

ÉCOLE DOCTORALE SCIENCES TECHNOLOGIES ET SANTÉ N° 585

Optimization of passenger flows and resources management through machine learning and constraint programming techniques

DOCTORAL THESIS

presented and publicly defended on 21 December, 2023

in partial fulfillment of requirements for the degree of

Doctor of Philosophy of Artois University
in the subject of Computer Science

by

Thibault Falque

Doctoral Committee

<i>Advisors:</i>	Bertrand Mazure	Artois University, Lens
	Karim Tabia	Artois University, Lens
<i>Supervisor:</i>	Christophe Lecoutre	Artois University, Lens
<i>Reviewers:</i>	Nadjib Lazaar	Montpellier University, Montpellier
	Frédéric Saubion	Angers University, Angers
<i>Examiners:</i>	Stéphanie Roussel	Onera, Toulouse
	Elise Vareilles	ISAE SUPAERO, Toulouse
<i>Guest:</i>	Pierre Talbot	University of Luxembourg

CENTRE DE RECHERCHE EN INFORMATIQUE DE LENS – CNRS UMR 8188
Université d’Artois, rue Jean Souvraz, S.P. 18 F-62307, Lens Cedex France
Secrétariat : +33 (0)3 21 79 17 23
<http://www.cril.univ-artois.fr>

Mise en page avec memcrl (B. Mazure, CRIL) et thloria (D. Roegel, LORIA).

Acknowledgements

Mais, vous savez, moi je ne crois pas qu'il y ait de bonne ou de mauvaise situation. Si je devais, aujourd'hui, résumer cette aventure académique, je dirais avant tout que c'est une histoire de rencontres, des êtres lumineux qui, peut-être à des moments où la solitude pesait lourd dans mon quotidien universitaire, ont tendu une main, une idée, un sourire. Il est étrange de penser comment les rencontres, les discussions apparemment fortuites, ont pu façonner un destin, tisser la toile de ma thèse.¹

Et parmi ces rencontres, trois étoiles ont brillé avec une intensité particulière : Bertrand Mazure, Christophe Lecoutre et Karim Tabia, mes encadrants de thèse. Bertrand avec qui j'ai partagé tant de discussions le soir sur le parking de la faculté. Ces moments, loin d'être de simples échanges, ont été le creuset où se sont forgées mes idées, où ma thèse a pris forme.

Dans ce labyrinthe qu'est la recherche, j'ai eu la chance de croiser le chemin de personnes extraordinaires. Chez Exakis Nelite, Ophélie Censier, Nicolas Cheymol, David Huard et Victor Salas ont été plus que des collègues ; vous avez été des alliés, des amis, m'offrant votre soutien et votre expertise. Céline Barthélémy, Arnaud Berrebi, Maxence Censier, Eric Prevost, Didier Zeitoun, merci pour avoir cru en moi et pour m'avoir guidé dans cette aventure professionnelle.

À l'Aéroport de Paris, Olivier Gosseaume, Hervé Nicolas, Christophe Sambo, Tom Weschler et l'ensemble des personnes que j'ai pu croiser, votre accueil et votre ouverture d'esprit ont enrichi ma recherche d'une dimension pratique indispensable.

Dans l'enceinte du village où j'ai passé beaucoup de mon temps ces dernières années, Daniel Le Berre et Pierre Marquis ont été les druides capables de vaincre la machinerie de l'administration Gallo-Romaine, en réussissant à obtenir mon laisser-passer doctoral.² Nadjib Lazaar et Frédéric Saubion, maîtres scribes et rapporteurs, vous avez scruté mon travail avec rigueur, contribuant à en façonner la version finale. Elise Vareille et Stéphanie Roussel, j'ai eu la chance de vous rencontrer lors de mes aventures à l'intérieur ou à l'extérieur des frontières gauloises et vous avez accepté d'être membres du conseil des sages ayant évalué mon travail, merci pour cela.

Ma compagne Ermelinda, mes parents, l'ensemble de ma famille, mes amis d'enfance, vous avez été le roc sur lequel je m'appuyais dans les moments de tempête. Votre amour, votre soutien sans faille ont été les vents qui ont gonflé les voiles de mon voyage, me portant vers des horizons que je n'aurais jamais cru atteignables.

Et comment ne pas évoquer mes compagnons de route, les autres doctorants et l'ensemble du village ? Vous avez été ma famille académique, partageant avec moi les succès et les échecs.

Le FCH Lens, mon échappatoire, mon oxygène. Les entraînements et les moments partagés avec l'équipe ont été des bouffées d'air frais dans le tumulte de la rédaction.

Dans cette aventure, je suis porté par l'affection, le respect et la gratitude envers toutes ces personnes qui ont marqué le chemin de ma thèse. Merci à vous tous.

¹Ce passage fait référence au célèbre monologue du scribe dans le film Astérix et Obélix Mission Cléopâtre.

²Ce laisser-passer est une référence au laisser-passer A38 de la bande dessinée : Les 12 travaux d'Astérix.

