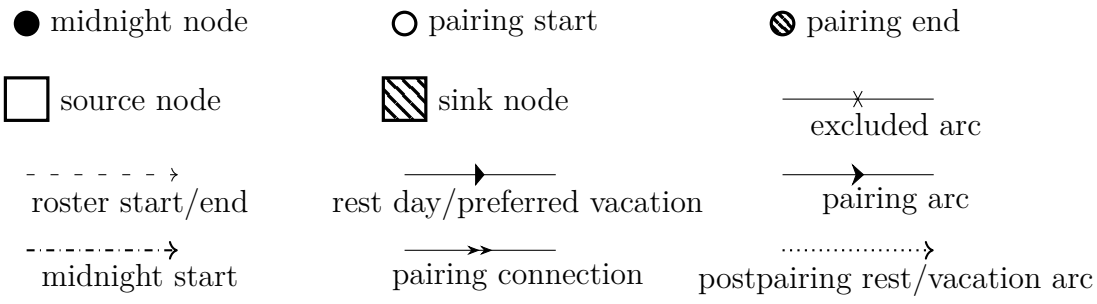


FIGURE 5.2 Pilot subproblem networks with frozen subpaths

excludes arcs allowing the pairing starting on the fourth day, as later pairings can only be assigned when the window of their departure is reached.

In the case with an initial solution (Figure 5.2b), we consider that the arcs included in the network must allow to choose whether the pairing starting on the first day is assigned, even though it was part of the ML solution, as the first window is currently treated. In the case where we choose to assign the pairing again, the networks will need to take this into account when treating future windows by imposing its selection, just as in Figure 5.2a ; in the opposite case, it will be forbidden instead. We impose the pairing starting on the third day, as it is part of the initial solution and does not start in the window treated. The pairing starting on the fourth day is not allowed, as it conflicts with the fixed pairing starting on the third day. We exclude arcs that can only be part of paths that do not select the pairing starting on the third day. Overall, the networks in Figures 5.2a and 5.2b are in accordance with the description of the network modifications given above : however, they exclude the pairing starting on the fourth day for different reasons. Using modified networks for each window successively leads to a full solution.

A general feature of windowing is that once all appropriate arcs have been removed in a network, its size is significantly reduced. For example, for the instance with the largest networks in our data, the number of remaining arcs for a window length of 10 days with an overlap of 3 days varies between 66462 and 86661 depending on the window and the presence of an initial solution, by comparison with 263377 for an unchanged network. Since the computational complexity of the subproblems increases approximately in proportion with the square of the number of arcs, these smaller networks lead to an accelerated resolution while applying the already heuristic branch-and-price algorithm. Furthermore, the maximum depth reached in the branching trees is also reduced (from 322 levels to at most 63 for the example given).

We close this section with a word on the challenges presented by windowing. As mentioned in Section 5.2, windowing has been used for a range of applications. However, it has been assumed until now that windowing might not suit the CRP as, by contrast with other problems such as the CPP where windowing approaches have been used successfully, the CRP includes many global constraints that extend for the entire time horizon of one month. In other words, these constraints overlap with all windows, rather than just one or two as when dealing with pairings. It is therefore necessary to implement windowing to see how well a state-of-the-art CRP solver can handle this additional complexity. Doing so also allows testing the possible benefits of an interaction between windowing and ML for the CRP : ideally, a good ML-generated roster helps in managing the added complexity of this problem better than basic windowing. We could imagine that fixing good pairings in later windows prevents

assigning conflicting pairings in the current window. Another possibility is that without an initial solution, days off are not taken into account while solving the first window, as only days off are assigned in the other ones. This can lead to sacrificing pilot preferences later on to ensure that the corresponding resource constraints are met in the end.

5.5 Experimental considerations

In this section, we describe the specific data sets we use for our experiments, as well as the conditions in which these experiments are carried out. We start by describing the data, and continue with a description of some solving parameters and performance indicators.

5.5.1 Data

We now present the data used for the experiments conducted in this paper, which seek to determine if the interaction of ML and windowing has the potential to improve the resolution of the CRP. The data set we used is directly taken from the material where it was originally published [20]. The original data set is made of seven instances (labeled I1 to I7), each with a different fleet size. Each fleet's instance is inspired by real data provided by a major airline carrier. We define an instance as a set of flights, a set of pilots, and a pairing solution for the CPP given the same set of flights. Furthermore, each pilot is attached to exactly one base in D, with D of size 3. The CPP solutions are for each instance obtained using a state-of-the art CPP solver : see Quesnel, Desaulniers and Soumis [114].

In this work, we focus on the two largest instances by number of pairings (I5 and I7). We provide the detailed information concerning the structure of each instance in Table 5.1. The first five columns of the table give the instance and number of pilots, airports, flights, and pairings in the instance, respectively. The last two columns represent the percentage of the pairings with length ≥ 4 days (i.e., long pairings) for the base with the most long pairings, and the total flight time required for all the pairings in the roster, respectively.

Tableau 5.1 Description of the instances available

Instance	Pilots	Airports	Flights	Pairings	Long pairings (%)	Flight time (min)
I5	239	34	5743	1222	64.5	1037333
I7	322	54	7765	1473	57.0	1289546

Table 5.1 shows that the percentage of long pairings is high for these two instances, which is a factor that increases the complexity of the resolution. Indeed, shorter pairings allow for

a greater variety of pairing arrangements and create fewer potential conflicts with pairing patterns that emulate an optimal roster.

A key parameter of the CRP is the average flight time per pilot. Racette *et al.* [112] show that this value correlates with the CRP's difficulty. In other words, instances where average flight time per pilot is higher are more constraining and harder to solve. To test our method under various difficulty conditions, we create new instances by changing the number of pilots. For both I5 and I7, the number of pilots available is chosen so that the average flight time per pilot required for the month varies between 50 and 75 hours (to the nearest hour), by increments of 5 hours.

Moreover, preferences, vacations and preassigned days off are not a priori part of the an instance, and must be created. Several scenarios are created for each new instance. A scenario consists of a new instance plus a set of preferences, i.e., preferred vacations and preferred days off. We generate 30 scenarios for each new instance, of which 25 are used for training and the remaining 5 for testing of the ML model used.

5.5.2 Test parameters and indicators

In our tests, we consider five different solution methods. The first is a standard branch-and-price method (without windowing). It was implemented using GENCOL 4.5, i.e., commercial software that is regularly used in industry to solve complex crew scheduling tasks, including the CRP. We call this method *alg-basic*, and we refer to it as our benchmark for all other methods in terms of speed and solution quality.

The next method is the *seqAsg* procedure by itself. In this case, the roster produced by sequential assignment is kept as a final solution. The remaining three methods are windowing starting with an ML solution (*win-ML*), basic windowing (*win-basic*), and a fast GENCOL heuristic with modified stopping criteria (*alg-fast*). *win-ML* uses the initial solution provided by *seqAsg* and serves to determine if ML has the potential to enhance basic windowing. *win-basic*, on the other hand, serves a benchmark for performance with the use of windowing.

The fast heuristic (*alg-fast*) modifies *alg-basic* by changing the stopping criteria N_{iter} and m_{iter}, which set the number of rolling iterations where a minimum improvement of the objective is required. Resource dominance is also changed so dominance goes down from three resources to one resource only. We did not notice any significant difference in performance based on the choice of the resource kept, although the number of resources considered for dominance does have an impact on computational speed.

We now give the details involved in getting ML-generated solutions. ML training is done

using a neural network approximator with two hidden layers. These layers have 4 neurons each. With 13 neurons on the input layer and 1 only as output, we get a total of of 77 ML weights when including bias terms as well. This is smaller than is typically the case for a neural network, as the CMA-ES algorithm generally works better for up to 100 learning parameters [97]. The activation function selected is the rectified linear unit (ReLU), where $f : \mathbb{R} \to \mathbb{R}$, $f(x) = \max(0, x)$. Early stopping is used after 120 iterations to prevent overfitting of the data. Furthermore, the *seqAsg* procedure is implemented in Python version 3.7.1. The assignment problems are implemented using the *pulp* library and solved using the CBC solver.

Table 5.2 shows the value for different problem parameters, including the upper bounds for each resource, the penalty weights in objective (5.2), and heuristic branching parameters. The last two parameters in the table refer to running GENCOL using a fast heuristic with stricter stopping criteria before branching. The required improvement and maximum number of iterations to reach it have been changed by factors of 20 and 125, respectively when compared to alg-basic. We also limit resource dominance to one resource instead of three.

Regarding windowing, we set the window length to 10 days, and the overlap to 3 days. These values were taken as a compromise. Indeed, shorter windows tend to lead to objectives of a worse quality, whereas larger windows take longer to solve.

Let us now define the key performance indicators that help measuring the performance of each method. We define S^* and t^* as the mean CRP solution value and solving time when using a given method $*$. These indicators taken with the *alg-basic* method serve as a baseline for other solution approaches and the five test scenarios used. We also note that the values above include penalties for any unassigned pairings, and that t^* includes the time needed by *seqAsg* to generate ML solutions, if applicable.

We also define :

— $L^* = 100 * \left(1 - \frac{S^*}{S^{alg-basic}}\right)$ is the loss of satisfaction for solutions obtained with a given method $*$ when compared to *alg-basic*.

— $p = 100 * \frac{t^*}{t^{alg-basic}}$ is the proportion of the solving time taken by a given method $*$ when compared to *alg-basic*.

These performance indicators allow to compare and contrast the use of windowing with both a standard branch-and-price algorithm and two accelerated resolution methods (the fast GENCOL heuristic and stand-alone sequential assignment). The differentiation between *win-ML* and *win-basic* helps clarify if any effect observed when using windowing can be improved by ML, i.e., if the nature of the input given has an impact.

Tableau 5.2 Solving constants by category

Constant	Value	Description
Resource bounds		
T^{work}	6	Max. number of consecutive duties
T^{off}	10	Min. number of days off
T^{min}	12	Min. postpairing rest time (hours)
T^{flight}	85	Max. flight time (hours) per pilot
Penalties		
C^F	100	Penalty for unassigned flight
C^D	1,000,000	Penalty for preassigned days off
Resolution		
N_{iter}	500	Number of iter. for min. improvement
m_{iter} (%)	0.05	Min. improvement over N_{iter}
CfixSelectThreshold	0.70	Column fixing threshold
ItimposeSelectThreshold	0.70	Intertask fixing threshold
Fast heuristic		
N_{iter}	4	Number of iter. for min. improvement
m_{iter} (%)	1.00	Min. improvement over N_{iter}

5.6 Computational results

In this section, we present results for the various solution methods outlined in this paper. In Section 5.6.1, we first compare the quality and time needed for solutions obtained with windowing to those produced by a standard branch-and-price algorithm. In Section 5.6.2, we compare the windowing method with (*win-ML*) and without (*win-basic*) an initial solution. Finally, in Section 5.6.3, we study the impact of changing the length of the windows on the algorithm's performance.

5.6.1 Impact of windowing

In this section, we compare the speed and objective value for solutions obtained with windowing alone (*win-basic*) to those obtained by the standard branch-and-price algorithm (*alg-basic*) and the accelerated GENCOL heuristic *alg-fast*.

We report these results in Table 5.3. Each row represents a different instance with a given level of average total flight time per pilot, rounded to the nearest hour. For each instance, the results are categorized based on the method that was used to obtain them, i.e., *alg-basic*, *alg-fast*, or *win-basic*.For each method, we report the average loss (L) as well as the average

computing time, in seconds (t). For *alg-fast* and *win-basic*, we also report the computing time relative to *alg-basic* (p).

TABLEAU 5.3 Quality and solving time of solutions obtained with *win-basic* compared to *alg-basic* and *alg-fast*

Data	Pilots	alg-basic		alg-fast				win-basic			
		S	t (s)	S	t (s)	L (%)	p (%)	S	t (s)	L (%)	p (%)
I5-50	346	253757	391	250136	223	1.43	57.0	252777	27.7	0.39	7.1
I5-55	315	231875	405	229165	225	1.17	55.6	230972	30.7	0.39	7.6
I5-60	288	210863	424	208176	223	1.27	52.6	209643	31.1	0.58	7.3
I5-65	266	195573	549	192070	246	1.79	44.8	193791	35.4	0.91	6.4
I5-70	247	181200	691	176674	244	2.50	35.3	178209	39.3	1.65	5.7
I5-75	230	167501	769	163200	243	2.27	31.6	163162	44.0	2.59	5.7
I7-50	429	316174	633	311408	290	1.51	45.8	314419	35.1	0.51	5.5
I7-55	391	288098	680	284343	278	1.30	40.9	286515	44.6	0.55	6.6
I7-60	358	264681	790	259883	263	1.81	33.3	261668	50.8	1.14	6.4
I7-65	331	245747	898	240191	229	2.26	25.5	241898	52.6	1.57	5.9
I7-70	307	225507	1228	218704	227	3.02	18.5	220418	64.5	2.26	5.3
I7-75	287	207255	1477	200403	220	3.31	14.9	199769	69.7	3.61	4.7
Average	**316**	**232353**	**745**	**227863**	**243**	**1.97**	**38.0**	**229437**	**43.8**	**1.35**	**6.2**

From Table 5.3, we observe that *win-basic* exhibits computing times sped up on average by a factor greater than 10 with respect to *alg-basic*. In fact, in many cases, we have that $p^{win-basic}$ is closer to 5% (e.g., for I7 and $W = 75$). Furthermore, the values of $p^{win-basic}$ show that the solving time grows more slowly when using windowing than for standard resolution when the CRP instances become progressively more difficult. For example, in all cases, we have that the reported value for $p^{win-basic}$ is lower when $W \in \{70, 75\}$ than when $W \leq 65$.

Method *win-basic* consistently performs faster than *alg-fast*, with a better average objective value. This is significant, as since this heuristic was configured to find a solution as quickly as possible, we have evidence that naive heuristics cannot outperform windowing in this context.

5.6.2 Impact of ML solutions

In this section, we test the impact of providing the windowing procedure with ML input (*win-ML*). We compare *win-ML* with basic windowing (*win-basic*) and the use of ML solutions alone (*seqAsg*).

We show the results for these comparisons in Table 5.4.

TABLEAU 5.4 Quality and solving time of solutions obtained with *win-ML* compared to *win-basic* and *alg-ML*

Data	Pilots	seqAsg		win-basic				win-ML			
		S	L (%)	S	t (s)	L (%)	p (%)	S	t (s)	L (%)	p (%)
I5-50	346	236462	6.82	252777	27.7	0.39	7.1	252823	40.2	0.37	10.3
I5-55	315	213577	7.89	230972	30.7	0.39	7.6	231105	42.6	0.33	10.5
I5-60	288	193213	8.37	209643	31.1	0.58	7.3	210014	44.8	0.40	10.6
I5-65	266	185979	4.91	193791	35.4	0.91	6.4	194658	48.5	0.47	8.8
I5-70	247	161701	10.76	178209	39.3	1.65	5.7	178740	50.7	1.35	7.3
I5-75	230	153856	8.15	163162	44.0	2.59	5.7	164796	53.7	1.61	7.0
I7-50	429	307825	2.64	314419	35.1	0.51	5.5	315346	53.9	0.26	8.5
I7-55	391	275503	4.37	286515	44.6	0.55	6.6	287093	62.6	0.35	9.2
I7-60	358	252426	4.63	261668	50.8	1.14	6.4	263156	64.0	0.58	8.1
I7-65	331	228589	6.98	241898	52.6	1.57	5.9	243622	68.5	0.86	7.6
I7-70	307	205510	8.87	220418	64.5	2.26	5.3	221864	71.3	1.62	5.8
I7-75	287	182271	12.05	199769	69.7	3.61	4.7	201816	85.4	2.62	5.8
Average	**316**	**216409**	**7.20**	**229437**	**43.8**	**1.35**	**6.2**	**230419**	**57.2**	**0.90**	**8.3**

From the results presented in Table 5.4, a number of facts stand out. First, both windowing methods provide much improved objective values when compared to *seqAsg*. In particular, we see that the average values of L^{seqAsg} and L^{win-ML} are quite different, with a drop from 7.20% to 0.90%, or a recovery of 87.5% of the objective loss with respect to $S^{win-basic}$. Next, of primary importance, we can observe that for all instances, L^{win-ML} is lower than $L^{win-basic}$, making *win-ML* better performing than *win-basic*. In other words, we provide a new way of generating accelerated solutions using both ML and windowing that beats windowing alone.

Furthermore, the enhanced quality of the *win-ML* solutions is more marked as the total flight time required from pilots increases. For example, for instance I5 and $W = 50$, the value of L^{win-ML} is 0.37% for the improved ML solution while $L^{win-basic}$ is 0.39 %. For $W = 75$, these values are 1.61% and 2.59%, respectively. Given that values of W near 75 hours flown per month are similar to industry standards and typical requirements placed on pilots, this means our method is particularly efficient in situations presenting a realistic challenge.

The average results for *win-ML* showed in Table 5.4 also confirm the findings discussed so far. We see that $L^{win-basic}$ is higher than L^{win-ML} : the loss of satisfaction with respect to $S^{alg-basic}$ drops from 1.35% to 0.90% (a recovery of 33.3% of the loss observed with basic windowing). Given that the linear relaxation found by *alg-basic* is in every case fully solved due to stringent stopping criteria ($N_{iter} = 500$) and that the integrality gap of *win-ML* solutions is under 0.10%, *win-ML* solutions are on average less than 1% away from optimal

solutions : if this threshold is judged acceptable, then the solutions can be used as final products. These findings are especially interesting in light of the fact that, for each window, the CRP is solved from scratch. This means that the information contained in a window cannot be used during the CRP run aiming to enhance it. This leads to an important question related to the factors that lead to ML-generated solutions performing better than solutions obtained through other means. In particular, good assignments within a window are found independently again after solving the problem for that window : they do not simply remain in place, while the solver leverages them. One hypothesis to explain the observed gains in performance is that since all but the current window remain frozen, the resolution of the CRP for the current window cannot cause the assignment of activities that are incompatible with good activities in the fixed parts of the roster.

Finally, we also see that while *win-ML* takes slightly longer to run than *win-basic*, mostly due to the requirement of generating an ML solution with *seqAsg*, the average running time is still less than 10% of what is needed in GENCOL with *alg-basic*, as shown by the average value of p^{win-ML} for this method.