Résumé

L’aviation civile, en tant que pilier fondamental de notre ère globalisée, facilite non seulement les déplacements mais enrichit également les interactions culturelles et économiques à une échelle sans précédent. Au cœur de cette infrastructure se trouvent les aéroports, tels que Paris Charles de Gaulle (CDG), qui se distingue par son envergure et sa complexité opérationnelle. En 2019, CDG a accueilli environ 76 millions de passagers et géré 498000 décollages et atterrissages, connectant 328 destinations dans 119 pays, ce qui le positionne comme le deuxième aéroport le plus fréquenté d’Europe et le neuvième au monde en termes de trafic passagers. Ces chiffres, issus du rapport annuel 2019³ des Aéroports de Paris, mettent en lumière l’ampleur des opérations et la nécessité d’une gestion optimisée pour faire face à l’afflux de passagers et de fret.

Malgré son importance et son volume de trafic, CDG ne prévoit pas actuellement de nouveaux développements majeurs en termes de terminaux, ce qui souligne un défi crucial : gérer et optimiser les flux de passagers et de ressources dans l’infrastructure existante. Cette situation est d’autant plus pressante que, bien que la pandémie de COVID-19 ait entraîné une baisse significative du trafic aérien, une reprise est observée, alignée sur les tendances pré-pandémiques, comme le rapporte Le Monde⁴. Ce rebond souligne l’importance cruciale de la préparation et de l’optimisation pour répondre à la demande croissante et éviter les goulots d’étranglement, garantissant ainsi une efficacité opérationnelle continue.

Dans cette optique, cette thèse se consacre à l’exploration de solutions innovantes pour surmonter les défis inhérents aux opérations des mégahubs aéroportuaires, en se penchant sur les méthodologies de la programmation par contraintes (CP) et de l’apprentissage automatique (ML).

La programmation par contraintes (*Constraint Programming* (CP)) [Apt03, RvW06, Lec09] a été introduite et formalisée dans les années 1960 et 1970. C’est un paradigme puissant et reconnu pour la modélisation et la résolution de problèmes variés, allant de la planification quotidienne aux défis combinatoires complexes dans des domaines aussi divers que la configuration, la planification et la bioinformatique. La force de la CP réside dans sa capacité à simplifier la modélisation des problèmes en étant aussi proche que possible de la description en langage naturel du problème. Elle fournit des méthodes génériques pour modéliser et résoudre des problèmes en spécifiant des variables de décision et des contraintes qui définissent les relations entre ces variables. En plus des contraintes standards, la CP utilise également des contraintes globales qui modélisent des relations sémantiques spécifiques et peuvent être appliquées à un nombre arbitraire de variables, rendant la modélisation plus simple et plus efficace. Il existe actuellement plus de 400 contraintes globales [VHK06, BCDP07]. Des langages de modélisation, comme le

³<https://www.parisaeroport.fr/docs/default-source/groupe-fichiers/finance/rerelations-investisseurs/information-financi%C3%A8re/rapports-annuels/report-on-activity-and-sustainable-development-2019.pdf>

⁴https://www.lemonde.fr/economie/article/2023/06/08/1-a380-s-impose-comme-l-avion-de-la-reprise-du-traffic-aerien_6176792_3234.html

format **XCSP3** offrent un ensemble de base de contraintes globales suffisantes pour modéliser la plupart des problèmes [BLAP20].

Le problème classique de satisfaction de contraintes (CSP) consiste à déterminer si un réseau de contraintes donné est satisfiable en fournissant une solution, si elle existe. La décision de savoir si un CSP est cohérent est un problème NP-complet [Mac77a].

Les problèmes sont résolus à l’aide de solveurs spécialisés qui emploient des méthodes génériques pour explorer efficacement l’espace de recherche et trouver des solutions satisfaisant toutes les contraintes.

Cette approche est extrêmement pertinente pour des problèmes combinatoires complexes tels que l’attribution des vols aux emplacements de stationnement et l’attribution des bureaux d’enregistrement dans les opérations aéroportuaires, où diverses contraintes et objectifs d’optimisation doivent être considérées.

Notre approche a donc consisté tout d’abord à proposer une modélisation sous la forme d’un COP (*Constraint Optimization Problem* (COP)) du problème de planification des bornes d’enregistrement tel que défini à l’aéroport Charles de Gaulle. Pour modéliser ce problème, divers langages de modélisation ou bibliothèques existent, tels que OPL [P. 99], MiniZinc [NSB⁺07, SBF10], Essence [FGJ⁺07] et PyCSP³ [LS20]. Le choix s’est porté sur la bibliothèque Python PyCSP³ récemment développée, qui permet de générer des instances spécifiques dans le format XCSP3, reconnu par des solveurs CP de l’état de l’art comme ACE (AbsCon Essence) [Lec23], OsaR [Osc12], Choco [PFL16] et PicatSAT [ZKF17].

Plusieurs variables et contraintes sont introduites pour représenter les différentes composantes du problème, telles que les enregistrements, les tâches et les comptoirs d’enregistrement, ainsi que les règles d’exclusivité, de disponibilité et de satisfaction des compagnies aériennes.

En plus de l’aspect modélisation, nous avons également proposé une contrainte globale et son propagateur pour représenter les contraintes **AllDiffExcept**.

Différents modèles pour le problème d’attribution des comptoirs d’enregistrement ont été présentés, avec trois formulations distinctes du problème et des améliorations spécifiques. L’approche utilisant la méthode **Gather** pour la contrainte **AllDiffExcept** a constamment montré d’excellentes performances. La comparaison avec l’algorithme ADP a démontré que, bien que des algorithmes plus simples puissent fournir des solutions, l’utilisation de solveurs génériques améliore significativement les performances, la qualité des solutions et la capacité d’évolution du système. Ces travaux sont actuellement déployés en prévisualisation et seront intégralement mis en production d’ici janvier 2024.

Ensuite, des modèles variés pour résoudre le problème d’allocation des places de stationnement pour avions sont explorés, prenant en compte diverses contraintes et objectifs, notamment la satisfaction des compagnies aériennes. Au total, trois formulations du problème sont proposées : une approche classique basée principalement sur des contraintes d’extension, une approche “allDifferent” qui inclut des contraintes “allDifferent” pour chaque clique maximale issue du graphe d’intervalles formé par les tâches en chevauchement, et une approche “notBreak” axée sur la maximisation de la satisfaction des compagnies aériennes et la minimisation des déplacements d’avions.

Les expériences réalisées comparent ces différentes approches et configurations du solveur ACE aux méthodes utilisées actuellement par ADP. Les résultats démontrent que les configurations testées surpassent les performances de la solution actuellement utilisée par ADP, en particulier grâce à l’heuristique de l’ordre de valeurs **Bivs** qui montre d’excellentes performances.

Nous avons ensuite examiné diverses techniques et méthodologies conçues pour améliorer l’efficacité des solveurs de contraintes. Nous avons d’abord abordé la technique de la descente agressive de bornes (*Aggressive Bound Descent* (ABD)), une méthode qui ajuste proactivement

les contraintes d’objectifs, montrant des améliorations potentielles de performance, particulièrement lorsqu’elle est intégrée au solveur PB **Sat4j**.

Nous avons ensuite proposé une approche utilisant des encodages booléens pour les domaines des variables CSP, mettant en évidence les capacités des solveurs PB, notamment en matière d’inférence et de comptage. Cependant, son efficacité s’est avérée principalement intéressante pour certains types de contraintes seulement, laissant de la place pour des affinements supplémentaires et l’exploitation de divers encodages.

Enfin, nous avons introduit un nouveau cadre logiciel pour le développement de solveurs de contraintes parallèles et distribués. En intégrant une multitude de solveurs via l’interface **Universe**, nous avons démontré la flexibilité et la puissance du cadre proposé, suggérant son adaptabilité pour une variété d’approches parallèles et l’intégration de techniques plus sophistiquées.

L’apprentissage automatique, de son côté, offre une promesse considérable pour prédire et optimiser divers aspects des opérations aéroportuaires. Nous avons donc commencé par démontrer l’importance de prédire précisément le nombre de passagers à mobilité réduite dans les aéroports, en mettant en avant les bénéfices en termes d’efficacité opérationnelle et l’objectif global de promouvoir l’inclusivité dans le transport aérien. Face à l’afflux massif de voyageurs, les aéroports de Paris sont confrontés à la tâche cruciale d’assurer un voyage sans encombre pour les personnes à mobilité réduite. Ainsi, l’objectif principal était d’améliorer l’exactitude de ces prédictions.

En analysant les données historiques et en utilisant des techniques d’apprentissage automatique, en particulier le modèle **FastTree**, nous avons cherché à optimiser ces prédictions. Notre analyse comparative par rapport aux méthodes existantes a souligné les améliorations apportées par notre modèle d’apprentissage automatique. Notre modèle a surpassé les prédictions réalisées par les outils et méthodes existants. Ces résultats soulignent le potentiel de l’utilisation des technologies avancées et de l’apprentissage automatique pour améliorer les opérations aéroportuaires.

Nous avons ensuite abordé le problème des retards au départ des vols à l’aéroport de Paris-Charles de Gaulle. La ponctualité y est un enjeu crucial pour l’expérience passager et la gestion des coûts. Notre étude a commencé par une analyse approfondie du problème et des besoins en termes de prédictions en temps réel et de prévisions à l’aéroport de Paris-CDG. Nous avons proposé deux catégories de caractéristiques pour la prédiction des retards : les caractéristiques statiques, pour les prévisions, et les caractéristiques dynamiques, mises à jour en temps réel, pour les prédictions en temps réel.

Ensuite, nous avons construit un processus pour extraire les données nécessaires du système d’informations opérationnelles de Paris-CDG, nous permettant de créer un ensemble de données représentant une année d’activité. Nous avons conduit une étude empirique pour sélectionner les caractéristiques et les données, puis pour choisir les modèles appropriés. Malgré les défis posés par les données très variables en raison de la pandémie de COVID-19 et de ses conséquences sur le trafic aérien, les résultats montrent que certains retards peuvent être mieux prédits que le modèle de base.