5.6.3 Impact of window length

An important point of consideration is the influence of window length. Short windows generally lead to a worse objective, and given that 10-day windows are already fast, there is little interest in gaining a few seconds of computational time at the cost of a worsened value for the objective function. However, longer windows may improve the objective, and it is for this reason interesting to see if they can do so with a reasonable acceleration with regard to *alg-basic.*

In Table 5.5, we use window lengths of 15 days with an overlap of 7 days, for a total of three windows over the entire horizon. We see how *alg-basic* and *win-ML* compare both for 10-day and 15-day windows. The rows and columns included have the same meaning as for Tables 5.3 and 5.4. We only keep here instances where $W \geq 65$ to test our method on the most complicated cases (i.e., those with a value of $L^{win-basic}$ near or above 1%). In doing so, we test whether using a longer window length is advisable.

Based on the data in Table 5.5, it seems that longer time windows yield on average a better objective quality when using *win-ML*, as L^{win-ML} goes down from 1.42% to 0.34%. The value of p^{win-ML}, however, increases on average from 7.1 % to 18.1% of the time needed by *alg-basic*. Which window length is more desirable is then a matter of the specific needs of the airline : it may be that the need to re-solve the CRP is pressing enough that faster resolution is preferred, or that satisfaction takes precedence. The window length can then be adjusted

accordingly.

TABLEAU 5.5 Quality of solutions obtained through windowing with ML : 10 and 15-day windows

Data	alg-basic		win-ML (10 days)				win-ML (15 days)			
	S	t (s)	S	t (s)	L (%)	p (%)	S	t (s)	L (%)	p (%)
I5-65	195573	549	194658	48.5	0.47	8.8	195350	116	0.11	21.1
I5-70	181200	691	178740	50.7	1.35	7.3	180404	126	0.44	18.2
I5-75	167501	769	164796	53.7	1.61	7.0	166790	130	0.43	16.9
I7-65	245747	898	243622	68.5	0.86	7.6	245428	188	0.13	21.0
I7-70	225507	1228	221864	71.3	1.62	5.8	224952	197	0.25	16.0
I7-75	207255	1477	201816	85.4	2.62	5.8	205906	230	0.65	15.5
Average	**203797**	**935**	**200916**	**63.0**	**1.42**	**7.1**	**203138**	**165**	**0.34**	**18.1**

5.7 Discussion

In this paper, we have presented a new method to obtain accelerated, high-quality solutions to the CRP. We have considered two large instances adapted from real-life data obtained from a major commercial carrier. We have considered different variations based on these instances to get a broader range of monthly flying times per pilot representative of the industry.

We used the *seqAsg* procedure presented in Racette *et al.* [112], a sequential assignment procedure that creates ML-generated solutions for the CRP. Once ML solutions were obtained, we improved them using windowing, a technique that solves the CRP again by considering certains parts of a solution and freezing the rest. To do so, we have created the first implementation of this approach for the CRP. Based on the results presented, we were able to produce high-quality solutions in under 10% of the time that a state-of-the-art solver, GENCOL version 4.5, would otherwise have taken. The loss of objective quality associated with these rosters when compared to optimality was under 1% on average for the cases considered ; with longer windows, we obtained better objectives, but at an increased time cost.

By doing so, we have provided two original contributions. First, we showed that windowing can be successfully implemented for the CRP. Second, we generated high-quality solutions for the CRP in less than one minute on realistic instances. This is new, as while many heuristics, ML techniques and other algorithms have improved the resolution of this problem, the extent to which we have accelerated the process with the *win-ML* method is unprecedented. This of course came at the cost of a slightly higher loss of pilot satisfaction, but our method compared

favorably with three other fast heuristics : ML alone (*seqAsg*), basic windowing without an initial solution (*win-basic*), and accelerated branching in GENCOL, which suggests that in cases where speed is critical, an approach merging ML and windowing outperforms many naive methods in terms of the quality of the rosters. The solutions obtained can also be used as final products when an average gap of under 1% is acceptable with respect to optimality.

Finally, with the work presented in this paper, we have given additional evidence that simple ML methods can be rich in possibilities to treat difficult scheduling tasks such as the CRP. In fact, it is noteworthy that such results could be obtained with our method : a well-established evolutionary algorithm (CMA-ES) with a small neural network. We anticipate that future research in this area will help clarify the potential of different ML/OR interactions on a variety of tasks.

CHAPITRE 6 EFFICACITÉ DE L'APPLICATION DE MÉTHODES ML AU PROBLÈME DES BLOCS MENSUELS PERSONNALISÉS AVEC DCA

Résumé

Les avancées récentes dans le domaine de l'apprentissage machine (ML) ont mené à un intérêt croissant envers l'application de techniques ML aux problèmes d'optimisation combinatoire et au domaine de la recherche opérationnelle (RO) en général. Une question encore sans réponse est le fait de savoir dans quelles conditions les applications ML siéent le mieux à cette fin. Dans ce chapitre, nous abordons un problème complexe de création d'horaires d'équipage, soit la conception de blocs mensuels personnalisés (CRP). Avec l'aide d'une procédure d'affectation successive (*seqAsg*) utilisant les poids issus de différentes heuristiques ML, nous générons des solutions plus rapidement que pour la littérature actuelle. Nous évaluons la capacité de ces solutions à agir comme entrée pour les algorithmes de la famille de l'agrégation dynamique de contrainte (DCA) au niveau nécessaire pour en tirer les pleins avantages. Nous contrastons les résultats obtenus avec des applications précédemment réalisées pour la planification et la résolution accélérée du CRP. Il ressort de cette analyse que, en accord avec la littérature déjà existante, l'interaction ML/RO a un meilleur rendement lorsque ce qui est appris est d'ordre plus général et moins axé sur la réplication de parties spécifiques d'une solution optimale.

6.1 Introduction

La création d'horaires d'équipage fait partie intégrante des opérations des grandes compagnies aériennes. Il s'agit de la deuxième source de dépenses pour ces compagnies, derrière les coûts de carburant. Elle a aussi un impact décisif sur la qualité de vie des pilotes et des agents de bord, puisque la création d'horaires tient habituellement à prendre en compte les préférences formulées par l'équipage. Les problèmes de création d'horaires d'équipage, particulièrement pour ce qui est des modèles d'optimisation réalistes et propres à l'industrie, impliquent normalement un nombre fini, mais intraitable de solutions possibles, faisant ainsi d'eux une famille de tâches relevant du domaine de l'optimisation combinatoire.

Au cours des dernières années, la CO a suscité un engouement croissant de la part des chercheurs en apprentissage machine (ML) et dont le but est de produire des solutions à la fois de bonne qualité et obtenues plus rapidement que l'état de l'art. Différentes méthodes de ce domaine ont été appliquées à une grande variété de tels problèmes, tels que le CPP, le VRP et le TSP.

Un autre problème important, mais assez peu étudié en comparaison et pouvant potentiellement bénéficier de cette coopération est le problème des blocs mensuels personnalisés (CRP). Le CRP est une tâche qui se base sur des rotations, ou séquences de vols et de repos commençant et finissant au même aéroport, la base d'équipage. Ceci est fait de façon à affecter un horaire complet à chaque membre d'équipage (pilotes ou agents de bord) tout en respectant certaines exigences essentielles. Ces exigences peuvent à la fois porter sur l'ensemble des horaires d'un bloc mensuel, comme c'est le cas pour les contraintes de langue ou celles liées aux qualifications techniques de l'équipage, ou encore appliquées sur une base individuelle, tel que pour le fait de s'assurer qu'un membre d'équipage ne dépasse pas un certain nombre d'heures de vol pour la période considérée.

Dans ce chapitre, nous considérons le CRP pour pilotes avec horaires s'étendant sur un horizon d'un mois. Nous couvrons une gamme de méthodes ML afin d'obtenir une résolution accélérée du problème par rapport à la littérature actuelle tout en nous assurant d'obtenir une meilleure valeur d'objectif que d'autres méthodes concurrentes. Ce faisant, le CRP tel que considéré peut être vu comme un problème propre à tester quelles méthodes ML aident à mieux résoudre des problèmes CO difficiles. En particulier, une approche centrale à ce travail est l'agrégation dynamique de contraintes (DCA). Nous montrons le potentiel et les limites d'utiliser l'apprentissage machine en combinaison avec cette technique, et nous nous penchons sur les moyens possibles de rehausser l'étape de prétraitement réalisée par l'algorithme ML en vue de créer de meilleures données d'entrée pour DCA. Nous résumons en outre des applications réussies des méthodes ML aux blocs mensuels personnalisés, comme une planification améliorée et une résolution accélérée avec fenêtrage, puis discutons des implications de ces trouvailles.

L'emphase est principalement mise sur la famille d'algorithmes DCA et les moyens d'amélioration qui sont apportés aux données d'entrée lui étant fournies. Ceux-ci incluent le choix de l'algorithme ML, la création de variables, et une approche moins myope envers la construction d'une solution initiale par affectation successive. Ceci est en grande partie parce que les méthodes de type DCA sont particulièrement sensibles à la qualité des données d'entrée, et de telles données doivent pour cette raison chercher autant que possible à répliquer une solution optimale afin de tirer les pleins bénéfices de l'agrégation dynamique de contraintes. De plus, l'usage de DCA permet une forme de synthèse. En effet, en utilisant le CRP en tant qu'exemple d'une tâche complexe de création d'horaires d'équipage où plusieurs méthodes traditionnelles ML/RO ont été testées, nous dressons un inventaire attentif des conditions qui aident à prédire si une technique ML donnée va parvenir à accélérer le CRP. En nous concentrant en particulier sur DCA, nous voyons comment les méthodes ML peuvent se comporter lorsqu'elles doivent réaliser une étape de prétraitement pour obtenir des données d'entrée

proches d'une solution optimale. Ce genre d'effort représentant une facette de l'interaction ML/OR ayant posé problème par le passé, l'analyse approfondie présentée dans ce chapitre aide à établir un contraste entre le rendement des algorithmes ML selon qu'ils doivent répliquer des éléments de solution spécifiques ou plutôt apprendre de l'information utile mais générale.

Nous rendons cette contribution plus forte en dressant un parallèle avec les travaux réalisés dans la littérature jusqu'à présent. En discutant de la structure spéciale de problèmes d'optimisation combinatoire ou de création d'horaire d'équipage complexes, des méthodes qui leur furent appliquées, et des résultats obtenus dans chaque cas, nous généralisons jusqu'à un certain point les conclusions obtenues pour le CRP et encourageons un examen mieux structuré des façons dont ML et RO interagissent bien ensemble. L'usage de l'apprentissage machine pour le prétraitement, comme dans ce travail, cherche également à remédier en partie à une lacune dans la littérature liée au prétraitement, alors que les discussions sur l'apprentissage de bout en bout et les techniques de branchement intelligent sont déjà étoffées. La polyvalence de cette approche nous aide enfin à tester davantage de méthodes qu'il n'aurait été autrement possible, puisque les données d'entrée produites par ML peuvent être utilisées de différentes manières.

Nous donnons maintenant un plan détaillé de l'organisation de ce chapitre. Dans la section 6.2, nous donnons un aperçu global du contexte sous-jacent au travail présenté ici. Dans la section 6.3, nous introduisons l'approche DCA. Dans la section 6.4, nous décrivons la performance du prétraitement réalisé en vue du lancement de l'algorithme DCA, de même que les raisons derrière ses limites dans le contexte du CRP. Dans la section 6.5, nous présentons différentes méthodes qui ont été utilisées pour remédier aux limitations susmentionnées. Dans la section 6.6, nous donnons des exemples d'applications réussies des méthodes ML pour les blocs mensuels personnalisés. Nous concluons avec une discussion à la section 6.7.

6.2 Contexte

Dans cette section, nous fournissons le contexte qui aide à guider la discussion et la synthèse qui suivent dans les parties ultérieures de ce chapitre. Nous commençons par une revue des applications ML aux tâches CO. Nous poursuivons avec une description plus détaillée du CRP, et concluons cette section avec une présentation succincte de l'affectation successive avec ML utilisée dans ce travail, telle que déjà présentée aux chapitres précédents. Pour chaque aspect qui suit, des explications plus complètes peuvent être trouvées à ces chapitres.

6.2.1 ML et optimisation combinatoire

Les méthodes ML interagissent typiquement avec les tâches d'optimisation combinatoire au travers des trois façons suivantes. La première est l'apprentissage de bout en bout, où l'algorithme d'apprentissage est conçu pour générer une solution complète au problème cible. Il est alors en général comparé à des résultats qui représentent l'état de l'art, tels que la valeur de l'objectif obtenue par un solveur d'optimisation mathématique ou encore par d'autres méthodes ML de bout en bout.

L'apprentissage du prétraitement cherche à générer des données (par exemple, une solution) qui sont ensuite transmises au solveur d'optimisation mathématique dans le but d'en améliorer l'efficacité. Ce type d'apprentissage a été relativement peu étudié, bien qu'il soit particulièrement polyvalent. En effet, il y a plusieurs sortes de données qui peuvent être fournies en entrée au solveur, et celles-ci peuvent être utilisées de multiples façons. Ces données portent souvent sur la structure du problème à résoudre (voir Lodi et Zarpellon [60] et Bonami, Lodi et Zarpellon [61]) ou certaines parties d'une bonne solution approximative.

Le troisième et dernier type est l'apprentissage en boucle, où les poids appris par l'algorithme ML interviennent tout au long du processus de résolution à certaines étapes prédéterminées. Un exemple courant est l'usage de l'apprentissage en boucle pour le branchement en vue d'obtenir une solution entière. Chaque décision de branchement est appuyée par l'apprentissage machine. Pour une discussion plus exhaustive à ce sujet, nous référons au travail de Bengio, Lodi et Prouvost [9]. D'autres revues de littérature riches et exhaustives incluent celles présentées par Kotary *et al.* [106] et Karimi-Mamaghan *et al.* [48].

Différents types d'algorithmes ML peuvent aussi être utilisés pour chaque mode d'apprentissage indiqué ci-dessus. Les trois catégories principales d'algorithmes ML incluent l'apprentissage supervisé, non-supervisé et par renforcement. Dans le cadre de l'apprentissage supervisé, toutes les données d'entrée possibles sont mises en relation avec une sortie possible. Le but de l'algorithme est d'apprendre correctement cette relation, de telle sorte que lorsqu'une entrée est fournie, la valeur de la sortie calculée soit la bonne. Ceci est typiquement fait en étiquetant les sorties associées avec les données d'entrée disponibles et en utilisant alors ces étiquettes pour entrainer, valider et tester l'exactitude de l'algorithme. L'apprentissage non-supervisé, pour sa part, se préoccupe généralement de discerner des tendances générales parmi les données. Ce peut être utile pour la formation de regroupements distincts, ou encore pour la création de variables, tel que tenté à la section 6.5.2. Finalement, l'apprentissage par renforcement (RL) apprend une politique qui cherche à maximiser l'utilité (c'est-à-dire le signal de récompense) que dérive un agent dans l'environnement d'apprentissage lorsqu'il y prend des décisions. Un exemple dans un contexte CO serait de considérer les pilotes d'avion

comme des agents qui, en réalisant certaines rotations plutôt que d'autres, tirent un niveau de satisfaction plus élevé de leur bloc mensuel. Le défi consisterait dans ce contexte à enseigner aux planificateurs aériens à prendre des décisions intelligentes et guidées par l'algorithme RL au nom des pilotes en s'assurant que toutes les contraintes du problème soient respectées.

En général, le type d'interaction ML/RO requérant les prédictions les plus spécifiques possibles, par exemple en déterminant avec justesse ce qui serait contenu dans la solution optimale exacte du problème, est l'apprentissage de bout en bout. En effet, dans ce contexte, la solution ne peut pas être retravaillée par un solveur d'optimisation mathématique une fois produite : toute imperfection dans la solution est laissée telle quelle. L'apprentissage du prétraitement peut aussi parfois exiger un haut niveau de précision, comme lorsqu'un solveur a besoin d'une solution dont plusieurs éléments se trouvent dans la solution optimale afin d'en corriger des imperfections mineures. C'est par exemple le cas pour l'agrégation dynamique de contraintes, que nous présentons dans la section 6.3 : cette méthode est sensible à la qualité de la solution fournie, et, selon le problème, peut ne pas fonctionner à son mieux si la ressemblance entre la structure de la solution initiale et celle de la solution optimale n'est pas suffisante. Par contraste, d'autres applications de l'apprentissage du prétraitement exigent uniquement de l'information utile, et non pas forcément qu'elle soit spécifique. Finalement, l'apprentissage en boucle, telles les applications de type *learning-to-branch*, ont en général besoin d'information moins spécifique. Pour en savoir plus à ce sujet, voir Balcan *et al.* [66], Gupta *et al.* [67] et Lodi et Zarpellon (2017) [68].

Les difficultés de l'apprentissage de bout en bout ont parfois été soulignées, tel que ce fut le cas pour Glasmachers [58]. Dans son travail, l'auteur porte l'attention sur le fait que bien que de telles méthodes fonctionnent souvent lorsque appliquées à des problèmes de taille modeste, elles rencontrent des problèmes de mise en échelle au point d'avoir une performance insuffisante dans des conditions réalistes, y compris la taille du problème ou la présence de contraintes spécifiques. Cependant, cette classe de contributions ML/RO, de même que les méthodes du prétraitement requérant une solution quasi-optimale, sont potentiellement celles où les meilleurs rendements peuvent être attendus en termes d'accélération étant donné que les solutions de bout en bout peuvent éliminer entièrement l'usage d'un solveur d'optimisation mathématique. Lorsque les poids ML assurant la génération de la solution sont appris au préalable, ils peuvent faire en sorte que les temps de calcul tombent à quelques secondes pour une tâche qui prendrait normalement plusieurs minutes, voire des heures. Ces possibilités ont justifié une grande partie de l'intérêt derrière les recherches proposées dans le présent travail. Nous ajoutons que plusieurs des difficultés subies par les algorithmes de ce type ont à voir avec les modèles existants. Ils sont normalement fondés sur l'apprentissage supervisé ou par renforcement, mais les problèmes de création d'horaires d'équipage impliquent typiquement

de grands modèles à la structure complexe, pouvant souvant prendre la forme de graphes de neurones, et dont les relations et la topologie internes présentent des défis uniques.

6.2.2 Blocs mensuels personnalisés

Les tâches qui impliquent la création de blocs mensuels (personnalisés ou non) cherchent à affecter un ensemble de rotations obligatoires aux pilotes au cours du mois suivant et ainsi à créer des blocs mensuels qui respectent certaines contraintes. Les critères qui forment la base de l'objectif à optimiser en résolvant ces problèmes varient selon la nature spécifique de ces derniers. Il existe trois catégories principales de problèmes liés aux blocs mensuels. Le premier est celui des blocs anonymes. Dans ce paradigme correspondant le plus souvent aux compagnies nord-américaines, des horaires anonymes sont construits, puis les pilotes choisissent un de ces horaires en ordre décroissant d'ancienneté. Les blocs personnalisés, quant à eux, maximisent la somme des satisfactions pour l'ensemble des pilotes ou une fonction tenant compte de l'équité. Cette approche est plus fréquente au sein des compagnies européennes. Dans ce contexte, l'horaire de chaque pilote est bâti sur une base individuelle en tenant aussi compte de l'impact que son horaire a sur la satisfaction des autres. Dans le cas des blocs personnalisés avec séniorité, les blocs mensuels sont bâtis en maximisant la satisfaction de chaque pilote dans l'ordre d'ancienneté, mais en s'assurant que le problème reste réalisable pour ceux qui suivent. Un exposé approfondi des différentes formulations pour les problèmes de blocs mensuels peut être trouvé dans le travail de Kohl et Karisch [1].

La formulation retenue pour ce travail est celle des horaires personnalisés. L'objectif cherche à maximiser la somme des satisfactions pour l'ensemble des pilotes sans favoriser certains pilotes. Soit K pilotes distribués sur $d = 3$ bases. Dans cette version du problème, les contraintes suivantes s'appliquent. D'abord, chaque rotation doit être affectée à exactement un pilote, sans quoi une variable d'écart impose une pénalité. Chaque pilote doit recevoir exactement un horaire complet. Une rotation non-affectée reçoit une pénalité de C^{flight} par vol dans la rotation. Le problème inclut également des contraintes individuelles et qui assurent la réalisabilité de chaque horaire. Les horaires ne doivent pas dépasser un total de $T^{credits}$ heures par mois ni T^{work} jours de travail consécutifs. Chaque pilote doit recevoir au minimum T^{off} jours de repos au cours du mois. De plus, la fin de chaque rotation doit être suivie par une quantité suffisante de repos postcourrier, ici fixée à T^{min} heures. Une contrainte additionnelle est que chaque jour de repos prédéterminé, c'est-à-dire accordé à la suite de requêtes spéciales effectuées par les pilotes, doit être respecté pour ne pas subir une pénalité au coût rédhibitoire valant C^{day}. Les jours prédéterminés contribuent à satisfaire le minimum T^{off}.

En ce qui a trait aux préférences des pilotes, elles peuvent être liées à des vacances préférées

d'une durée de trois jours ou à des vols spécifiques. Les pilotes émettent donc à cette fin un certain nombre de préférences en lien avec les vols et les vacances. Les pilotes leur assignent des poids pour mieux indiquer l'importance relative de chacune d'entre elles. La valeur de l'objectif pour une solution donnée du CRP est alors la somme pondérée de toutes les requêtes liées à la satisfaction qui sont accordées, dont on soustrait toute pénalité liée aux variables d'écart. Le modèle complet peut être trouvé à la section 4.3.1.

Afin de résoudre le problème plus efficacement, nous utilisons la génération de colonnes. La génération de colonnes est une technique qui aide à gérer les problèmes qui ont un grand nombre, voire un nombre intraitable, de variables. Elle alterne entre deux types d'étapes : la résolution du problème maître restreint (RMP) et la phase de résolution des sous-problèmes. Durant la phase RMP, les colonnes (ou variables) disponibles sont utilisées afin de trouver une solution optimale pour ce modèle restreint. La valeur de la solution duale est alors exploitée dans la phase des sous-problèmes pour générer de nouveaux horaires avec un coût réduit négatif. Il faut pour ce faire résoudre un SPPRC (acronyme pour *shortest path problem with resource constraints*) à l'intérieur d'un graphe acyclique représentant le mois complet pour chaque pilote. Ces horaires sont alors créés et ajoutés à l'itération suivante du RMP. La procédure de génération de colonnes est inscrite à l'intérieur d'un algorithme de type *branch-and-price* pour que l'algorithme converge vers une solution entière.

Les données que nous utilisons sont prises de Kasirzadeh *et al.* [20]. Les sept jeux de données incluent chacun un ensemble de rotations, la liste des vols spécifiques les constituant, et une liste complète des aéroports servant à constituer les vols eux-mêmes. Cependant, nous altérons le nombre de pilotes inclus pour chaque instance, ainsi que leurs préférences. Ceci assure que la productivité moyenne exigée de chacun varie entre 50 et 75 heures de vol, avec les valeurs plus proches de la borne supérieure de cet intervalle représentant la productivité fréquemment attendue en milieu industriel. Pour chaque appariement d'une instance et d'un niveau moyen de productivité, nous générons 30 scénarios différents où les préférences de chaque pilote, de même que les jours de congé prédéterminés, subissent un rebrassage.

Il est important de noter qu'au cours des dernières années, le CRP a été traité par différents chercheurs dans le but d'en améliorer la résolution. Plusieurs articles ont présenté des métaheuristiques sophistiquées dans cette optique [77, 78, 80, 81, 84, 85, 88, 92]. Parmi les autres méthodes utilisées, nous mentionnons le cas de Quesnel *et al.* [63], où des réseaux de neurones sont utilisés pour restreindre le CRP à un ensemble réduit de rotations comprenant la majorité de celles qu'on peut s'attendre à trouver dans une solution optimale. Cette application a permis d'accélérer la résolution du problème. Néanmoins, le CRP a été relativement peu étudié en termes de méthodes ML en comparaison avec d'autres problèmes classiques

d'optimisation combinatoire tels que le CPP ou le TSP.

6.2.3 Affectation successive

Dans cette section, nous donnons une explication condensée de la procédure d'affectation successive (*seqAsg*) utilisée pour générer des blocs mensuels en quelques secondes et de l'algorithme ML exploité pour apprendre les poids qui guident cette procédure.

Afin d'obtenir une solution complète en moins de 15 secondes, nous utilisons le concept d'affectation successive. L'idée principale est qu'à partir du premier jour de l'horizon mensuel, un problème d'affectation est mis en place. Les pilotes disponibles pour travailler à cette date sont inclus dans la formulation du problème, de même que les rotations dont le départ survient au cours de cette période. Les variables d'affectation binaires représentent l'affectation d'une rotation, ou possiblement d'un jour de congé ou d'une période de vacances à un pilote. Ces variables sont pondérées afin de représenter la valeur utilitaire de donner une rotation ou un congé spécifique au pilote en question. Le problème est ensuite résolu pour que la somme des valeurs utilitaires pour toutes les affectations soit maximisée. Par la suite, un nouveau problème d'affectation est généré pour le deuxième jour du mois en ajustant la liste des pilotes disponibles, celle des départs quotidiens, l'utilité de chaque affectation et le bloc mensuel en cours de construction. Ces étapes sont répétées pour chaque jour du mois successivement. Nous notons que certains arcs sont omis des problèmes d'affectation afin d'éviter la violation de certaines contraintes de ressources données dans la section 6.2.2.

Le défi inhérent à cette approche est d'apprendre quels coefficients donner aux affectations, c'est-à-dire déterminer quelles affectations sont désirables et à quel point. C'est sur ce point que l'apprentissage machine est utile. En définissant des variables liées aux pilotes, rotations, et à des parties du bloc mensuel déjà créé, nous pouvons les fournir à un réseau de neurones en tant que données d'entrée. Avec l'aide de poids sur les connexions de ce réseau appris à l'aide d'un algorithme approprié, cette entrée peut être transformée en un score d'utilité prédite ou estimée différent pour chaque variable d'affectation. Nous ajoutons que les variables incluses tiennent aussi compte des affectations futures, par exemple en gardant en tête les rotations préférées ou les vacances demandées et qu'il pourrait être impossible d'accorder à la suite d'une affectation particulière. Cette notion est une des clefs du succès de la procédure et justifie l'usage de l'apprentissage machine, car l'habileté à apprendre de l'information liée au futur rend la procédure moins myope. Les affectations sont alors réalisées un jour à la fois, mais leurs coefficients sont fondés sur des périodes plus longues.

Une question importante qui subsiste ensuite se rapporte au choix de l'algorithme ML utilisé pour apprendre les poids et ainsi donner de bons coefficients pour chaque affectation. Un

obstacle majeur quant à la réponse à cette question se trouve dans l'implantation de méthodes basées sur les gradients, puisque le travail que nous présentons ici s'inscrit dans le cadre de l'apprentissage par renforcement. En effet, il n'y a pas d'étiquette à partir desquelles déterminer si une affectation a été réalisée correctement, rendant ici la possibilité d'utiliser l'apprentissage supervisé inapplicable. On pourrait certes prendre une solution optimale, ou à la rigueur quasi-optimale et étiqueter les affectations faisant partie de ces solutions, mais tel que sera à la section 6.4, ce choix présente ses propres problèmes. Pour cette raison, l'apprentissage par renforcement correspond mieux à notre objectif. Cependant, parce qu'il y a des centaines de pilotes dans les jeux de données représentatifs de la réalité de l'industrie, avec chaque pilote agissant comme un agent dans l'environnement d'apprentissage, nous devons essentiellement implanter une forme de RL multiagent. Le RL multiagent introduit une couche supplémentaire de complexité, car certaines des suppositions propres aux processus de décision de Markov ne tiennent alors plus. Ce fait rend l'implantation de méthodes basées sur les gradients significativement plus complexe.

Étant donné ces préoccupations, nous avons choisi un algorithme évolutif, la stratégie d'évolution avec adaptation de la matrice de covariance (*covariance matrix adaptation evolution strategy* ou CMA-ES). Les algorithmes évolutifs ont été notés comme compétitifs avec les méthodes de gradients pour une gamme de tâches RL différentes dans Salimans *et al.* [100]. De plus CMA-ES est un algorithme évolutif correspondant à l'état de l'art dans ce domaine (voir Hansen et Ostermeier [96]). L'idée fondamentale sous-jacente à cette approche est de prendre les poids du réseau neuronal à une itération d'entrainement donnée et d'y introduire de légers changements afin d'obtenir plusieurs ensembles de poids modifiés. Les changements apportées sont obtenus à partir d'une loi normale multivariée. Chaque ensemble de poids modifiés est alors fourni au réseau neuronal afin d'obtenir des blocs mensuels par affectation successive, et les poids modifiés donnant les meilleurs blocs mensuels sont alors conservés pour l'itération suivante. Ils sont aussi utilisés pour mettre à jour la matrice de covariance de la loi normale multivariée utilisée pour générer de nouveaux poids aux itérations suivantes.

En pratique, la qualité des blocs mensuels évalués ainsi est définie par une fonction de récompense légèrement différente d'une mesure simple de la satisfaction totale, car certains blocs mensuels peuvent être irréalisables durant l'entrainement dû à un temps de vol dépassant la limite permise. Il faut donc inclure un terme permettant d'en tenir compte et décourageant ainsi fortement cet état des faits. Pour les quelques blocs mensuels irréalisables produits par l'affectation successive une fois les poids ML appris, une heuristique de réalisabilité est appliquée de façon à retirer le plus petit nombre possible de rotations et ainsi éviter des pénalités abusives.

6.3 Agrégation dynamique de contraintes

Dans cette section, nous décrivons l'agrégation dynamique de contraintes, car il s'agit de la méthode centrale considérée dans ce chapitre. Nous notons que bien que la première version d'un algorithme DCA fut fournie par Elhallaoui *et al.* [7], de nombreux travaux récents se basent sur Elhallaoui *et al.* (2011) [8]. Cette contribution introduit le simplexe primal amélioré (*improved primal simplex* ou IPS) qui repose sur de meilleures bases mathématiques et qui est une généralisation de DCA. Afin d'éviter une confusion terminologique, nous établissons la distinction entre DCA standard et IPS, que nous regroupons à l'intérieur d'une même famille d'algorithmes DCA.

La famille de méthodes DCA cherche à accélérer la résolution de programmes linéaires de grande taille. Le principe de base est lié au phénomène de la dégénérescence. Dans le cadre d'un problème dont le modèle peut inclure des milliers de contraintes et encore davantage de colonnes ou de variables, la solution optimale du problème (ou du RMP dans le cas de la génération de colonnes) va souvent être constituée de seulement quelques colonnes non-nulles. En d'autres termes, beaucoup de variables dans les solutions de base réalisables sont nulles et la solution est hautement dégénérée. Un algorithme DCA tente de s'adresser à ce problème en agrégeant (DCA standard) ou éliminant (IPS) plusieurs des contraintes du modèle liées aux activités prévues et à affecter. Ceci a pour effet de réduire la taille du problème et de la base, réduisant de ce fait la quantité de variables nulles qui s'y trouvent.

Afin de parvenir à appliquer ce principe avec DCA standard, nous créons une partition des activités devant être affectées (soit, dans notre cas, les rotations et les jours de congé prédéterminés) en agrégats ayant pour éléments une suite d'activités qu'il est probable de voir réalisées par une même personne. Ces activités doivent être soit toutes présentes ou absentes des colonnes ajoutées au RMP. Plus formellement, soit W l'ensemble des activités devant être couvertes. Nous partitionnons W en un ensemble de L agrégats disjoints $W_l, l \in L$ et où la partition complète est $Q = \{W_l : l \in L\}$. Alors, nous disons que la colonne θ_p (ici prise indépendamment des pilotes) est compatible avec la partition Q si, pour tout $l \in L$, l'une ou l'autre des deux conditions suivantes est satisfaite :

$$T_p \cap W_l = \emptyset \tag{6.1}$$

$$T_p \cap W_l = W_l \tag{6.2}$$

où T_p est l'ensemble des activités qui apparaissent dans la colonne p. La partition Q permet ainsi l'agrégation de toutes les contraintes pour les rotations et les congés prédéterminés à l'intérieur d'un agrégat, puisque les colonnes ajoutées au RMP doivent soit toutes les

satisfaire ou n'en satisfaire aucune. Le RMP est alors réduit à ces contraintes agrégées. Les congés prédéterminés sont inclus dans certains agrégats, car en raison des contraintes (4.3), ceux-ci doivent être affectés au même titre que les rotations. Ils constituent cependant une petite minorité des congés.

Dans le cas d'IPS, aucune agrégation n'a lieu comme telle. Cependant, on considère un RMP réduit qui garde autant de contraintes que de variables strictement positives dans la base de la solution actuelle, et nous avons que l'ensemble des colonnes dans le RMP réduit sont dites compatibles avec elles dans la mesure où elles peuvent être exprimées comme une combinaison linéaire de celles-ci. L'ensemble des colonnes du RMP réduit peuvent alors être utilisées pour déterminer la partition DCA compatible avec elles. Cette partition peut être utilisée afin d'en déterminer les agrégats, notamment pour vérifier plus rapidement si une colonne est compatible avec les variables strictement positives du RMP réduit.

De plus, un algorithme DCA doit porter une attention particulière à différents aspects afin de s'assurer une implantation réussie. D'abord, durant la résolution d'un problème avec génération de colonnes, il est nécessaire d'utiliser les variables duales obtenues avec la solution optimale lors de l'itération précédente du RMP agrégé ou réduit. Cela rend possible l'évaluation des colonnes, que celles-ci soient déjà présentes ou générées pendant la phase des sous-problèmes. Cependant, comme le RMP résolu est limité aux colonnes compatibles et à un nombre réduit de contraintes, la solution duale n'est pas directement utilisable. Dans le cas de DCA standard, les variables duales sont agrégées, tandis que pour IPS, la solution duale est incomplète. Deuxièmement, la partition à une itération donnée peut ne pas être la meilleure disponible. Il est donc essentiel d'être capable de la changer quand il devient évident que certaines des colonnes incompatibles évaluées ne peuvent pas être ajoutées au RMP agrégé ou réduit alors qu'elles seraient peut-être plus désirables que celles qui sont compatibles. Finalement, nous avons besoin d'une partition initiale des activités devant être couvertes au moment de démarrer l'algorithme. La qualité de cette partition fait une différence dans l'efficacité des algorithmes de la famille DCA, et celle-ci doit donc être choisie habilement. Pour les deux premiers points mentionnés, nous indiquons la manière dont ils sont traités pour IPS et référons à Elhallaoui *et al.* [7] pour DCA standard.

Dans le but d'obtenir une solution duale complète avec IPS, un problème complémentaire est résolu. Soit A la matrice des colonnes générées jusqu'à présent et A_j la colonne j de cette matrice. Soit B la base de la solution du RMP non-réduit (i.e., comprenant toutes les colonnes générées et contraintes) actuel avec m colonnes. Soit P l'ensemble des p colonnes dans la base B et dont la valeur est strictement positive. Soit C l'ensemble des colonnes déjà générées et compatibles avec P. Soit I l'ensemble des colonnes générées incompatibles avec

P. Soit c_j le coût de la variable j. Soit π une solution duale possible au RMP non-réduit. Alors, le problème complémentaire (PC) est exprimé comme suit :

$$z_B^{PC} = \max_{y,\pi} y \tag{6.3}$$

sujet à :

$$c_j - \pi^T A_j = 0, \quad \forall \quad j \in P \tag{6.4}$$

$$c_j - \pi^T A_j \geq y, \quad \forall \quad j \in I \tag{6.5}$$

Selon cette formulation, z_B^{PC} représente la valeur optimale du problème pour la base B. On note que lorsque les équations (6.4) sont satisfaites, les contraintes de la forme de (6.5) et $y = 0$ le sont aussi pour les colonnes dans $C \setminus P$, comme démontré par Elhallaoui *et al.* [8]. La solution duale trouvée à ce problème est complète et satisfait donc toutes les contraintes de complémentarité d'écart usuelles pour RMP non-réduit.

Concernant les changements de partition, nous donnons une esquisse du principe permettant les mises à jour apportées à une partition DCA. Lorsque la solution du problème complémentaire est telle que $z_B^{PC} < 0$, il existe au moins une colonne de coût réduit négatif qui est incompatible avec les variables strictement positives de la base actuelle. Cette colonne a donc la capacité d'améliorer l'objectif du prochain RMP. À partir de ce constat, la solution du PC est utilisée pour sélectionner au plus $m - p + 1$ de ces colonnes qui sont ajoutées au RMP à l'itération suivante. La partition est alors modifiée pour être compatible avec l'union entre C et ces nouvelles colonnes.

Dans le cas où aucune des colonnes incompatibles générées jusqu'à présent n'a un coût réduit négatif, de nouvelles colonnes de coût réduit négatif sont générées par la résolution de sous-problèmes exploitant la solution duale trouvée. Celles qui sont compatibles avec la base actuelle sont ajoutées au RMP suivant, tandis que celles qui ne le sont pas sont ajoutées au problème complémentaire de l'itération suivante. En l'absence de colonnes générées à cette étape, on réalise une étape de branchement ou l'algorithme se termine si la solution est entière.