L’amélioration de la précision de ces prédictions pourrait significativement progresser en explorant systématiquement d’autres modèles et leurs meilleurs hyperparamètres. L’exploitation d’informations supplémentaires, comme l’avancée du chargement des bagages, pourrait également améliorer les prédictions. Un élément crucial pour notre application reste l’explicabilité des prédictions, surtout l’identification des explications pouvant aider à la gestion des retards.

L’étude des sujets ci-dessus a permis une amélioration de la gestion opérationnelle d’ADP mais ouvre également des perspectives. Concernant la planification des parkings avions,

une adaptation des modèles proposés pour des cas spécifiques, comme celui du terminal 1 de l'aéroport CDG. Les deux problèmes de planification étudiés pourraient dans le futur être considérés comme des problèmes véritablement multi-objectifs. L'utilisation de solveurs capables de gérer de multiples critères serait alors nécessaire.

Contents

Résumé	iii
Acronyms	xiii
Aeronautical terms	xv
Notations (General)	xvii
Notations (Constraint Programming)	xix
Notations (Machine Learning)	xxi
General Introduction	1
 Part I Constraint Programming and Machine Learning in an airport context	
 Chapter 1 Constraint Programming	5
1.1 Introduction	5
1.2 Constraint network - Definitions and Notations	6
1.3 Resolution	11
1.3.1 Consistency and propagation	11
1.3.2 Resolution methods	12
1.3.3 Heuristics	13
1.3.4 Restart policies	13
1.4 Airport stand allocation problem	14
1.4.1 Introduction	14
1.4.2 Stand Allocation Formulation	17
1.5 Airport check-in desk allocation problem	20
1.5.1 Introduction	20

1.5.2	Check-in Desk Allocation Formulation	24
1.6	Conclusion	28
Chapter 2	Machine learning	29
2.1	Introduction	29
2.2	Definitions and notations	30
2.3	Explanation methods for a regression model	36
2.4	Example: How many hotdogs will be sold?	37
2.4.1	Exploratory Data Analysis (EDA)	38
2.4.2	Correlation Analysis	39
2.4.3	Experiments	40
2.4.4	Explanations	41
2.4.5	Conclusion	42
2.5	Machine Learning Usage at Airport	42
2.5.1	Forecasting and simulation	43
2.5.2	Passengers flight prediction	43
2.5.3	Flight delays	43
2.6	Machine learning usage at Paris Airports	44
2.6.1	Room	44
2.6.2	PAX	45
2.6.3	CNX PAX	45
2.6.4	Estimating the shape of PAX presentation curves	45
2.6.5	PAX presentation	46
2.7	Conclusion	47
Part II	Optimization of airport resources	49
Chapter 3	Modeling of the Airport check-in desk assignment problem	51
3.1	Introduction	51
3.2	Modeling	52
3.2.1	A first COP formulation	53
3.2.2	Some refinements	59
3.3	Experimental environment	61
3.3.1	Vocabulary	61
3.3.2	Experiments on HPC	62
3.3.3	Experiments on Paris Airport information system	62
3.3.4	Analysis and reproducibility	62

3.4	Experiments	64
3.4.1	Instances	64
3.4.2	Comparison between <code>ACE</code> and <code>ACEURANCETOURIX</code>	65
3.4.3	Modelizations and solver configurations	65
3.4.4	Scoring	67
3.4.5	No-overlapping family	68
3.4.6	Overlapping family	71
3.4.7	Overlapping with a limited number of tasks family	75
3.4.8	Comparison with the ADP algorithm	80
3.5	Conclusion	84
Chapter 4 Modeling of the Airport stand allocation problem		85
4.1	Introduction	85
4.2	Modeling of the Stand Allocation Problem	86
4.2.1	Classical variant	88
4.2.2	<code>AllDifferent</code> variant	90
4.2.3	<code>not-break</code> variant	91
4.3	Introduction	92
4.4	Modeling of the Stand Allocation Problem	92
4.4.1	Classical variant	94
4.4.2	<code>AllDifferent</code> variant	96
4.4.3	<code>not-break</code> variant	97
4.5	Experiments	98
4.5.1	Instances	98
4.5.2	<code>classical</code> vs <code>alldiff</code> modeling	98
4.5.3	<code>not-break</code> modeling	100
4.6	Conclusion	103
4.7	Experiments	103
4.7.1	Instances	103
4.7.2	<code>classical</code> vs <code>alldiff</code> modeling	104
4.7.3	<code>not-break</code> modeling	104
4.8	Conclusion	108
Chapter 5 Contributions to Resolution Methods of Constraint Programming Problems		109
5.1	Aggressive Bound Descent	110
5.1.1	Introduction	110