Le troisième point mentionné précédemment (c'est-à-dire la nécessité d'obtenir une bonne partition) est vraisemblablement l'aspect le plus délicat des méthodes de type DCA. Tel que montré dans Elhallaoui *et al.* [7], l'efficacité de l'agrégation dynamique de contraintes dépend directement de la qualité de la partition. Une façon d'assurer que cette dernière soit suffisamment bonne est de créer des blocs mensuels par ML et de former un agrégat pour les rotations et congés prédéterminés de chaque horaire. Cette approche est celle que nous

utilisons dans ce travail, un de ses buts étant d'analyser la capacité de méthodes ML à donner de bonnes solutions en ce sens.

Les éléments présentés ci-dessus permettent l'usage approprié de DCA. Cependant, comme mentionné dans Elhallaoui *et al.* [30], un constat qui est ressorti à l'origine en implantant de tels algorithmes se rapporte au fait que la partition est souvent raffinée en pratique. Ceci peut mener à des agrégats rapidement désagrégés, voire à une majorité de singletons. Cette caractéristique de l'application DCA pratique présente des problèmes du point vue de la performance, car l'algorithme perd alors beaucoup de sa vitesse d'exécution et donc de son utilité.

Pour surmonter cet obstacle, les auteurs ont introduit la notion d'un algorithme multiphase, où seules les colonnes respectant un certain degré d'incompatibilité maximal dépendant de la phase peuvent être évaluées. Plus formellement, ils définissent le score d'incompatibilité suivant pour chaque colonne p :

$$k_p^Q = \sum_{l \in L} \kappa_{lp} \tag{6.6}$$

où k_p^Q est le degré d'incompatibilité de la colonne p par rapport à la partition Q, et où κ_{lp} vaut 1 si la colonne p contient certaines des activités dans W_l, mais pas toutes, et 0 autrement. En d'autres mots, le score d'incompatibilité d'un horaire étant donné la partition Q est le nombre d'agrégats dans Q qui ne sont pas compatibles avec lui. Une colonne avec un score k est alors dite être k-incompatible.

Avec cette quantité, nous définissons la phase d'un algorithme DCA multiphase comme étant k si l'incompatibilité est permise jusqu'à un score de k. Alors, l'algorithme commence par évaluer les colonnes compatibles pendant la phase des sous-problèmes de la génération de colonnes (phase $k = 0$). Quand aucune colonne au coût réduit négatif n'est générée, la phase suivante commence et les colonnes avec incompatibilité $k = 1$ sont évaluées également. Ce procédé pourrait théoriquement être répété pour un nombre arbitrairement grand de phases, mais les phases sont généralement limitées à $k = 0$, $k = 1$ et $k = 2$, après quoi toutes les restrictions sur l'incompatibilité sont levées (ce à quoi on réfère parfois comme étant la phase $k = \infty$).

Un moyen d'imposer la génération de colonnes limitées à un score d'incompatibilité d'au plus k est d'introduire une nouvelle ressource sur les arcs du sous-problème de chaque pilote qui cumule le nombre d'incompatibilités du chemin à chaque nœud. Alors, en résolvant le SPPRC du pilote, les chemins sont considérés réalisables seulement s'ils n'excèdent pas la borne supérieure k sur la ressource.

6.4 Obstacles à la performance avec DCA

Dans cette section, nous traitons des limites liées à l'utilisation d'algorithmes de la famille DCA. Nous établissons que la qualité d'une solution initiale en termes de la valeur de la fonction objectif a peu d'impact sur la compatibilité des agrégats qui en découlent avec les colonnes de la solution optimale. Nous discutons également des particularités propres au CRP qui peuvent contribuer à cet état des faits.

Dans cette section, nous rapportons certaines des observations liées à l'utilisation d'un algorithme multiphase, c'est-à-dire avec ressource d'incompatibilité. Les solutions initiales sont produites en utilisant le ML en vue d'une résolution accélérée du CRP. Nous considérons trois types de solutions. Le premier correspond à celles produites par *seqAsg*. Les deux autres types de solutions ML sont obtenues en utilisant des fenêtres de 10 et 15 jours, respectivement. Dans ces deux cas, les solutions obtenues par *seqAsg* sont fournies en entrée pour le fenêtrage. La qualité de ces diverses solutions offre différents niveaux de perte d'objectif en comparaison avec les blocs mensuels de la solution trouvée par GENCOL avec la méthode *alg-basic* définie au chapitre 5. La variabilité de la perte observée pour la fonction objectif des méthodes accélérées pour obtenir une solution initiale permet de déterminer si leur objectif est lié à la compatibilité de la solution initiale avec les colonnes de la solution GENCOL.

Comme vu précédemment, la relaxation linéaire des solutions obtenues avec *alg-basic* est résolue de façon optimale, et l'écart d'intégralité reste inférieur à 0,1%. Cet écart d'intégralité, lorsqu'il est non-nul, est dû à un petit nombre de nœuds de branchement (généralement < 10). Ces solutions correspondent à l'état de l'art et les solutions optimales n'étant pas toujours connues, nous retenons la compatibilité d'une solution initiale DCA avec les solutions *alg-basic* comme indicateur de sa capacité à prédire les éléments d'une solution optimale.

Nous présentons maintenant la qualité des solutions initiales dans le tableau 6.1. Nous fournissons d'abord l'instance avec le niveau de productivité attendue, suivie du nombre de pilotes qui y sont attitrés. Chaque rangée correspond à une telle instance avec niveau de productivité. Nous rappelons que le nombre de rotations ne varie pas et demeure fixé à 1222 et 1473 rotations pour les instances dérivées de I5 et I7, respectivement. Comme au chapitre 5, nous définissons S_{GEN} comme étant la valeur moyenne de l'objectif du CRP obtenue par le solveur GENCOL version 4.5 en relation avec les cinq scénarios testés pour chaque instance et niveau de productivité moyen W. La quantité S_{ML} est l'équivalent pour les solutions ML générées par *seqAsg* avec ces mêmes scénarios. Nous avons également proposé les quantités S_{10} et S_{15} pour mesurer le score moyen atteint par les solutions améliorées avec des fenêtres de longueur 10 et 15 jours, respectivement. Nous définissons de plus L^{sat}_{ML}, L^{sat}_{10}, et L^{sat}_{15} pour

Tableau 6.1 Valeur de l'objectif et perte de satisfaction avec GENCOL, *seqAsg* et le fenêtrage

Instance	Pilotes	S_{GEN}	S_{ML}	S_{10}	S_{15}	$L^{sat}_{ML}(\%)$	$L^{sat}_{10}(\%)$	$L^{sat}_{15}(\%)$
I5-50	346	253757	236462	252823		6,82	0,37	
I5-55	315	231875	213577	231105		7,89	0,33	
I5-60	288	210863	193213	210014		8,37	0,40	
I5-65	266	195573	185979	194658	195350	4,91	0,47	0,11
I5-70	247	181200	161701	178740	180404	10,76	1,35	0,44
I5-75	230	167501	153856	164796	166790	8,15	1,61	0,43
I7-50	429	316174	307825	315346		2,64	0,26	
I7-55	391	288098	275503	287093		4,37	0,35	
I7-60	358	264681	252426	263156		4,63	0,58	
I7-65	331	245747	228589	243622	245428	6,98	0,86	0,13
I7-70	307	225507	205510	221864	224952	8,87	1,62	0,25
I7-75	287	207255	182271	201816	205906	12,05	2,62	0,65

indiquer la perte de satisfaction (mesurée en référence à S_{GEN}), en pourcentage, pour les solutions générées avec *seqAsg* ou construites avec l'aide du fenêtrage avec une longueur de 10 ou 15 jours, respectivement. Les formules pour chacun de ces pourcentages sont les mêmes qu'à la section 5.5.2 pour les méthodes *seqAsg*, *win-ML* avec fenêtres de 10 jours et *win-ML* avec fenêtres de 15 jours, respectivement.

Tel que nous pouvons le voir, les solutions passées en entrée au fenêtrage ont recouvré la majorité de la satisfaction perdue par les solutions ML initiales. Nous avons exclu l'objectif et la perte de satisfaction dans le cas des fenêtres de 15 jours pour les cas exigeant moins de productivité, puisqu'ils atteignent les mêmes résultats que *alg-basic*. Même dans cette situation, nous avons une bonne variété de valeurs pour la perte de satisfaction allant de $L^{sat}_{15} = 0,11\%$ pour I5-65 jusqu'à $L^{sat}_{ML} = 12,05\%$ pour I7-75. Cela donne un bon intervalle de valeurs pour tester si la qualité de l'objectif d'une solution initiale obtenue rapidement est liée à la compatibilité observée avec la solution optimale.

Pour vérifier si les solutions trouvées sont assez compatibles avec la solution optimale, nous avons calculé le score d'incompatibilité des horaires finaux dans les blocs mensuels générés par GENCOL à l'aide de la partition initiale inspirée des solutions ML avec ou sans fenêtrage. Les résultats sont affichés au tableau 6.2. Celui-ci est constitué des colonnes suivantes. L'instance avec niveau de productivité moyen W et la longueur de la fenêtre sont donnés afin d'indiquer les données utilisées. Le nombre de pilotes dont le bloc mensuel présente au plus un score d'incompatibilité $\tilde{k}^{Q}_{p} = 2$ avec la solution obtenue par *alg-basic* est donné. Soit ce dernier

appelé N_{low}. Nous rapportons aussi la médiane et le mode pour le nombre d'incompatibilités parmi les horaires du bloc mensuel considéré. Nous notons que nous avons seulement utilisé un scénario par instance ici, car les distributions pour les nombres d'incompatibilités étaient semblables entre les scénarios des mêmes instances, niveau de productivité et longueur de fenêtres.

À partir des résultats montrés au tableau 6.2, nous pouvons faire les remarques suivantes. D'abord, dans tous les cas, le nombre de colonnes faiblement incompatibles est limité. Si on calcule le pourcentage obtenu en divisant N_{low} par le nombre de pilotes, nous avons que la proportion des colonnes faiblement incompatibles va de 0 % avec 0 rotation sur 287 à 15,1 % avec 65 rotations sur 429. Ces valeurs sont trouvées pour I7-75 sans fenêtres et I7-50 avec ou sans fenêtres de 10 jours, respectivement. De plus, la médiane ne tombe jamais en dessous de 4 incompatibilités pour une instance, et elle est souvent plus élevée (jusqu'à 8 pour I5-75 et I7-75 avec ou sans fenêtre de 10 jours). La distribution du mode est semblable. Cela semble vrai peu importe la méthode utilisée ou le nombre d'heures de vol en moyenne par pilote : les solutions ML, qu'elles soient améliorées par fenêtrage ou non, restent fortement incompatibles avec les blocs mensuels optimaux ou quasi-optimaux fournis par GENCOL.

Tableau 6.2 Degré d'incompatibilité pour les horaires GENCOL avec les partitions ML par instance

Instance	Pilotes	Sans fenêtre			10 jours			15 jours		
		N_{low}	Médiane	Mode	N_{low}	Médiane	Mode	N_{low}	Médiane	Mode
I5-50	346	23	5	4	57	4	4			
I5-55	315	16	6	6	39	5	4			
I5-60	288	11	6	6	16	6	5			
I5-65	266	7	7	6	13	6	6	2	6	6
I5-70	247	4	8	8	3	7	6	3	7	6
I5-75	230	1	8	8	1	8	8	2	7	6
I7-50	429	65	4	5	65	4	3			
I7-55	391	31	5	5	39	5	4			
I7-60	358	7	6	5	26	6	5			
I7-65	331	6	7	7	10	6	6	15	6	6
I7-70	307	5	7	7	7	7	6	7	7	7
I7-75	287	0	8	7	5	8	8	8	7	8

Les résultats suggèrent que les données d'entrée fournies à l'algorithme DCA ne sont pas au niveau généralement requis pour en tirer les pleins avantages. Il y a essentiellement peu de colonnes faiblement incompatibles, ce qui signifie que la solution initiale est en fait moins bonne que celle trouvée par une heuristique simple ou naïve mais apte à regrouper des tâches

qui vont souvent ensemble dans une solution optimale ou correspondant à l'état de l'art. De telles relations peuvent être plus évidentes pour certains problèmes, rendant ainsi plus facile pour de telles heuristiques naïves de les repérer et de créer une bonne partition de départ pour DCA. Un exemple est le CPP, comme expliqué à la section 6.4, de même que le problème de création d'horaires et de tournées pour le transit urbain mentionné par Elhallaoui *et al.* [7]. Zeighami, Saddoune et Soumis [28] considèrent de plus les blocs mensuels personnalisés d'une façon qui permet d'obtenir des relations plus facilement : en traitant les problèmes des pilotes et copilotes simultanément, l'un peut aider à guider l'autre.

Ces observations étaient initialement surprenantes, car on aurait pu s'attendre qu'un écart de la valeur de l'objectif aussi faible que 0,11 % ou 0,13 % par rapport à *alg-basic*, comme pour les instances I5-65 et I7-65 respectivement, ait mené à des solutions dont les horaires ressemblent fortement à ceux qu'on cherche à répliquer. Il est essentiel, cependant, de réaliser que bien que la solution dont chaque horaire contient plusieurs parties du bloc mensuel optimal a en général un objectif proche de l'optimalité, la réciproque n'est pas forcément vraie. Il est certainement possible que deux solutions avec des objectifs proches l'un de l'autre soit en fait fort différentes les unes des autres s'il existe plusieurs moyens distincts (c'est-à-dire avec peu de recoupements) d'obtenir une bonne solution.

Un problème apparent en appliquant un algorithme DCA au CRP est l'exactitude attendue en prédisant à qui affecter les rotations. La famille des algorithmes DCA nécessitent en effet que la solution initiale fournie contienne certaines parties de la solution optimale elle-même pour donner le meilleur rendement possible. Ceci exige de pouvoir prédire les rotations exactes qui se trouvent dans ces parties de solutions. Or, pour le CRP, il n'y a aucune tendance constante selon laquelle les rotations sont affectées. Les rotations hautement préférées sont généralement accordées au pilote qui en fait la requête, mais pas toujours, puisque ces rotations peuvent entrer en conflit avec des vacances demandées, d'autres rotations désirées ou des jours de congé prédéterminés, ou encore sans raison immédiatement apparente. Il n'existe aucune variable basée sur les villes de départ et d'arrivée de la prochaine tâche à affecter, puisque les rotations, qui commencent et se terminent à la même base, sont déjà formées avant que le problème ne démarre. De surcroît, comme nous considérons le cas des pilotes plutôt que des agents de bord, il n'existe aucune contrainte de langue permettant au minimum de restreindre davantage les rotations considérées.

Ceci contraste par exemple avec le travail de Yaakoubi, Soumis et Lacoste-Julien [32] pour le CPP. Dans cette contribution, les auteurs cherchent à améliorer la résolution du CPP en utilisant un algorithme DCA multiphase. Ils obtiennent des gains considérables, et sont en fait capables d'obtenir une solution quasi-optimale en environ 10 % du temps normalement

requis. Ils fournissent à DCA une solution générée par ML de haute qualité. Cependant, certains faits sont à noter. D'abord, le CPP diffère du CRP en ce que, dans la plupart des cas, le vol suivant d'un membre d'équipage peut simplement être déterminé en choisissant l'appareil déjà monté jusqu'à présent. En effet, l'équipage suit en général le même avion. Il semble que ce soit en fait le cas au-delà de 98 % du temps, ce qui signifie que, dans la plupart des cas, il est possible de déterminer le vol suivant d'un membre de l'équipage sans effectuer d'apprentissage automatique. Un tel avantage n'existe pas pour le CRP.

Ceci étant dit, on note que DCA fonctionne souvent bien même avec des solutions initiales médiocres. C'est le cas de Boubaker, Desaulniers et Elhallaoui [15], qui considèrent le problème des blocs mensuels anonymes en l'absence d'information utile à la création de bonnes solutions de départ. Ils utilisent une heuristique de recherche taboue et les solutions initiales sont de piètre qualité, mais elles suffisent néanmoins à enclencher DCA. C'est dans ce genre de situations que l'apprentissage machine peut avoir le plus d'impact en apprenant à créer de bonnes solutions initiales, car une méthode déjà efficace peut l'être encore davantage avec l'apport ML. Cependant, l'absence d'information spéciale qui empêche de créer de telles solutions naïvement rend également la tâche plus difficile pour les algorithmes ML.

6.5 Améliorations ML

À la lumière des éléments présentés à la section 6.4, il semble que les solutions ML ne puissent pas atteindre le niveau typiquement attendu pour DCA en grande partie dû au fait qu'il faut prédire quelles rotations individuelles se trouvent dans l'horaire optimal d'un pilote. Dans cette section, nous présentons différentes approches qui ont été implantées dans le but de remédier à cette situation. Les méthodes considérées s'adressent à trois faiblesses méthodologiques potentielles. Premièrement, nous cherchons à voir s'il est possible d'améliorer l'algorithme ML utilisé, ou s'il est possible de lui en substituer un autre plus puissant. Ce peut inclure la modification des paramètres de notre algorithme. Ensuite, nous considérons des méthodes qui cherchent à ajouter des variables à celles fournies en entrée à l'algorithme ML. La dernière catégorie cherche à rendre l'algorithme évolutif moins myope sans en changer la nature fondamentale.

La première catégorie inclut l'usage de stratégies d'évolution naturelle et de réseaux de neurones en graphes (GNN). La deuxième implique d'une part la recherche de règles d'associations entre les aéroports apparaissant ensemble dans les rotations d'un même horaire mensuel et, d'autre part, le fait d'apprendre à partir de l'horaire idéal (et le plus souvent non-réaliste) pour chaque pilote. La dernière catégorie comprend une approche : considérer l'affectation simultanée de deux rotations consécutives à chaque étape pour voir une étape plus loin. Nous

donnons les détails de chaque méthode dans cette section.

6.5.1 Choix de l'algorithme

Dans cette section, nous considérons les méthodes qui représentent des alternatives potentielles à l'affectation successive avec CMA-ES. Nous essayons de contourner les difficultés qui y sont associées. Par contraste, les améliorations suggérées aux sections 6.5.2 et 6.5.3 essaient d'améliorer les différents aspects de l'algorithme ML à l'intérieur d'un même cadre. Nous présentons donc deux méthodes dans cette section : les stratégies d'évolution naturelle (NES) et les GNN.