5.1.2	ABD Policy	111
5.1.3	Simple ABD Implementation	111
5.1.4	Experiments and Results	113
5.1.5	Conclusion	118
5.2	Pseudo-Boolean Encodings	123
5.2.1	Introduction	123
5.2.2	Preliminaries	123
5.2.3	Purely PB Encodings	124
5.2.4	Experiments and Results	127
5.2.5	Conclusion	132
5.3	Parallel solving	132
5.3.1	Related works	133
5.3.2	Architecture of our Framework	134
5.3.3	EPS approaches	136
5.3.4	Experiments and Results	140
5.3.5	Conclusion	141
5.4	Conclusion	144

Part III Predicting flow passengers and flight delays 145

Chapter 6 Predicting the Number of Passengers with Reduced Mobility on Flights 147

6.1	Introduction	147
6.2	Exploratory data analysis	148
6.3	Model definition	153
6.3.1	Basic flight features (BFF)	153
6.3.2	PRM count features	153
6.4	Model configuration	154
6.5	Correlation Analysis	154
6.6	Results	154
6.7	Conclusion	159

Chapter 7 Predicting and explaining off-block delays 161

7.1	Milestones before off-block	162
7.2	Exploratory Data Analysis - Off-block delays at CDG	163
7.3	Problem statement and objectives	165
7.3.1	Real-time off-block delay prediction	165

7.3.2	Off-block delay forecast	165
7.4	Model definition	165
7.4.1	Basic Flight Features (BFF)	165
7.4.2	Off-block Milestone Features (OMF)	166
7.4.3	Previous and Current Flights Delay Features (PCFDF)	167
7.4.4	Weather condition features (WCF)	167
7.4.5	Passenger Flow Features (PFF)	168
7.5	Data and feature extraction, preprocessing, and selection	168
7.5.1	Data extraction and preprocessing	168
7.5.2	Feature selection	168
7.6	Model selection	170
7.6.1	Gradient-boosted decision trees	171
7.6.2	Recurrent neural networks	172
7.6.3	Baseline model	172
7.7	Model evaluation	172
7.7.1	Forecast	172
7.7.2	Real-time predictions	175
7.8	Explaining off-block delay predictions	181
7.9	Deployment	184
7.10	Conclusions	184
	General Conclusion and Perspectives	187
	Index	
	Bibliography	195

Acronyms

ABD *Aggressive Bound Descent.* [iv](#), [xi](#), [109](#), [118](#), [119](#), [144](#), [188](#)

AC *Arc Consistency.* [xi](#), [11](#), [12](#)

AIBT *Actual In-Block Time.* [xi](#), [162](#)

AOBT *Actual Off-Block Time.* [xi](#), [162](#), [165](#), [190](#)

AODB *Airport Operational Database.* [xi](#), [44](#)

AOP *Airport Operation Plan.* [xi](#), [44](#)

ATC *Air Traffic Control.* [xi](#), [xv](#)

BIP *Binary Integer Programming.* [xi](#), [17](#)

BRS *Baggage Reconciliation System.* [xi](#), [44](#)

CDAP *Check-in Desk Allocation Problem.* [xi](#), [14](#), [20](#), [52](#), [129](#), [187](#), [189](#)

CLI *Command Line Interface.* [xi](#), [63](#)

CN *Constraint Network.* [xi](#), [5](#), [10](#)

CNF *Conjunctive Normal Form.* [xi](#), [123](#)

CNO *Constraint Network under Optimization.* [xi](#), [11](#), [52](#), [111](#)

COP *Constraint Optimization Problem.* [iv](#), [xi](#), [11](#), [12](#), [51](#), [52](#), [123](#)

CP *Constraint Programming.* [iii](#), [xi](#), [1](#), [2](#), [5](#), [63](#), [187](#)

CSP *Constraint Satisfaction Problem.* [xi](#), [5](#), [10](#), [11](#), [12](#), [123](#)

CTOT *Calculated Take-Off Time.* [xi](#), [xv](#), [162](#)

EIBT *Estimated In-Block Time.* [xi](#), [162](#)

EOBT *Estimated Off-Block Time.* [xi](#)

EPS *Embarrassingly Parallel Search.* [xi](#), [132](#), [133](#), [189](#)

GAP *Gates Allocation Problem.* [xi](#), [14](#)