Stratégies naturelles d'évolution

Une préoccupation possible avec l'algorithme CMA-ES est qu'il n'utilise pas la descente de gradients. Bien que le travail de Salimans *et al.* [100] montre que les stratégies d'évolution sont compétitives avec les méthodes axées sur les gradients pour une gamme de tâches RL, notre choix d'algorithme reste moins courant dans la littérature, et deux questions ouvertes subsistent. La première est la raison pour laquelle CMA-ES a été choisi en premier lieu, et la deuxième se préoccupe de savoir si les méthodes de gradients peuvent faire mieux.

Concernant la première question, il y a deux raisons principales pour lesquelles nous avons choisi CMA-ES comme algorithme plutôt que des méthodes axées sur les gradients. D'abord, tel que mentionné ci-dessus, les stratégies d'évolution telles que CMA-ES sont compétitives avec les méthodes de gradients sur différentes tâches RL. En outre, implanter une méthode de gradients traditionnelle présente des difficultés additionnelles pour les blocs mensuels générés par ML. En effet, la plupart des techniques RL se fient au fait qu'un seul agent circule dans l'environnement d'apprentissage. Quand cette condition est remplie, ces techniques disposent d'un protocole clair pour la mise à jour des gradients. Cependant, dans notre cas, ceci serait l'équivalent d'avoir un seul sous-problème durant la génération de colonnes. Or, des centaines de pilotes circulent dans un même réseau, et il s'agit d'un environnement multiagent. Pour cette raison, l'action suivante prise dans cet environnement, telle que décidée par la solution de l'affectation successive, n'implique pas un seul état, pas plus que l'action n'en impacte un seul. Ceci complexifie grandement la nature de l'environnement RL.

Néanmoins, la question de savoir si une méthode de gradients pourrait faire mieux dans ce contexte reste irrésolue. Pour cette raison, nous avons examiné la performance de stratégies d'évolution naturelle. Bien que cette approche constitue également un algorithme évolutif, elle est différente en ce que les poids ML générés par une distribution multinormale sont

transformés à l'aide d'une mise à jour de gradients simulant celle réalisée pour les méthodes de politiques de gradients (d'où ont émergé les méthodes acteur-critique fréquemment utilisées aujourd'hui). L'algorithme NES peut être formalisé en considérant comment l'espérance de la somme des récompenses avec un ensemble fixé de poids ML pour un bloc mensuel peut être exprimée de façon à être dérivable. En particulier, soit x l'ensemble des poids ML à l'itération actuelle. Soit θ les paramètres de la distribution gaussienne cherchant de bonnes valeurs de x. Soit $f(x)$ la fonction d'évaluation des blocs mensuels obtenus avec les poids ML x (c.-à-d., la somme des récompenses pour les horaires des blocs mensuels). Finalement, soit $\pi(x|\theta)$ la probabilité que x soit généré avec la distribution gaussienne des paramètres θ. Nous avons l'espérance conditionnelle mentionnée ci-dessus comme étant

$$J(\theta) = E_\theta[f(x)] = \int f(x)\pi(x|\theta)dx \tag{6.7}$$

À partir de ce point, nous obtenons le gradient de la récompense espérée selon les paramètres θ comme suit :

$$\nabla_\theta J(\theta) = \nabla_\theta \int f(x)\pi(x|\theta)dx = \int f(x)\nabla_\theta \pi(x|\theta)dx \tag{6.8}$$

Nous pouvons alors écrire

$$\int f(x)\nabla_\theta \pi(x|\theta)dx = \int f(x)\nabla_\theta \pi(x|\theta)\frac{\pi(x|\theta)}{\pi(x|\theta)}dx \tag{6.9}$$

$$= \int \left[f(x)\nabla_\theta \log \pi(x|\theta)\right]\pi(x|\theta)dx \tag{6.10}$$

$$= E_\theta[f(x)\nabla_\theta \log \pi(x|\theta)] \tag{6.11}$$

Avec les simplifications et reformulations données ci-dessus, nous avons montré que le gradient de l'espérance conditionnelle de la récompense totale $\nabla_\theta J(\theta)$ peut être vu comme la qualité espérée de $f(x)$ pondérée par la log-probabilité que x soit généré par la distribution des paramètres θ. L'avantage est alors de rendre le calcul des gradients estimés plus simple et plus efficace, l'espérance conditionnelle pouvant être approximée en utilisant une méthode de Monte-Carlo avec les échantillons obtenus pour cette distribution. La valeur moyenne de la qualité pondérée associée à ces échantillons est alors tenue pour représenter le gradient approximatif. Si N tels échantillons sont considérés, nous avons alors

$$\nabla_\theta \hat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N} f(x_i)\nabla_\theta \log \pi(x_i|\theta) \tag{6.12}$$

Il devient ainsi possible de réaliser une mise à jour des gradients des paramètres θ. Puisque

nous souhaitons réaliser une mise à jour avec les gradients naturels, cependant, nous utilisons la matrice d'information de Fisher $\mathbf{F}$ inversée pour ajuster les calculs. Selon la démonstration donnée par Yi *et al.* [99], nous avons que $\mathbf{F}$ peut être exprimée comme une matrice en blocs. Ces blocs sont exprimés comme la somme de termes impliquant la matrice de covariance $\mathbf{C}$ pour la distribution gaussienne générée par les paramètres θ et les coefficients $\mathbf{A}$, où $\mathbf{C} = \mathbf{A}^\intercal\mathbf{A}$ est la décomposition de Cholesky de la matrice de covariance. La structure de $\mathbf{F}$ et l'utilisation d'heuristiques spécialisées assurent alors qu'elle puisse être inversée en temps cubique. La mise à jour des gradients est alors

$$\theta \leftarrow \theta + \eta\mathbf{F}^{-\mathbf{1}}(\nabla_\theta \hat{J}(\theta) - \mathbf{b}) \tag{6.13}$$

où η est le taux d'apprentissage et $\mathbf{b}$ est un vecteur de référence qui augmente la stabilité des mises à jour.

Les mises à jour de gradients présentées par (6.13) ont l'avantage de contenir de l'information au sujet des dérivées secondes de θ d'une façon qui diminue les oscillations ou la convergence prématurée. Cependant, après avoir implanté l'algorithme décrit par Yi *et al.* [99], nous avons trouvé que l'algorithme NES ne fournit pas de meilleurs résultats que CMA-ES. En fait, la performance observée était souvent pire. Les raisons expliquant ce constat ne sont pas claires, car différents algorithmes ont souvent un rendement différent pour des problèmes variés, tel que montré par Bäck, Foussette et Krause [97]. Néanmoins, ce résultat ajoute à la crédibilité de l'idée voulant que la difficulté d'obtenir des blocs mensuels personnalisés de haute qualité a moins à voir avec le choix de l'algorithme, et davantage avec le fait que les méthodes ML cherchant à répliquer plusieurs éléments d'une solution optimale ont davantage de difficulté à atteindre leur but que celles cherchant à apprendre de l'information moins spécifique.

Réseaux de neurones en graphes

Les réseaux de neurones en graphes (GNN) sont des modèles qui cherchent à apprendre de la topologie d'un graphe. Ils s'appuient sur les réseaux de neurones traditionnels et incluent des variables liées aux voisins de chaque nœud et aux arcs. Bien qu'il y ait plusieurs formulations possibles pour les modèles des GNN, ceux-ci sont en général caractérisés par des plongements, où différentes parties de la topologie du graphe sont encapsulées par un vecteur de dimension fixée. Les poids créés par de tels plongements doivent être appris, de même que ceux qui prennent le plongement en entrée pour faire des prédictions concernant les nœuds spécifiques devant être visités ensuite par les agents circulant dans le graphe. Pour plus de contexte, plusieurs des idées dans cette section peuvent être trouvées dans le travail de Cappart *et al.* [55].

Plus formellement, un modèle de GNN peut être construit de la façon suivante. Soit le graphe $G(V, E)$ avec nœuds V et arcs E. Alors, nous pouvons définir une procédure pour trouver le plongement p-dimensionnel μ_v pour chque nœud dans V. Une telle procédure se fie à une fonction pouvant effectuer des mises à jour aux plongements, que ce soit de façon synchrone ou non, pendant un certain nombre d'itérations. Le plongement $\mu_v^{(t+1)}$ à l'itération $t+1$ est souvent lié à $\mu_v^{(t)}$ à l'itération t avec une fonction qui suit une structure générale telle que

$$\mu_v^{(t+1)} = \Psi\left(x_v, \{\mu_u^{(t)}\}_{u \in N(v)}, \{e(u,v)\}_{u \in N(v)}\right)$$

où x_v est un vecteur de variables associées au nœud v, $e(u, v)$ contient les variables pour l'arc entre les nœuds u et v, et $N(v) = \{u : (u, v) \in E\}$ est l'ensemble des nœuds voisins de v. La fonction ci-dessus doit s'assurer que l'information contenue dans le plongement des nœuds avoisinants et les variables sur les arcs est agrégée d'une façon ou d'une autre, par exemple à l'aide d'opérateurs divers de sommation ou de maximisation. Cette fonction a l'effet de propager l'information de chaque nœud un lien plus loin à chaque itération : après T mises à jour, μ_v prend en compte les nœuds jusqu'à T liens de distance.

Une fois les plongements des nœuds calculés, ils peuvent être utilisés pour effectuer des prédictions sur les nœuds devant être affectés lors de la prochaine étape. Dans ce cas, il est utile de créer une fonction aide qui tient compte de la solution partiellement bâtie au moment où l'étape d'affectation est considérée. Cette fonction peut simplement garder la trace des nœuds qui ont déjà été visités ou calculer des statistiques impliquant certaines de leurs variables. La prédiction finale pour le nœud v est alors obtenue avec

$$p_v = \phi(h(S), \mu_v)$$

où h est la fonction aide, S est l'ensemble des nœuds et des arcs dans la solution courante, et μ_v est le plongement pour le nœud v.

Nous notons que les deux fonctions Ψ et ϕ sont exprimées à l'aide de certains poids Θ qui constituent leurs paramètres et déterminent comment les arguments passés aux fonctions sont transformés. Ces poids doivent être appris, par exemple avec un algorithme tel que la descente de gradients, pour déterminer de bonnes façons d'obtenir un plongement pour la topologie d'un graphe et de l'utiliser pour réaliser de bonnes prédictions.

Les bénéfices possibles d'utiliser les GNN pour produire des solutions de haute qualité sont clairs. L'affectation successive présentée jusqu'à présent, comme la plupart des algorithmes ML, apprend peu d'information approfondie sur la topologie d'un réseau pour le CRP, alors qu'il s'agit justement du rôle des GNN. Pour cette raison, nous avons cherché à quel point il

serait réalisable d'apprendre quelles affectations donner aux pilotes avec ce type de modèle. Une préoccupation sensiblement importante est dans ce contexte la taille du problème avec lequel on travaille.

Si on considère le nombre de nœuds présents uniquement, les instances de notre problème ne sont pas d'une taille exceptionnelle dans le domaine de l'optimisation combinatoire. La plus grande des sept instances a environ 3000 nœuds, ce qui a été approché par d'autres utilisant les GNNs : voir Gasse *et al.* [65] et Khalil *et al.* [53]. Cependant, le nombre de nœuds d'un graphe n'est pas suffisant pour en appréhender la complexité, puisque sa densité est également essentielle à cette question et à la capacité des algorithmes ML à en apprendre les relations topologiques - voir Joshi *et al.* [57] à ce sujet.

Une revue des applications des GNNs aux problèmes CO peut être trouvée dans dans Zhou *et al.* [115]. Certains exemples de tâches CO traitées par les GNNs incluent le TSP, le problème d'affectation quadratique, les problèmes d'ensemble stable maximum et de couverture des sommets minimum, des enchères combinatoires, l'emplacement d'installations et plusieurs autres. Cependant, dans la plupart des cas considérés, les graphes étaient relativement peu denses. Pour cette raison, Joshi *et al.* [57] ont analysé la performance des GNN pour le TSP avec différentes tailles de problèmes. Le TSP impliquant des graphes pleinement connectés, le défi est fortement accru pour cette tâche quand le nombre total de nœuds augmente. Les auteurs ont trouvé que pour les instances avec 200 nœuds ou plus, les GNN se généralisent mal en échelle. Ils ont mentionné que certaines heuristiques visant à alléger les graphes en en retirant des arcs peu prometteurs peuvent améliorer la situation jusqu'à un certain point, comme dans le cas de Khalil *et al.* [53], mais avec certaines limites. Par exemple, la longueur moyenne d'un tour dans ce dernier article reste plus de 10 % au-delà de l'optimalité.

Dans le même ordre d'idées, le travail d'Angelini et Ricci-Tersenghi [59] a montré que de simples heuristiques gloutonnes peuvent souvent faire mieux que les GNN, comme dans le cas d'une instance difficile du problème de l'ensemble stable maximum. En conjonction avec le travail de Joshi *et al.* [57], l'état de l'art des GNN présente des problèmes évidents pour les graphes denses. Par contraste, les instances les plus difficiles du CRP parmi nos données comportent non seulement 3000 nœuds, mais aussi jusqu'à environ 250 000 arcs avec de multiples ressources. Cet état des faits suggère que pour l'instant, les GNN ne sont pas en mesure d'en exploiter la complexité pour prédire avec exactitude ce qui se retrouve dans un bloc mensuel optimal.

Il demeure important de mentionner que les GNN ont souvent été utilisés avec un succès considérable. Par exemple l'article de Gasse *et al.* [65] a aussi mené à des résultats impressionnants pour les problèmes testés. Cependant, leur modèle a été implanté dans le cadre d'un

algorithme de type *learning-to-branch*, ce qui signifie que le but n'était pas de reproduire les éléments d'une solution optimale ou compétitive avec l'état de l'art. Leur travail est plutôt une forme d'apprentissage en boucle.

6.5.2 Création de variables

Dans cette section, nous tentons d'améliorer la puissance de l'algorithme CMA-ES. Nous nous concentrons sur la création des variables additionnelles qui ne sont évidentes ou facilement conçues à partir des données disponibles. Nous commençons par mentionner des règles d'association visant à trouver des tendances dans les aéroports qui font partie de l'horaire d'un pilote. Ensuite, nous décrivons une méthode qui génère l'horaire préféré de chaque pilote indépendamment des contraintes de couverture globale du CRP de manière à apprendre à partir des rotations qui s'y trouvent.

Règles d'association

Une source possible d'information qui n'a pas encore été exploitée dans ce travail est l'ensemble des endroits parcourus par chaque horaire. Chaque rotation inclut différents aéroports, et si certaines associations riches de sens entre ces aéroports peuvent être trouvées, par exemple en reconnaissant les aéroports qui apparaissent souvent ensemble dans un même horaire, il peut y avoir de la valeur à ajouter les différents aéroports dans une rotation parmi les variables ML.

Une façon de procéder est de considérer des règles d'association, un type de ML non-supervisé qui cherche à mettre en évidence des tendances utiles dans les données disponibles. Dans cette section, nous recherchons de telles tendances pour un des scénarios de l'instance I7 avec $W = 65$. Pour ce scénario, nous mettons en commun tous les aéroports à l'intérieur d'un même horaire, de sorte que chaque pilote soit associé à une liste d'aéroports apparaissant dans son horaire respectif. Afin de décrire les règles d'association plus formellement, nous introduisons aussi les quantités suivantes :

- l'antécédent A qui fait référence aux aéroports que nous voulons utiliser comme base afin de trouver une règle d'association
- le conséquent B qui représente les aéroports qui sont censés accompagner ceux faisant partie de l'antécédent
- le support $T(A \Rightarrow B)$ fait référence à la proportion d'horaires dans le bloc mensuel comprenant $A \cup B$
- le support de l'antécédent $T(A)$ est la proportion d'horaires incluant A. De façon

analogue, le support du conséquent $T(B)$ représente la même quantité pour B.

— la confiance $C(A \Rightarrow B)$ est la probabilité conditionnelle que B soit présent dans un horaire donné lorsque A y figure aussi. Elle peut être exprimée par $C(A \Rightarrow B) = \frac{T(A \Rightarrow B)}{T(A)}$

— le levier $L(A \Rightarrow B)$ est défini comme la probabilité estimée que A et B apparaissent ensemble divisée par les probabilités que chacun soit présent indépendamment. Il peut s'écrire $\frac{T(A \Rightarrow B)}{T(A)T(B)}$

Nous pouvons alors affirmer que la règle $A \Rightarrow B$ associe le conséquent B à l'antécédent A avec support $T(A \Rightarrow B)$ et confiance $C(A \Rightarrow B)$. Le levier pour la règle donne de l'information additionnelle quant à la significativité de cette association. S'il y a un nombre suffisant de règles d'association significatives, il s'agit d'une indication que la répartition géographique des aéroports visités dans un même horaire contient des tendances intéressantes qui pourraient être apprises par un algorithme ML prenant en compte les variables liées aux aéroports.

Afin de trouver ces règles, nous devons décider certains critères qui permettent de juger si une règle est valable. De plus, un algorithme efficace doit être implanté pour trouver des règles qui satisfont tous ces critères. Pour ce qui est du premier point, nous imposons que les règles sélectionnées aient un support d'au moins 0,10 et une confiance de 0,50 ou plus. Le premier seuil est choisi pour restreindre le nombre de règles détectées à une quantité qui reste gérable et pour assurer que la relation trouvée ne soit pas marginale. Le deuxième vérifie que la règle implique une relation assez forte pour vérifier que si l'antécédent est présent, le conséquent l'est aussi plus souvent qu'autrement. Il n'y a aucune condition imposée sur le levier, mais il peut aider à affiner l'analyser des règles trouvées : dans le cas de hauts support et confiance mais avec levier faible, la règle détectée ne se manifeste que légèrement plus souvent que le hasard le prédirait et ce qui a été trouvé reste trivial.

En ce qui a trait à l'algorithme de résolution, il s'agit de l'algorithme *Apriori* présenté par Agrawal et Srikant [116]. Son principe général est de procéder itérativement avec des règles dont les ensembles de rotations sont de cardinalité croissante, en commençant avec un aéroport. Si l'aéroport est présent avec un support suffisant (i.e., un pourcentage suffisant d'horaires), il est conservé. Il est rejeté dans le cas contraire. Les aéroports retenus sont alors jumelés à un deuxième aéroport, et les combinaisons au-dessus du support minimal sont conservées tandis que les autres sont rejetées. Ce processus continue jusqu'à ce que toutes les combinaisons permises par cette méthode aient été testées. Cette procédure diminue grandement le nombre d'ensembles d'aéroports à tester, car chaque antécédent de support insuffisant, lui-même un sous-ensemble pour un certain nombre de règles plus complexes, garantit que ces règles ne satisfont pas davantage aux conditions requises.

Avant de présenter les résultats trouvés avec cette méthode, deux autres notes sont nécessaires. D'abord, les règles d'association tiennent uniquement compte de la présence ou non de certains aéroports dans un même horaire, et non du nombre d'occurrences de ces aéroports. Ensuite, nous traitons uniquement les aéroports qui ne sont pas l'une des trois bases de pilotes, car, sauf exception, chacune des trois bases apparait dans les horaires. En effet, pour nos données, bien que chaque pilote commence et termine sa rotation à une base précise, les autres sont visitées régulièrement en tant qu'escales. Les bases aériennes apportent donc peu d'information.

En appliquant la procédure esquissée ci-dessus avec les valeurs de paramètres choisies, 45 règles d'association ont été trouvées. Le tableau 6.3 montre les statistiques descriptives pour ces règles.

Tableau 6.3 Règles d'association - statistiques descriptives

Statistique	Support (antécédent)	Support (conséquent)	Support	Confiance	Levier
Compte	45	45	45	45	45
Moyenne	0,264	0,474	0,145	0,558	1,18
Écart	0,0833	0,0247	0,0414	0,0599	0,131
Minimum	0,148	0,429	0,106	0,500	1,02
Q1	0,224	0,462	0,121	0,519	1,08
Médiane	0,239	0,471	0,130	0,533	1,14
Q3	0,290	0,486	0,154	0,573	1,24
Maximum	0,505	0,505	0,260	0,755	1,55

Plusieurs observations ressortent de ce tableau. D'abord, nous pouvons voir que le support médian pour les 45 règles est 0,130, ce qui est seulement légèrement au-dessus du seuil requis de 0,10. Le maximum atteint est 0,260, suggérant ainsi qu'aucune règle n'apparait dans plus de 26,0 % des horaires. En outre, les statistiques montrées pour le levier sont instructives elles aussi : le levier médian est de 1,14, ce qui suggère que les règles trouvées surviennent seulement quelque peu plus fréquemment que le hasard. Le levier maximum trouvé de 1,55 va également dans ce sens. La distribution pour la confiance des règles d'association montre que peu d'entre elles peuvent être déterminées avec un degré de certitude élevé à partir de l'antécédent, puisque la médiane est 0,533. La valeur du troisième quartile est à peine meilleure, avec une valeur de 0,573.

Dans l'ensemble, ces résultats ne suggèrent pas qu'il y a beaucoup à apprendre des aéroports comme variables. Les règles d'association entre les aéroports ne surviennent pas fréquemment, ne tiennent pas toujours, et se distinguent peu du hasard. Pour ces raisons, nous avons choisi

de ne pas inclure ces variables comme composante de l'algorithme ML.

Horaires idéalisés

Lors de la résolution du CRP avec génération de colonnes, les sous-problèmes font appel aux variables duales trouvées à l'étape précédente pour calculer les coûts réduits. Un moyen d'avoir une meilleure idée des rotations dans l'horaire du bloc mensuel optimal pour un pilote serait de contourner ce besoin pour résoudre une version allégée du CRP dont certaines contraintes seraient levées. Pour ce faire, nous pouvons simplement résoudre un problème de plus court chemin avec contraintes de ressources dans le réseau pour le sous-problème de chaque pilote sans ajuster les coûts sur les arcs pour obtenir les coûts réduits. Chaque pilote obtient alors le meilleur horaire réalisable qu'il lui est possible d'avoir compte tenu de ses préférences. Évidemment, les blocs mensuels obtenus ainsi ne satisfont pas aux contraintes de couverture du CRP. Le principe est donc d'avoir un aperçu de ce à quoi ressemblerait le meilleur horaire pour un pilote afin de s'en inspirer.

Plus formellement, soit c_p^k le coût du chemin dans le réseau p pour le pilote $k \in K$. Soient a_{wp}^k et a_{qp}^k les variables indicatrices montrant si la rotation $w \in W$ ou le congé prédéterminé $q \in Q^k$ sont dans le chemin p, respectivement. Soient α_w, β_q et γ^k les variables duales pour l'affectation de la rotation w, l'affectation du congé prédéterminé q ou l'attribution d'un horaire au pilote k, respectivement. Alors les coûts réduits peuvent être exprimés comme

$$\overline{c}_p^k = c_p^k - \sum_{w \in W} \alpha_w a_{wp}^k - \sum_{q \in Q^k} \beta_q a_{qp}^k - \gamma^k \tag{6.14}$$

Avec les méthodes testées dans cette section, on se sert du coût c_p^k plutôt que des coûts réduits. Trouver l'horaire préféré d'un pilote est alors possible en résolvant un SPPRC modifié dans le réseau du sous-problème.

La figure 6.1 montre le nombre de rotations pour chaque pilote pour un scénario de la plus grande instance parmi nos données (I7). Afin de générer ce bloc mensuel, un paramètre additionnel a dû être ajouté à GENCOL version 4.5. pour que la version modifiée du problème décrite ci-dessus puisse être implantée. L'axe horizontal représente les pilotes numérotés de 1 à $K = 322$. L'axe vertical montre combien de rotations à poids non-nul ont été accordées au pilote. Nous ne tenons pas compte des rotations à poids nul, car seul un petit nombre de telles rotations ont tendance à apparaitre dans les différents horaires, et ce, de façon récurrente.

En analysant la figure 6.1, un fait saillant est que les valeurs les plus fréquentes pour le nombre de rotations reçues sont 1 et 2, tandis que 16 pilotes n'en reçoivent aucune. Le nombre total de rotations qui sont dans le bloc mensuel idéal de l'un ou l'autre des pilotes est 335 sur 1473

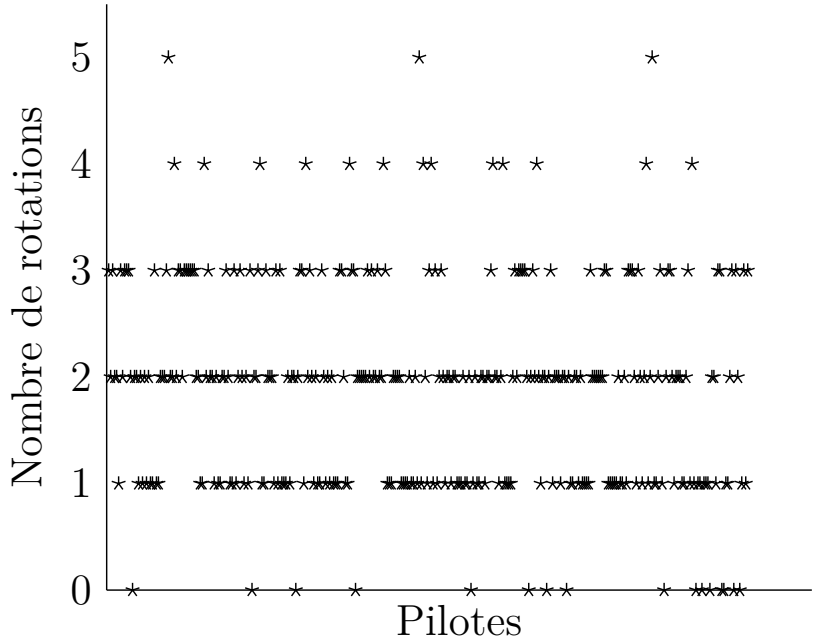


FIGURE 6.1 Nombre de rotations dans l'horaire idéal de chaque pilote, scénario d'exemple pour l'instance I7 (322 pilotes)

après avoir retiré les doublons dus aux rotations qui sont attribuées à plusieurs horaires, ou environ 22,8 % des rotations.

Ces nombres aident à expliquer pourquoi une approche aussi intuitive qu'apprendre de l'horaire idéal d'un pilote a ses limites. En effet, la plupart des rotations sont omises des horaires idéaux, car il y a peu de matière à les distinguer. Cependant, nous avons implanté cette méthode directement afin d'en constater les effets. Nous avons ajouté des variables vérifiant si une rotation ou une période de vacances figurait parmi l'horaire idéal d'un pilote (c'est-à-dire que la valeur de cette variable change selon le pilote). Ces variables supplémentaires n'ont cependant eu aucun impact sur l'apprentissage machine, car les horaires idéaux sont simplement trop éloignés de la réalité pour être utiles quand vient le temps d'affecter la grande majorité des rotations qui ne s'y trouvent pas. L'algorithme ML de base parvenait de plus déjà à affecter correctement les rotations préférées se retrouvant dans les horaires idéaux avec les variables déjà mises en place.

6.5.3 Affectation multiple

Une possibilité additionnelle est d'améliorer les prédictions exactes quant aux rotations dans la solution optimale en retravaillant la procédure d'affectation successive elle-même. Puisque *seqAsg* ne gère qu'un problème d'affectation à la fois et en ordre chronologique, l'algorithme

peut être perçu à certains égards comme étant myope. Dans ce contexte, son habileté à prendre en considération le futur dépend des variables ML qui en saisissent certains éléments. Pour cette raison, dans cette section, nous considérons une version modifiée de la procédure qui permet de résoudre un programme linéaire pour deux jours consécutifs.

Nous posons donc des problèmes d'affectation sur deux jours à la fois. Nous nous concentrons sur quatre types de paires d'affectation (ou d'affectation unique, le cas échéant) : une seule rotation commençant le premier jour ; une rotation commençant le premier jour, suivi d'une autre le jour suivant ; un jour de congé individuel le premier jour suivi d'une rotation partant le lendemain ; ou encore un temps de repos commençant le premier jour, qu'il s'agisse d'un congé individuel ou de vacances. Dans tous les cas, lorsqu'une rotation en suit une autre, il ne doit y avoir aucun conflit avec les contraintes de ressources, telles que les jours consécutifs travaillés, ou avec le repos postcourrier minimum. Nous n'avons pas considéré d'affectations multiples pour les rotations sur trois jours ou plus, comme le nombre d'affectations possibles augmente rapidement avec le nombre de jours considérés. De plus, l'usage d'affectations multiples introduit le besoin de variables additionnelles. Elles sont liées à la deuxième rotation dans la paire d'affectations (ou à son absence).

Sur cette base, nous pouvons définir un nouveau type de problème d'affectation qui inclut les rotations qui doivent être couvertes aux jours t et $t+1$. À cette fin, nous pouvons représenter le problème comme un graphe. Soit W^t l'ensemble des rotations à couvrir pour la période de deux jours. Soient P^t et Q^t les ensembles des pilotes prêts à travailler au jour t et dont la dernière arrivée était aux jours t et $t-1$ respectivement. Soit A^t l'ensemble des affectations possibles, où une affectation est admissible si elle ne crée aucun conflit de ressource ou violation d'un congé prédéterminé pour un pilote. Soit x_{ij} avec poids w_{ij} de valeur 1 si le pilote i reçoit une affectation sur deux jours j et 0 sinon. Soit a_{wj} la variable indicatrice qui prend la valeur 1 si la rotation w fait partie de l'affectation sur deux jours j et 0 autrement. Finalement, soit y_w la variable d'écart prenant la valeur 1 lorsque la rotation $w \in W$ n'est pas assignée et 0 autrement.

Alors, le problème d'affectation sur deux jours pour le jour t s'écrit

$$\max \sum_{(i,j)\in A^t} w_{ij} x_{ij} \tag{6.15}$$

sujet à :

$$\sum_{(i,j)\in A^t} a_{wj}x_{ij} + y_w = 1, \quad \forall w \in W^t \tag{6.16}$$

$$\sum_{j:(i,j)\in A^t} x_{ij} \leq 1, \quad \forall i \in P^t \tag{6.17}$$

$$\sum_{j:(i,j)\in A^t} x_{ij} = 1, \quad \forall i \in Q^t \tag{6.18}$$

$$\sum_{w\in W^t} y_w = |W^t| - M^t, \tag{6.19}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in A^t \tag{6.20}$$

où M^t est une constante pour le problème du jour t représentant le nombre de rotations qu'il est possible de retrouver dans une solution réalisable. En d'autres mots, il y a exactement le même nombre de variables d'écart de valeur 1 que le plus petit nombre de rotations ne pouvant être affectées dans aucune des solutions du problème, car les contraintes (6.16) et (6.20) prises ensemble assurent que les les variables d'écart sont binaires. Nous notons qu'une fois le problème résolu, nous affectons seulement les rotations et les congés commençant le premier des deux jours traités. Il est ainsi possible d'anticiper l'effet des décisions prises une étape plus loin pour déterminer si les affectations au premier jour permettent des affectations favorables plus tard. Ainsi, une fois le problème résolu au jour t, le problème suivant est configuré pour le jour $t+1$ comme auparavant, et non pour le jour $t+2$.

Lorsque ce modèle est lancé avec les poids ML appris par l'algorithme CMA-ES, nous n'obtenons pas de meilleures solutions qu'avant. Une différence clé, cependant, est que le temps requis pour les obtenir est plus long. En fait, il faut plus de temps pour y arriver qu'il n'en faut pour utiliser l'affectation successive suivie du fenêtrage. Par conséquent, cette méthode ne donne pas l'amélioration voulue. En effet, il semble que bénéficier de l'anticipation d'une étape supplémentaire ne permet pas de saisir suffisamment d'informations sur la topologie du réseau du CRP pour obtenir des gains tangibles. Ceci fait écho à la discussion sur les GNN developpée à la section 6.5.1.

6.6 Méthodes ML efficaces pour le CRP

Les différentes expériences montrées à la section 6.5 n'ont pas pu améliorer la qualité des données d'entrée fournies à l'algorithme DCA multiphase. Une possibilité aurait été de continuer à chercher d'autres méthodes pour accomplir ce but. Cependant, étant donné que les essais précédents avaient considéré différents types possibles de lacunes sans pouvoir mener

à la génération de blocs mensuels déviant peu de l'optimalité, nous avons plutôt opté pour exploiter la méthode d'affectation successive et l'algorithme CMA-ES tels quels, mais dans d'autres contextes. Ceux-ci sont décrits plus en profondeur dans les deux articles précédents, donc nous en fournissons seulement ici les grandes lignes pour permettre la discussion qui suit.

La première contribution, au chapitre 4, fut de montrer qu'il existe une forte relation entre la valeur de l'objectif des solutions ML obtenues par affectation successive et celle des solutions obtenues par GENCOL pour un nombre donné de pilotes. De plus, cette relation est plus solide que pour d'autres heuristiques rapides ou naïves. En utilisant les solutions ML, les employés des compagnies aériennes peuvent alors rapidement ajuster le nombre de pilotes pour une instance du CRP de sorte que les blocs mensuels soient à la fois productifs et satisfaisants pour les pilotes.

Ensuite, nous avons montré que la solution obtenue par notre heuristique ML pouvait être passée à une approche de fenêtrage et que les données ML fournies faisaient en sorte que la procédure de fenêtrage donne de meilleurs résultats qu'en l'absence d'une solution initiale. Bien que l'accélération obtenue par le fenêtrage ait été d'un facteur d'au moins 10 avec ou sans solution ML initiale, ce sont les solutions ML qui ont permis d'obtenir les meilleures solutions pour une variété d'instances.

Le contraste entre le travail présenté dans ce chapitre et les deux applications réussies que nous avons mentionnées dans cette section est frappant. Dans le premier cas, nous avons cherché à créer de meilleures solutions initiales plutôt qu'une solution médiocre afin de mieux tirer avantage de l'algorithme DCA utilisé. Cependant, il est ressorti que même pour des écarts aussi faibles que 0,11 % entre la valeur de l'objectif de solutions ML réalisables et celle des solutions optimales ou quasi-optimales fournies par GENCOL, les solutions ML restaient fortement incompatibles. Par comparaison, lorsque les solutions ML sont prises telles quelles en entrée pour la planification ou l'utilisation du fenêtrage, des blocs mensuels parfois au-delà de 10 % d'écart de l'optimalité restent utiles pour obtenir des résultats positifs.

La situation est fort semblable dans d'autres publications qui utilisent l'apprentissage machine pour améliorer la résolution des tâches de création d'horaires d'équipage. Par exemple, Morabit, Desaulniers et Lodi [64] apprennent à sélectionner certains arcs de chaque sous-problème dans le contexte du VRP avec fenêtres de temps, réduisant ainsi la taille des SP-PRC correspondants. Ici encore, toutefois, il est possible de choisir beaucoup plus d'arcs que ceux figurant exactement dans la solution finale. La sélection plus vaste ainsi réalisée permet tout de même d'obtenir des économies substantielles de temps de calcul. Les travaux de Furian *et al.* [70] font écho à ce constat à l'aide de travaux pour le VRP également. Dans ces

deux cas, alors que les algorithmes apprennent des tendances pertinentes et utiles, les méthodes employées ne peuvent pas dire quel arc ou rotation fait réellement partie de l'horaire ou de la tournée de chaque personne. Dans le premier cas, par exemple, les sous-problèmes réduits ne parviennent pas toujours à générer des colonnes et il est alors nécessaire d'utiliser le sous-problème comprenant tous les arcs pour certaines itérations. Ces résultats permettent néanmoins d'améliorer la résolution du problème complexe considéré.

Dans d'autres cas, les problèmes de création d'horaires d'équipage sont abordés à l'aide de métaheuristiques. Il peut effectivement s'agir d'une bonne approche. Or, les métaheuristiques n'impliquent habituellement pas d'émettre des prédictions. Il y a de nombreux exemples de travaux réalisés dans ce contexte, comme vu à la section 6.2.2. Le point commun de toutes les applications de métaheuristiques qui en font partie est le choix d'un algorithme intelligent permettant de mieux exploiter l'information apprise chaque itération. Il s'agit donc d'un mode de fonctionnement n'impliquant pas de prédire exactement les rotations dans la solution optimale.

La même tendance se dessine pour les problèmes CO au sens plus général. Bien que les travaux de Bello *et al.* [50] et Khalil *et al.* [53] donnent des résultats impressionnants pour certaines données, les valeurs observées sont plus éloignées de l'optimalité que pour les solveurs traditionnels standards s'attaquant à ses problèmes. En contrepartie, les articles discutant du branchement, tels que Gasse *et al.* [65] et ceux mentionnés à la section 6.2.1, mènent souvent à une accélération significative de la résolution. Ces faits saillants semblent confirmer la tendance observée à présent vu la variété des problèmes (problèmes de théorie des graphes, enchères combinatoires, emplacement d'installations) traités avec l'apprentissage de bout en bout et les algorithmes en boucle de type *learning-to-branch*.