- GBDT** *Gradient Boosting Decision Tree*. [xi](#), [34](#), [171](#)
- HPC** *High-Performance-Computing*. [viii](#), [xi](#), [51](#), [62](#)
- IATA** *International Air Transport Association*. [xi](#), [148](#)
- IP** *Integer Linear Programming*. [xi](#), [17](#)
- LIME** *Local Interpretable Model-agnostic Explanations*. [xi](#), [36](#), [37](#)
- LSTM** *Long Short Time Memory*. [xi](#), [44](#), [172](#)
- MAC** *Maintaining Arc Consistency*. [xi](#), [12](#)
- MAE** *Mean Absolute Error*. [xi](#), [35](#), [40](#), [175](#)
- MILP** *Mixed Integer Linear Programming*. [xi](#)
- ML** *Machine Learning*. [xi](#), [29](#), [36](#), [44](#), [47](#), [187](#)
- MSE** *Mean Squared Error*. [xi](#), [35](#), [40](#)
- PAX** *Passenger*. [xi](#), [43](#), [45](#)
- PB** *Pseudo-Boolean*. [xi](#), [63](#), [109](#)
- PRM** *Passengers with Reduced Mobility*. [xi](#), [2](#), [43](#)
- QBF** *Quantified Boolean Formula*. [xi](#), [63](#)
- RMS** *Resource Management System*. [xi](#)
- RMSE** *Root Mean Squared Error*. [xi](#), [xxi](#), [35](#), [40](#), [44](#), [175](#)
- RNN** *Recurrent Neural Network*. [xi](#), [44](#), [172](#)
- SAP** *Stand Allocation Problem*. [xi](#), [14](#), [86](#), [92](#), [187](#), [188](#)
- SAT** *SATisfiability Problem*. [xi](#), [63](#)
- SHAP** *SHapley Additive exPlanations*. [xi](#), [37](#), [41](#)
- SIBT** *Scheduled In-Block Time*. [xi](#)
- SOBT** *Scheduled Off-Block Time*. [xi](#), [153](#), [162](#), [165](#), [166](#), [168](#), [170](#), [190](#)
- STTT** *Scheduled Turn-round Time*. [xi](#)
- TOBT** *Target Off-Block Time*. [xi](#), [xv](#), [162](#)
- TSAT** *Target Start-Up Approval Time*. [xi](#), [162](#), [166](#), [168](#), [170](#)
- XAI** *eXplainable AI*. [xi](#), [36](#)

Aeronautical terms

AIBT The time that an aircraft arrives in-blocks.. [xi](#)

AOBT Time the aircraft pushes back / vacates the parking position.. [xi](#)

ATC Service provided by ground-based controllers who direct aircraft on the ground and in the air.. [xi](#)

CNX PAX The number of PAX in transit (connexion).. [xi](#), [45](#)

CTOT A time calculated and issued by the appropriate Central Management unit, as a result of tactical slot allocation, at which a flight is expected to become airborne.. [xi](#)

EIBT The estimated time that an aircraft will arrive in-blocks.. [xi](#)

EOBT The estimated time at which the aircraft will start movement associated with departure.. [xi](#)

Registration A registration represents a flight or a set of flights.. [xi](#)

SIBT The time that an aircraft is scheduled to arrive at its parking position.. [xi](#)

SOBT The time that an aircraft is scheduled to depart from its parking position.. [xi](#)

STTT which corresponds to $SOBT - SIBT$. [xi](#)

TOBT The time that an Aircraft Operator or Ground Handler estimates that an aircraft will be ready, all doors closed, boarding bridge removed, push back vehicle available and ready to start up / push back immediately.. [xi](#)

TSAT The time provided by [ATC](#) taking into account [TOBT](#), [CTOT](#) and/or the traffic situation that an aircraft can expect start-up / push back approval.. [xi](#)

Notations (General)

$\mathbb{B}$ set of Booleans. [xi](#)

$\mathbb{N}$ set of natural integers. [xi](#)

$\mathbb{R}$ set of reals. [xi](#), [30](#), [31](#)

$\mathbb{Z}$ set of integers. [xi](#), [30](#)

$\prod_{i=1}^r S_i$ cartesian product which corresponding to $S_1 \times S_2 \times \dots \times S_r$. [xi](#), [7](#)

Notations (Constraint Programming)

$I[\mathcal{X}]$ restriction of I on $\mathcal{X}$. [xi, 10](#)

I instantiation. [xi, 10](#)

$\mathcal{C} = \text{ctrs}(\mathcal{P})$ set of constraints of constraint network $\mathcal{P}$. [xi, 10, 11](#)

$\mathcal{O} = \text{obj}(\mathcal{X})$ objectif of constraint network $\mathcal{P}$. [xi, 11](#)

$\mathcal{P}$ constraint network. [xi, 10, 11, 12, 111](#)

$\mathcal{S} = \text{sols}(\mathcal{P})$ set of solutions of constraint network $\mathcal{P}$. [xi, 10](#)

$\mathcal{X} = \text{vars}(\mathcal{P})$ set of variables of constraint network $\mathcal{P}$. [xi, 10, 11](#)

$\text{dom}(x)$ current domain of a variable x . [xi, 6](#)

$\text{dom}^{\text{init}}(x)$ initial domain of a variable x . [xi, 6](#)

$\text{fut}(\mathcal{X})$ set of futur variables (not-fixed) of set $\mathcal{X}$ of variables. [xi](#)

$\text{rel}(c)$ relation of constraint c . [xi, 8](#)

$\text{scp}(c)$ variable of the constraint c . [xi, 8](#)

$\tau[i]$ i th value of tuple τ . [xi, 7](#)

$\tau[x]$ value of x from tuple τ . [xi](#)

τ tuple of values. [xi, 7](#)

c constraint. [xi, 8](#)

Notations (Machine Learning)