En somme, le contraste mettant en relief les succès rencontrés (heuristique ML pour la planification et accélération du CRP avec fenêtrage) et l'incapacité à générer une solution initiale au niveau attendu par DCA s'inscrit dans le cadre de ce qui a été observé pour d'autres travaux réalisés sur le CRP et d'autres problèmes. L'analyse réalisée dans ce chapitre permet de traiter le CRP plus en profondeur tout en proposant une méthode pour évaluer systématiquement différents types d'apprentissage, soit ici plusieurs formes d'apprentissage du prétraitement, pour un même problème au lieu d'une approche plus morcelée.

6.7 Discussion générale

En résumant, le travail dans ce chapitre et les conclusions qui y sont présentées découlent d'un processus réparti sur plusieurs étapes. Nous avons considéré initialement le CRP, ou problème

des blocs mensuels personnalisés, soit un problème complexe de création d'horaires d'équipage ayant un impact important sur la qualité de vie des pilotes. Nous avions l'intention d'utiliser une méthode ML afin de générer des blocs mensuels pouvant se rapprocher de l'optimalité en prenant comme indicateur les solutions données par un solveur correspondant à l'état de l'art, GENCOL version 4.5. L'objectif était que ces solutions servent de données d'entrée de bonne qualité pour l'algorithme DCA multiphase considéré. Cependant, il est rapidement devenu clair que si ces solutions initiales apprenaient bien certains éléments, elles n'étaient pas au niveau requis pour exploiter pleinement l'approche DCA et ce, même dans le cas de solutions avec une valeur de l'objectif quasi-optimale en raison du découplage entre la solution elle-même et la valeur de l'objectif.

Plusieurs facteurs ont contribué à cet état des faits. L'utilisation du RL due à l'absence d'étiquettes pour classifier les affectations de rotations, le peu de variables riches à partir desquelles apprendre et la complexité de la topologie des réseaux au sein desquels les pilotes doivent circuler au moment de résoudre leurs sous-problèmes pour la génération de colonnes ont tous joué un rôle. Pour pallier ces difficultés, nous avons mis en place différentes mesures pour améliorer la qualité de la solution initiale fournie à l'algorithme DCA, y compris certains changements d'algorithmes ou de modèle avec les gradients naturels ou les GNN, la création de variables à l'aide de règles d'association de même que l'horaire idéal de chaque pilote et une affectation sur deux jours. Malgré ces changements, les solutions initiales sont restées incompatibles à un degré élevé avec la solution de référence du CRP. En raison des résultats obtenus, nous avons changé de cap et obtenu ce qui est décrit aux deux articles précédents tant pour la planification aérienne que l'accélération améliorée du CRP avec le fenêtrage et l'apprentissage machine.

Les idées présentées dans ce travail sont donc significatives à plusieurs égards. Ainsi, nos expériences ont mené à une première implantation du fenêtrage pour le CRP. Ensuite, les améliorations obtenues avec l'affectation successive et CMA-ES donnent de nouveaux outils pour mieux gérer le problème des blocs mensuels personnalisés. En combinaison avec les difficultés rencontrées avec DCA, ces résultats renforcent l'idée selon laquelle l'apprentissage de bout en bout ou du prétraitement dans le cas où on cherche à connaitre avec précision les éléments d'une solution unique ont plus de difficultés que l'apprentissage machine plus flexible. Cette analyse a en outre été réalisée de façon plus systématique pour un même problème, ce qui permet d'avoir une compréhension plus complète des interactions ML/RO dans ce contexte.

Idéalement, les conclusions atteintes dans ce chapitre seraient testées davantage en appliquant plusieurs techniques différentes pour chacun des problèmes où l'interaction ML/RO

est une possibilité pertinente. Cela permettrait de renforcer ou de nuancer les tendances observées liées aux apprentissages de bout en bout, du prétraitement et en boucle. Nous aurions alors également une meilleure connaissance des propriétés propres à chaque problème qui expliquent que certaines formes d'interactions ML/RO fonctionnent et d'autres non dans ces contextes. Dans certains cas, par exemple le CPP, il est possible que des éléments du problème permettent d'apprendre avec plus d'exactitude ce qui se trouve dans une solution optimale, ou encore de mieux exploiter la topologie du graphe le représentant. De plus, les algorithmes ML et les métaheuristiques, et plus particulièrement ceux appliqués à l'intersection entre l'apprentissage machine et la recherche opérationnelle, évoluent rapidement. Il est donc plausible qu'avec des algorithmes plus généralisables à grande échelle, le besoin de connaitre de façon rigoureuse leurs forces et leurs limites, ainsi que le discernement dans l'emploi de telles méthodes deviennent plus grands.

CHAPITRE 7 CONCLUSION

Les techniques de résolution des problèmes d'optimisation et l'apprentissage machine sont des domaines en constante évolution. Depuis quelques années, l'interaction entre les deux s'est également accrue, sans qu'il ne soit pour autant toujours clair quelles sont les situations où les algorithmes ML ont le plus de potentiel d'aider. Dans ce travail, nous avons considéré le problème des blocs mensuels personnalisés pour pilotes avec préférences tel que résolu par un algorithme de type *branch-and-price*. Nous avons utilisé des instances de tailles différentes et considéré la somme totale de la satisfaction des pilotes comme critère d'optimisation principal. Partant de ce contexte, nous avons mis sur pied une procédure d'affectation successive entrainée par ML afin de tester les façons dont la solution ainsi obtenue pouvait faciliter la planification du CRP par le personnel aérien ou accélérer la résolution du problème.

Dans un premier temps, nous avons effectué une revue de la littérature portant sur le CRP et les approches utilisées pour accélérer la résolution de problèmes d'optimisation complexes. Il en est ressorti d'une part, que les méthodes ML et les métaheuristiques gagnaient en popularité pour mieux résoudre les problèmes d'optimisation combinatoire en plus des méthodes d'optimisation mathématique spécialisées, et d'autre part que le succès ou non d'une méthode ML à produire de bonnes solutions dépendait du type d'apprentissage réalisé et de la nature du problème auquel il est appliqué. Or, il ne semblait pas exister de description systématique et rigoureuse des situations où les méthodes ML sont recommandables, et lesquelles le cas échéant. Nous avons considéré trois types d'apprentissage : de bout en bout, où une solution finale est obtenue par ML, par prétraitement, où certains éléments sont fournis en entrée à un optimiseur standard pour en améliorer la performance, et en boucle, où l'algorithme ML interagit avec le solveur tout au long du processus. Il est apparu que l'apprentissage du prétraitement était relativement peu couvert par la littérature, alors qu'il se distinguait précisément par sa polyvalence et la diversité de l'information qu'il pouvait transmettre à un solveur traditionnel.

Nous avons ensuite montré que la valeur de la fonction objectif d'une solution produite par affectation successive était fortement corrélée avec celle de la solution optimale ou quasi-optimale trouvée. Cette relation permet aux planificateurs aériens de mieux déterminer le nombre de pilotes devant desservir un mois donné ou être gardés en réserve. Cette méthode dépasse en efficacité et en rapidité les heuristiques naïves et métaheuristiques considérées. Par la suite, nous avons réutilisé l'affectation successive pour fournir une solution initiale à une première implantation d'une méthode d'optimisation par fenêtrage pour le CRP. Nous

avons obtenu que les temps de résolution du CRP étaient alors réduits à moins de 10 % de ce qui était observé en résolution standard dans le solveur traditionnel GENCOL version 4.5. De plus, la qualité des solutions était supérieure en utilisant les solutions initiales générées par ML, avec un écart d'optimalité inférieur à 1% en moyenne pour les instances testées. La perte d'objectif pour cette méthode était de 0,90% en moyenne par rapport à l'état de l'art, alors qu'elle était plutôt de 1,35 % pour le fenêtrage sans ML et de 7,20 % lorsque les solutions générées par l'affectation successive sont gardées telles quelles. Le fenêtrage avec ML a donc permis de réduire l'erreur observée de 33,3 % et 87,5 %, respectivement.

Dans un autre ordre d'idées, nous avons également étudié l'effet d'une solution produite par affectation successive sur le fonctionnement de l'agrégation dynamique de contraintes. Nous avons vu que malgré l'incorporation de méthodes diverses touchant au choix de l'algorithme, à la création de nouvelles variables riches en information, ou à l'affectation moins myope de rotations, les solutions correspondant à l'état de l'art demeuraient fortement incompatibles avec les solutions initiales fournies.

Dans l'ensemble, ces résultats semblent corroborer ce qui a été jusqu'à présent observé dans la littérature. Dans le cas de DCA, les solutions initiales fournies, même lorsque de bonne qualité, étaient fortement incompatibles avec les solutions optimales ou quasi-optimales trouvées par le solveur, ce qui peut mener à plus de désagrégations de contraintes et donc d'itérations. Étant donné que le CRP ne dispose pas d'éléments particuliers aidant à la prédiction, comme dans le cas du problème des rotations d'équipage, ceci n'est pas surprenant. Cependant, lorsque l'affectation successive transmettait une solution sans chercher à donner au solveur traditionnel certaines parties d'une solution correspondant à l'état de l'art, voire optimale, nous avons obtenu de bons résultats pour la planification et l'accélération du CRP.

Implications futures

Comme mentionné dans l'introduction, on peut considérer les expériences réalisées dans cette thèse comme constituant une étude de cas, c'est-à-dire celle d'un problème d'optimisation combinatoire complexe. Ces contributions ayant été réalisées sur des instances directement adaptées d'une compagnie aérienne nord-américaine, elles ont le potentiel d'appuyer et de renforcer la manière dont le CRP est géré en pratique. Cependant, ce travail ne représente qu'une première étape pour développer une compréhension plus approfondie des facteurs qui favorisent l'interaction entre ML et RO.

D'une part, les travaux préalablement réalisés sont trop peu diversifiés pour qu'on puisse avoir un véritable aperçu de la question. Bien qu'il s'agisse d'une généralisation, la plupart des travaux en matière d'interactions ML-RO étaient de type de bout en bout à l'aide de GNN ou d'autres réseaux de neurones complexes, ou encore de type en boucle avec des algorithmes in-

telligents permettant de mieux effectuer les étapes de branchement pour obtenir une solution entière. S'il existe des contributions d'autre nature, celles-ci sont trop peu nombreuses pour qu'on puisse dire avoir réalisé un tour d'horizon systématique des forces et des limites des algorithmes ML appliqués aux problèmes abordés. L'apprentissage du prétraitement, pour sa part, est peu considéré de façon générale, et a fortiori dans le cadre de problèmes complexes et bien établis, bien que les résultats obtenus lors de nos expériences montrent que ce type d'apprentissage a aussi du potentiel.

Un autre facteur dont il faut tenir compte est que les algorithmes ML progressent rapidement. Comme vu précédemment, plusieurs chercheurs s'intéressent déjà aux moyens de mieux généraliser l'apprentissage de bout en bout pour que l'entrainement effectué sur des instances plus petites puisse être repris pour d'autres plus grandes. De plus, il existe de nombreux efforts afin de rendre les technologies actuelles plus puissantes pour qu'apprendre sur une structure topologiquement complexe ne soit plus un problème intraitable.

Enfin, la question des métaheuristiques et de leur potentiel d'amélioration n'est pas à négliger. Bien que nous ayons réussi dans ce contexte-ci à obtenir des solutions rapides de façon à mener à une accélération tangible et dépassant l'état de l'art dans la planification et la résolution du CRP, le domaine des métaheuristiques donne parfois lieu à des succès remarquables. Il n'est cependant pas clair quelles situations favorisent davantage l'utilisation d'algorithmes ML, de métaheuristiques, ou d'une combinaison des deux. Jusqu'à présent, les différentes recherches effectuées, y compris celles dans ce travail, se sont contentées d'indiquer que l'approche proposée battait les autres méthodes souvent employées pour une application particulière. La question de savoir quand l'apprentissage machine vaut mieux ou non que les métaheuristiques reste donc entière.

L'objectif de cette thèse n'est donc pas d'apporter une réponse définitive à ces questions, mais plutôt d'amorcer un processus. Il existe de nombreux problèmes où l'évaluation systématique de techniques ML pourrait s'avérer révélatrice. L'implantation du fenêtrage, qui est déjà bien ancrée dans la littérature, pourrait être un bon point de départ dans le cas où elle est couplée avec un algorithme ML comme nous l'avons fait pour les blocs mensuels personnalisés. En outre, les problèmes de recherche opérationnelle étant plus structurés que ceux souvent traités par les praticiens de l'apprentissage machine en général et qui portent souvent sur le traitement de signaux naturels (par exemple la vision par ordinateur ou le traitement du langage naturel), le fait de dresser la liste de ce qui fonctionne bien ou non pour de multiples problèmes et algorithmes permettrait de développer une meilleure idée de ce qui peut être attendu en termes de résultats.

En conclusion, nous estimons qu'une meilleure connaissance de l'applicabilité de modèles ML

à la recherche opérationnelle est une question importante. En effet, l'apprentissage machine étant de plus en plus populaire, il devient de plus en plus critique d'en comprendre le potentiel et les limites. Dans ce contexte, il s'agit entre autres d'éviter le gaspillage de ressources et de talent afin de mieux orienter les chercheurs vers les techniques susceptibles de les aider à obtenir les percées voulues pour mieux gérer des problèmes d'optimisation complexes. De façon générale ou plus philosophique, on pourrait également considérer que cette démarche s'inscrit dans un questionnement d'ordre social et éthique en lien avec l'utilisation avisée d'une technologie à l'essor galopant, comme en témoignent des applications ML telles que les grands modèles de langage (aussi connus comme les *large language models*). En effet, mieux comprendre pourquoi l'apprentissage machine fonctionne ou non sur un problème donné permet de rendre son utilisation plus transparente. Cela est vrai pour les problèmes de transport, mais aussi pour de nombreuses autres applications.

RÉFÉRENCES

[1] N. Kohl et S. E. Karisch, "Airline crew rostering : Problem types, modeling, and optimization," *Annals of Operations Research*, vol. 127, nº. 1-4, p. 223–257, 2004.

[2] G. Desaulniers *et al.*, "A unified framework for deterministic time constrained vehicle routing and crew scheduling problems," dans *Fleet management and logistics*, T. G. Crainic et G. Laporte, édit. Boston, MA : Springer, 1998, p. 57–93. [En ligne]. Disponible : https://link.springer.com/chapter/10.1007/978-1-4615-5755-5_3

[3] G. B. Dantzig et P. Wolfe, "Decomposition principle for linear programs," *Operations research*, vol. 8, nº. 1, p. 101–111, 1960.

[4] J. Desrosiers *et al.*, "Time constrained routing and scheduling," *Handbooks in Operations Research and Management Science*, vol. 8, p. 35–139, 1995.

[5] C. Barnhart *et al.*, "Branch-and-price : Column generation for solving huge integer programs," *Operations research*, vol. 46, nº. 3, p. 316–329, 1998.

[6] J. Desrosiers *et al.*, "Branch-and-price," Groupe d'études et de recherche en analyse des décisions, Cahiers du GERAD G-2024-36, 2024.

[7] I. Elhallaoui *et al.*, "Dynamic aggregation of set-partitioning constraints in column generation," *Operations Research*, vol. 53, nº. 4, p. 632–645, 2005.

[8] ——, "An improved primal simplex algorithm for degenerate linear programs," *INFORMS Journal on Computing*, vol. 23, nº. 4, p. 569–577, 2011.

[9] Y. Bengio, A. Lodi et A. Prouvost, "Machine learning for combinatorial optimization : a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, nº. 2, p. 405–421, 2021.

[10] J. Desrosiers *et al.*, "A breakthrough in airline crew scheduling," Groupe d'études et de recherche en analyse des décisions, Montréal, QC, Les Cahiers du GERAD G-91-11, 1991. [En ligne]. Disponible : https://www.gerad.ca/fr/papers/G-91-11

[11] G. Desaulniers *et al.*, "Crew pairing at Air France," *European Journal of Operational Research*, vol. 97, nº. 2, p. 245–259, 1997.

[12] C. Barnhart, L. Hatay et E. L. Johnson, "Deadhead selection for the long-haul crew pairing problem," *Operations Research*, vol. 43, nº. 3, p. 491–499, 1995.

[13] B. Gopalakrishnan et E. L. Johnson, "Airline crew scheduling : State-of-the-art," *Annals of Operations Research*, vol. 140, p. 305–337, 2005.

[14] A. Caprara *et al.*, "Modeling and solving the crew rostering problem," *Operations research*, vol. 46, n°. 6, p. 820–830, 1998.

[15] K. Boubaker, G. Desaulniers et I. Elhallaoui, "Bidline scheduling with equity by heuristic dynamic constraint aggregation," *Transportation Research Part B : Methodological*, vol. 44, n°. 1, p. 50–61, 2010.

[16] A. I. Jarrah et J. T. Diamond, "The problem of generating crew bidlines," *Interfaces*, vol. 27, n°. 4, p. 49–64, 1997.

[17] J. D. Weir et E. L. Johnson, "A three-phase approach to solving the bidline problem," *Annals of Operations Research*, vol. 127, p. 283–308, 2004.

[18] I. T. Christou *et al.*, "A two-phase genetic algorithm for large-scale bidline-generation problems at delta air lines," *Interfaces*, vol. 29, n°. 5, p. 51–65, 1999.

[19] M. Gamache *et al.*, "A column generation approach for large-scale aircrew rostering problems," *Operations Research*, vol. 47, n°. 2, p. 247–263, 1999.

[20] A. Kasirzadeh, M. Saddoune et F. Soumis, "Airline crew scheduling : models, algorithms, and data sets," *EURO Journal on Transportation and Logistics*, vol. 6, n°. 2, p. 111–137, 2017.

[21] W. El Moudani *et al.*, "A bi-criterion approach for the airlines crew rostering problem," communication présentée à Evolutionary Multi-Criterion Optimization : First International Conference, EMO 2001 Zurich, Switzerland, March 7–9, 2001 Proceedings 1, Berlin, Heidelberg, 06 juillet 2001, p. 486–500.

[22] H. Dawid, J. König et C. Strauss, "An enhanced rostering model for airline crews," *Computers & Operations Research*, vol. 28, n°. 7, p. 671–688, 2001.

[23] M. Gamache *et al.*, "The preferential bidding system at air canada," *Transportation Science*, vol. 32, n°. 3, p. 246–255, 1998.

[24] H. Achour *et al.*, "An exact solution approach for the preferential bidding system problem in the airline industry," *Transportation Science*, vol. 41, n°. 3, p. 354–365, 2007.

[25] Y. Guo *et al.*, "A partially integrated airline crew scheduling approach with time-dependent crew capacities and multiple home bases," *European Journal of Operational Research*, vol. 171, n°. 3, p. 1169–1181, 2006.