R^2 Coefficient of determination.. [xi](#), [35](#)

E Set of examples.. [xi](#), [30](#), [34](#), [35](#)

$\text{RMSE}(E)$ [Root Mean Squared Error](#) (RMSE). [xi](#), [35](#)

$\mathcal{F}$ a forest of *Decision Tree*.. [xi](#), [33](#)

χ an instance or an observation. [xi](#), [30](#)

$\mathcal{T}$ a decision tree.. [xi](#), [31](#)

$f : E \mapsto Y$ A decision function that maps the set of examples E to Y .. [xi](#), [30](#)

General Introduction

The aviation industry plays a vital role in shaping our modern world, providing a means for millions of people to explore new destinations, conduct business, reunite with loved ones, and experience different cultures.

Airports are essential nodes in the worldwide transportation matrix, offering the infrastructure for passenger mobility and cargo logistics. Paris Charles de Gaulle Airport (CDG) is a prime example of operational complexity among global airports. As noted in the Paris Airports 2019 Annual Report⁵, CDG managed approximately 76 million passengers (approximately 31 million for Paris Orly), oversaw 498,000 takeoffs and landings (218,349 for Paris Orly), and connected to 328 different destinations in 119 countries (128 in 48 countries for Paris Orly). CDG has 3 large terminals (1 single terminal with 4 check-in areas for Orly) and 4 runways (3 runways for Orly). This makes it the second busiest airport in Europe and the ninth busiest in the world regarding passenger traffic. As the primary hub for Air France, it is also a major international gateway for the European Union.

Interestingly, despite its scale and traffic, there are no immediate plans for constructing new terminals at CDG. This imposes an additional layer of complexity for managing growing passenger numbers and resources, making optimization more crucial than ever. Furthermore, although the COVID-19 pandemic led to a significant downturn in airport traffic, recent data indicate a recovery in line with pre-crisis trends. Some airlines have resumed flying super jumbo aircraft to absorb the upturn in traffic, as reported by Le Monde⁶. This resurgence underscores the urgency to address the challenges of resource management and passenger flow to ensure operational efficiency and resilience against future disruptions.

Facing the intricacies of airport operations, especially at mega hubs like CDG, calls for innovative solutions. Traditional methods, based on technologies several decades old, may require greater adaptability to handling today's challenges, especially with the volatile nature of the aviation industry. With the evolving landscape of computational techniques, two avenues offer promise. The first leverages advancements in constraint programming, which can efficiently handle problems with large variables and constraints. The second harnesses the predictive power of machine learning to optimize specific parameters vital for efficient operations.

In order to respond to the problem and the proposed challenges, this manuscript proposes a plan of three parts divided into seven chapters. The first part of this manuscript presents the state-of-the-art on *Constraint Programming* (CP) paradigm and the two main problems studied: the allocation of registration counters and the allocation of aircraft stands. The second part focuses on machine learning techniques and methodology in this manuscript.

⁵<https://www.parisaeroport.fr/docs/default-source/groupe-fichiers/finance/relations-investisseurs/information-financi%C3%A8re/rapports-annuels/report-on-activity-and-sustainable-development-2019.pdf>

⁶https://www.lemonde.fr/economie/article/2023/06/08/1-a380-s-impose-comme-l-avion-de-la-reprise-du-traffic-aerien_6176792_3234.html

The second part comprises 3 chapters describing proposed contributions to the field of CP. The third chapter describes different modeling proposals for the check-in desk allocation problem [FALM23] and allows us to present a new pragmatic constraint that improves the resolution performance. Experiments will be carried out in an ideal computing environment and then in the real environment of Paris Airports. This first experimental chapter allows us to present our tool for extracting and analyzing the results of experimental campaigns: Metrics [FWW20, FWW21]. The fourth chapter focuses on the problem of aircraft stand allocation. Similar to Chapter 3, it presents different modeling adapted to this problem. The experiments confirm the contribution of our approach to the method currently used by Paris Airports [FLMT22, FMLT22]. The last chapter of Part 2 presents various generic resolution methods that can be used to solve the Paris Airports allocation problems and a set of instances from the XCSP problem library. These works have been published nationally [FLMW21, FW22a] and internationally [FLMW22, FW22b].

The third part of this thesis delves into two chapters focused on machine learning techniques for optimizing specific aspects of airport operations. The first chapter of this part focuses on the prediction of **Passengers with Reduced Mobility**. Given the significance of ensuring a seamless airport experience for PRM, an accurate forecast of their number becomes crucial. By leveraging historical data and advanced machine learning algorithms, this chapter provides a robust model for predicting the numbers of PRMs with heightened precision. The subsequent chapter shifts its lens to the prediction of off-block delays. Delays in off-block times can lead to cascading effects on flight schedules, making their timely prediction essential. Here, machine learning comes into play, offering the potential to preemptively identify and manage potential delays, ensuring timely departures and upholding the operational integrity of the airport. Together, these chapters underscore the transformative potential of machine learning in reimagining and refining airport operations [FMT23a, FMT23b].