[26] M. Saddoune *et al.*, "Integrated airline crew pairing and crew assignment by dynamic constraint aggregation," *Transportation Science*, vol. 46, n°. 1, p. 39–55, 2012.

[27] V. Zeighami et F. Soumis, "Combining Benders' decomposition and column generation for integrated crew pairing and personalized crew assignment problems," *Transportation Science*, vol. 53, n°. 5, p. 1479–1499, 2019.

[28] V. Zeighami, M. Saddoune et F. Soumis, “Alternating lagrangian decomposition for integrated airline crew scheduling problem,” *European Journal of Operational Research*, vol. 287, n°. 1, p. 211–224, 2020.

[29] M. E. Lübbecke et J. Desrosiers, “Selected topics in column generation,” *Operations Research*, vol. 53, n°. 6, p. 1007–1023, 2005.

[30] I. Elhallaoui *et al.*, “Multi-phase dynamic constraint aggregation for set partitioning type problems,” *Mathematical Programming*, vol. 123, p. 345–370, 2010.

[31] A. Zaghrouti, F. Soumis et I. El Hallaoui, “Integral simplex using decomposition for the set partitioning problem,” *Operations Research*, vol. 62, n°. 2, p. 435–449, 2014.

[32] Y. Yaakoubi, F. Soumis et S. Lacoste-Julien, “Flight-connection prediction for airline crew scheduling to construct initial clusters for or optimizer,” *arXiv :2009.12501*, 2020.

[33] ——, “Machine learning in airline crew pairing to construct initial clusters for dynamic constraint aggregation,” *EURO Journal on Transportation and Logistics*, vol. 9, n°. 4, 2020.

[34] M. Saddoune, G. Desaulniers et F. Soumis, “A rolling horizon solution approach for the airline crew pairing problem,” dans *2009 International Conference on Computers & Industrial Engineering*, Troyes, France, 06-09 juillet 2009, p. 344–347. [En ligne]. Disponible : https://ieeexplore.ieee.org/abstract/document/5223922

[35] L. K. Nielsen, *Rolling Stock Rescheduling in Passenger Railways : Applications in short-term planning and in disruption management*. Erasmus University, Erasmus Research Institute of Management, 2011.

[36] K. F. Abdelghany, A. F. Abdelghany et G. Ekollu, “An integrated decision support tool for airlines schedule recovery during irregular operations,” *European Journal of Operational Research*, vol. 185, n°. 2, p. 825–848, 2008.

[37] K. Gkiotsalitis et E. Van Berkum, “An exact method for the bus dispatching problem in rolling horizons,” *Transportation Research Part C : Emerging Technologies*, vol. 110, p. 143–165, 2020.

[38] P. Jaillet *et al.*, “Delivery cost approximations for inventory routing problems in a rolling horizon framework,” *Transportation Science*, vol. 36, n°. 3, p. 292–300, 2002.

[39] H. M. Afsar, M.-L. Espinouse et B. Penz, “A two-step heuristic to build flight and maintenance planning in a rolling-horizon,” communication présentée à 2006 International Conference on Service Systems and Service Management, vol. 2, Troyes, France, 25-27 octobre 2006, p. 1251–1256. [En ligne]. Disponible : https://ieeexplore.ieee.org/abstract/document/4114670

[40] M. Başdere et Ü. Bilge, "Operational aircraft maintenance routing problem with remaining time consideration," *European Journal of Operational Research*, vol. 235, nº. 1, p. 315–328, 2014.

[41] X. Qi, J. F. Bard et G. Yu, "Class scheduling for pilot training," *Operations Research*, vol. 52, nº. 1, p. 148–162, 2004.

[42] N. Secomandi et F. Margot, "Reoptimization approaches for the vehicle-routing problem with stochastic demands," *Operations Research*, vol. 57, nº. 1, p. 214–230, 2009.

[43] N. Bostel *et al.*, "Multiperiod planning and routing on a rolling horizon for field force optimization logistics," dans *The Vehicle Routing Problem : Latest Advances and New Challenges*, B. Golden, S. Raghavan et E. Wasil, édit. Boston, MA : Springer, 2008, p. 503–525. [En ligne]. Disponible : https://link.springer.com/chapter/10.1007/978-0-387-77778-8_23

[44] H. H. Millar, "The impact of rolling horizon planning on the cost of industrial fishing activity," *Computers & Operations Research*, vol. 25, nº. 10, p. 825–837, 1998.

[45] H. D. Sherali, S. M. Al-Yakoob et M. M. Hassan, "Fleet management models and algorithms for an oil-tanker routing and scheduling problem," *IIE Transactions*, vol. 31, nº. 5, p. 395–406, 1999.

[46] M. Alimian *et al.*, "Solving a parallel-line capacitated lot-sizing and scheduling problem with sequence-dependent setup time/cost and preventive maintenance by a rolling horizon method," *Computers & Industrial Engineering*, vol. 168, 2022.

[47] L. Scavuzzo *et al.*, "Machine learning augmented branch and bound for mixed integer linear programming," *Mathematical Programming*, p. 1–44, 2024.

[48] M. Karimi-Mamaghan *et al.*, "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems : A state-of-the-art," *European Journal of Operational Research*, vol. 296, nº. 2, p. 393–422, 2022.

[49] N. Mazyavkina *et al.*, "Reinforcement learning for combinatorial optimization : A survey," *Computers & Operations Research*, vol. 134, 2021.

[50] I. Bello *et al.*, "Neural combinatorial optimization with reinforcement learning," *arXiv :1611.09940*, 2016.

[51] M. Nazari *et al.*, "Reinforcement learning for solving the vehicle routing problem," communication présentée à Advances in neural information processing systems, vol. 31, 2018.

[52] W. Kool, H. van Hoof et M. Welling, "Attention, learn to solve routing problems!" *arXiv :1803.08475*, 2019.

[53] E. Khalil *et al.*, “Learning combinatorial optimization algorithms over graphs,” communication présentée à Advances in neural information processing systems, vol. 30, 2017.

[54] A. Nowak *et al.*, “Revised note on learning quadratic assignment with graph neural networks,” communication présentée à 2018 IEEE Data Science Workshop (DSW), Lausanne, Suisse, 04-06 juin 2018. [En ligne]. Disponible : https://ieeexplore.ieee.org/abstract/document/8439919

[55] Q. Cappart *et al.*, “Combinatorial optimization and reasoning with graph neural networks,” *Journal of Machine Learning Research*, vol. 24, nº. 130, p. 1–61, 2023.

[56] C. K. Joshi, T. Laurent et X. Bresson, “An efficient graph convolutional network technique for the travelling salesman problem,” *arXiv :1906.01227*, 2019.

[57] C. K. Joshi *et al.*, “Learning the travelling salesperson problem requires rethinking generalization,” *Constraints*, vol. 27, nº. 1-2, p. 70–98, 2022.

[58] T. Glasmachers, “Limits of end-to-end learning,” communication présentée à Asian conference on machine learning, vol. 77, 2017, p. 17–32.

[59] M. C. Angelini et F. Ricci-Tersenghi, “Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set,” *Nature Machine Intelligence*, vol. 5, nº. 1, p. 29–31, 2023.

[60] M. Kruber, M. E. Lübbecke et A. Parmentier, “Learning when to use a decomposition,” communication présentée à International conference on AI and OR techniques in constraint programming for combinatorial optimization problems, Padoue, Italie, 05-08 juin 2017, p. 202–210. [En ligne]. Disponible : https://link.springer.com/chapter/10.1007/978-3-319-59776-8_16

[61] P. Bonami, A. Lodi et G. Zarpellon, “A classifier to decide on the linearization of mixed-integer quadratic problems in CPLEX,” *Operations Research*, vol. 70, nº. 6, p. 3303–3320, 2022.

[62] M. Beulen, L. Scherp et B. F. Santos, “Dynamic evaluation of airline crew’s flight requests using a neural network,” *EURO Journal on Transportation and Logistics*, vol. 9, nº. 4, 2020.

[63] F. Quesnel *et al.*, “Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering,” *Computers & Operations Research*, vol. 138, 2022.

[64] M. Morabit, G. Desaulniers et A. Lodi, “Machine-learning-based arc selection for constrained shortest path problems in column generation,” *INFORMS Journal on Optimization*, vol. 5, nº. 2, p. 191–210, 2023.

[65] M. Gasse *et al.*, “Exact combinatorial optimization with graph convolutional neural networks,” communication présentée à Advances in neural information processing systems, vol. 32, 2019.

[66] M.-F. Balcan *et al.*, “Learning to branch,” communication présentée à International conference on machine learning, vol. 80, Stockholm, Sweden, 10-15 juillet 2018, p. 344–353. [En ligne]. Disponible : https://proceedings.mlr.press/v80/balcan18a.html

[67] P. Gupta *et al.*, “Hybrid models for learning to branch,” communication présentée à Advances in Neural Information Processing Systems, vol. 33, 2020, p. 18 087–18 097.

[68] A. Lodi et G. Zarpellon, “On learning and branching : a survey,” *TOP*, vol. 25, p. 207–236, 2017.

[69] Y. Tang, S. Agrawal et Y. Faenza, “Reinforcement learning for integer programming : Learning to cut,” dans *International conference on machine learning*, 2020, p. 9367–9376. [En ligne]. Disponible : https://proceedings.mlr.press/v119/tang20a.html

[70] N. Furian *et al.*, “A machine learning-based branch and price algorithm for a sampled vehicle routing problem,” *OR Spectrum*, vol. 43, n°. 3, p. 693–732, 2021.

[71] M. Morabit, G. Desaulniers et A. Lodi, “Machine-learning–based column selection for column generation,” *Transportation Science*, vol. 55, n°. 4, p. 815–831, 2021.

[72] P. Pereira *et al.*, “Learning to branch for the crew pairing problem,” *Les Cahiers du GERAD*, vol. 711, p. 2440, 2022.

[73] R. Václavík *et al.*, “Accelerating the branch-and-price algorithm using machine learning,” *European Journal of Operational Research*, vol. 271, n°. 3, p. 1055–1069, 2018.

[74] J. Nillius, “Deep learning in state of the art airline crew rostering algorithms,” mémoire de maîtrise, Department of Computer Science and Engineering, Chalmers University of Technology et University of Gothenburg, Gothenberg, Suède, 2022.

[75] H. T. Ozdemir et C. K. Mohan, “Flight graph based genetic algorithm for crew scheduling in airlines,” *Information Sciences*, vol. 133, n°. 3-4, p. 165–173, 2001.

[76] P. Lučic et D. Teodorović, “Simulated annealing for the multi-objective aircrew rostering problem,” *Transportation Research Part A : Policy and Practice*, vol. 33, n°. 1, p. 19–45, 1999.

[77] P. Lučić et D. Teodorović, “Metaheuristics approach to the aircrew rostering problem,” *Annals of Operations Research*, vol. 155, p. 311–338, 2007.

[78] J. de Armas *et al.*, “A multi-start randomized heuristic for real-life crew rostering problems in airlines with work-balancing goals,” *Annals of Operations Research*, vol. 258, p. 825–848, 2017.

[79] A. A. Juan *et al.*, “MIRHA : multi-start biased randomization of heuristics with adaptive local search for solving non-smooth routing problems,” *TOP*, vol. 21, p. 109–132, 2013.

[80] N. Souai et J. Teghem, “Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem,” *European Journal of Operational Research*, vol. 199, nº. 3, p. 674–683, 2009.

[81] G.-F. Deng et W.-T. Lin, “Ant colony optimization-based algorithm for airline crew scheduling problem,” *Expert Systems with Applications*, vol. 38, nº. 5, p. 5787–5793, 2011.

[82] M. Deveci et N. C. Demirel, “A survey of the literature on airline crew scheduling,” *Engineering Applications of Artificial Intelligence*, vol. 74, p. 54–69, 2018.

[83] S. Saemi *et al.*, “Solving an integrated mathematical model for crew pairing and rostering problems by an ant colony optimisation algorithm,” *European Journal of Industrial Engineering*, vol. 16, nº. 2, p. 215–240, 2022.

[84] B. Maenhout et M. Vanhoucke, “A hybrid scatter search heuristic for personalized crew rostering in the airline industry,” *European Journal of Operational Research*, vol. 206, nº. 1, p. 155–167, 2010.

[85] C. Boufaied *et al.*, “A construction of rotations-based rosters with a genetic algorithm,” communication présentée à 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, Canada, 24-29 juillet 2016, p. 2389–2394. [En ligne]. Disponible : https://ieeexplore.ieee.org/abstract/document/7744084

[86] F. Z. Sargut, C. Altuntaş et D. C. Tulazoğlu, “Multi-objective integrated acyclic crew rostering and vehicle assignment problem in public bus transportation,” *OR Spectrum*, vol. 39, nº. 4, p. 1071–1096, 2017.

[87] S.-Z. Zhou *et al.*, “A multi-objective ant colony system algorithm for airline crew rostering problem with fairness and satisfaction,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, nº. 11, p. 6784–6798, 2020.

[88] T. Zhou *et al.*, “A hybrid multi-objective genetic-particle swarm optimization algorithm for airline crew rostering problem with fairness and satisfaction,” communication présentée à International Conference on Machine Learning for Cyber Security, Guangzhou, China, 02-04 décembre 2022, p. 563–575.

[89] T. Doi, T. Nishi et S. Voß, “Two-level decomposition-based matheuristic for airline crew rostering problems with fair working time,” *European Journal of Operational Research*, vol. 267, nº. 2, p. 428–438, 2018.

[90] B. Santosa, A. Sunarto et A. Rahman, “Using differential evolution method to solve crew rostering problem,” *Applied Mathematics*, vol. 1, n°. 4, p. 316–325, 2010.

[91] A. Legrain, J. Omer et S. Rosat, “A rotation-based branch-and-price approach for the nurse scheduling problem,” *Mathematical Programming Computation*, vol. 12, p. 417–450, 2020.

[92] C.-Y. Chan *et al.*, “A hyper-heuristic inspired by pearl hunting,” communication présentée à International Conference on Learning and Intelligent Optimization, vol. 7219, 2012, p. 349–353.

[93] S. Er-Rbib *et al.*, “Preference-based and cyclic bus driver rostering problem with fixed days off,” *Public Transport*, vol. 13, n°. 2, p. 251–286, 2021.

[94] H. Lu, X. Zhang et S. Yang, “A learning-based iterative method for solving vehicle routing problems,” communication présentée à International conference on learning representations, 2020. [En ligne]. Disponible : https://openreview.net/forum?id=BJe1334YDH

[95] Y. Wu *et al.*, “Learning improvement heuristics for solving routing problems,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, n°. 9, p. 5057–5069, 2021.

[96] N. Hansen et A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies,” *Evolutionary Computation*, vol. 9, n°. 2, p. 159–195, 2001.

[97] T. Bäck, C. Foussette et P. Krause, *Contemporary evolution strategies*. Berlin, Heidelberg : Springer, 2013, vol. 86.

[98] D. E. Moriarty, A. C. Schultz et J. J. Grefenstette, “Evolutionary algorithms for reinforcement learning,” *Journal of Artificial Intelligence Research*, vol. 11, p. 241–276, 1999.

[99] S. Yi *et al.*, “Stochastic search using the natural gradient,” communication présentée à Proceedings of the 26th annual international conference on machine learning, 2009, p. 1161–1168. [En ligne]. Disponible : https://dl.acm.org/doi/abs/10.1145/1553374.1553522

[100] T. Salimans *et al.*, “Evolution strategies as a scalable alternative to reinforcement learning,” *arXiv :1703.03864*, 2017.

[101] K. O. Stanley, D. B. D’Ambrosio et J. Gauci, “A hypercube-based encoding for evolving large-scale neural networks,” *Artificial Life*, vol. 15, n°. 2, p. 185–212, 2009.

[102] A. J. Schaefer *et al.*, “Airline crew scheduling under uncertainty,” *Transportation Science*, vol. 39, n°. 3, p. 340–348, 2005.

[103] P. Cappanera et G. Gallo, “A multicommodity flow approach to the crew rostering problem,” *Operations Research*, vol. 52, n°. 4, p. 583–596, 2004.

[104] O. Vinyals, M. Fortunato et N. Jaitly, “Pointer networks,” communication présentée à Advances in Neural Information Processing Systems, vol. 28, 2015.

[105] D. Bahdanau, K. Cho et Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv :1409.0473*, 2014.

[106] J. Kotary *et al.*, “End-to-end constrained optimization learning : A survey,” *arXiv :2103.16378*, 2021.

[107] A. Tahir *et al.*, “An improved integral column generation algorithm using machine learning for aircrew pairing,” *Transportation Science*, vol. 55, n°. 6, p. 1411–1429, 2021.

[108] F. Quesnel, G. Desaulniers et F. Soumis, “A branch-and-price heuristic for the crew pairing problem with language constraints,” *European Journal of Operational Research*, vol. 283, n°. 3, p. 1040–1054, 2020.

[109] ——, “Improving air crew rostering by considering crew preferences in the crew pairing problem,” *Transportation Science*, vol. 54, n°. 1, p. 97–114, 2020.

[110] C. F. Higham et D. J. Higham, “Deep learning : An introduction for applied mathematicians,” *Siam Review*, vol. 61, n°. 4, p. 860–891, 2019.

[111] D. Silver *et al.*, “Deterministic policy gradient algorithms,” communication présentée à International conference on machine learning, Beijing, China, 21-26 juillet 2014, p. 387–395.

[112] P. Racette *et al.*, “Gaining insight into crew rostering instances through ML-based sequential assignment,” *TOP*, vol. 32, n°. 3, p. 537–578, 2024.

[113] G. Desaulniers, J. Desrosiers et M. M. Solomon, “Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems,” dans *Essays and Surveys in Metaheuristics*, C. C. Ribeiro et P. Hansen, édit. Boston, MA : Springer US, 2002, p. 309–324. [En ligne]. Disponible : https://link.springer.com/chapter/10.1007/978-1-4615-1507-4_14

[114] F. Quesnel, G. Desaulniers et F. Soumis, “A new heuristic branching scheme for the crew pairing problem with base constraints,” *Computers & Operations Research*, vol. 80, p. 159–172, 2017.

[115] J. Zhou *et al.*, “Graph neural networks : A review of methods and applications,” *AI Open*, vol. 1, p. 57–81, 2020.

[116] R. Agrawal et R. Srikant, “Fast algorithms for mining association rules,” communication présentée à Proc. 20th int. conf. very large data bases, VLDB, vol. 1215, 1994, p. 487–499.