In the constantly evolving landscape of the aviation industry, optimizing operations, especially in mega hubs like CDG, has implications for efficiency and the economic, environmental, and socio-cultural fabric of our globalized world. The forthcoming chapters delve deeper into the challenges and the proposed innovative solutions. This manuscript aims to chart a path toward a more resilient and efficient future for airport operations by using the powers of constraint programming and machine learning.

Part I

Constraint Programming and Machine Learning in an airport context

Chapter 1

Constraint Programming

Contents

1.1 Introduction	5
1.2 Constraint network - Definitions and Notations	6
1.3 Resolution	11
1.3.1 Consistency and propagation	11
1.3.2 Resolution methods	12
1.3.3 Heuristics	13
1.3.4 Restart policies	13
1.4 Airport stand allocation problem	14
1.4.1 Introduction	14
1.4.2 Stand Allocation Formulation	17
1.5 Airport check-in desk allocation problem	20
1.5.1 Introduction	20
1.5.2 Check-in Desk Allocation Formulation	24
1.6 Conclusion	28

1.1 Introduction

Constraint Programming (CP) [Apt03, RvW06, Lec09] is a powerful and recognized paradigm for modeling and solving everyday problems (for example, the time-schedule problem can be modeled using constraint programming), as well as combinatorial problems ranging from configuration and planning to bioinformatics. CP offers generic methods for modeling and solving this type of problem. It aims to reduce modeling complexity by being as close to the natural language description of the problem as possible.

In the CP approach, users define the problem by specifying decision variables and constraints that define the relationships between these variables. The objective is to find an assignment for all variables that satisfies all the given constraints. This is known as a *Constraint Satisfaction Problem* (CSP). The task is then solved by employing specialized solvers, which use generic methods to efficiently explore the search space and find solutions that satisfy all the constraints.

This chapter first introduces the definitions and notations related to constraint networks (CN). Then, we explore the extension of constraint networks to optimization problems, where

the goal is not only to find any solution but to find the best solution according to specific optimization criteria.

Since this manuscript focuses on combinatorial problems in an airport context, we present two primary problems of interest:

- *Assigning Flights to Parking*: This problem involves efficiently assigning flights to available parking slots at the airport, considering various constraints and optimization objectives.
- *Check-in Desks Assignment*: The problem of assigning check-in desks to different flights, considering factors like flight schedules and desk capacities.

Constraint programming is an essential tool for handling complex combinatorial problems in airport operations. Its ability to model problems naturally allows for more intuitive problem-solving approaches.

1.2 Constraint network - Definitions and Notations

Constraint programming was introduced and formalized in the 1960s and 1970s [Mon74]. This section introduces the concepts usually used to describe a constraint network [Lec09].

Definition 1 (Variable)

A variable x is an object that can take on different values. This value belongs to a finite set of values called the *current domain* of x and is denoted $\text{dom}(x)$.

The domain of variable x may evolve, but it is always included in a set called the initial domain of x , denoted by $\text{dom}^{\text{init}}(x)$. This manuscript considers only *discrete variables* and *finite domains*.

Example 1

Let the variables x and y have the current domain $\text{dom}(x) = \{a, b\}$ and $\text{dom}(y) = \{b, c\}$. Their initial domain may be $\text{dom}^{\text{init}}(x) = \text{dom}^{\text{init}}(y) = \{a, b, c\}$

A variable is *fixed* when its current domain contains one value and *unfixed* otherwise. A variable can be fixed explicitly. In this case, the variable is fixed on a given value a from its current domain $\text{dom}(x)$ during the execution of an algorithm, every other value $b \neq a$ is considered to be removed from $\text{dom}(x)$. Assigning a value to a variable is called a *variable assignment*. We say that the value a is assigned to x . A variable can also be fixed implicitly when deduction mechanisms are used.

Example 2

Let two variables x and y with the common initial domain $\text{dom}^{\text{init}}(x) = \text{dom}^{\text{init}}(y) = \{1, 2\}$ and the equality on this variables $x = y$. If the variable x is assigned to 1, by reasoning from the equality we can deduce that y must also be equal to 1. The value 2 can be removed from $\text{dom}(y)$ by deduction. The two variables are then fixed, the first one explicitly and the second one implicitly.

The notions of *tuple*, *Cartesian product*, and *relation* are necessary to define constraints.

Definition 2 (Tuple)

A *tuple* τ is a sequence of values. A tuple containing r values is called an r -tuple. The i th value of an r -tuple (with $1 \leq i \leq r$), is denoted by $\tau[i]$.

Definition 3 (Cartesian Product)

Let $S_1, S_2, \dots, S_r$ be a sequence of r sets. The Cartesian product $S_1 \times S_2 \times \dots \times S_r = \prod_{i=1}^r S_i$ is the set of r -tuple $\{(a_1, a_2, \dots, a_r) \mid a_1 \in S_1, a_2 \in S_2, \dots, a_r \in S_r\}$.

Example 3

Let 3 variables x, y, z such that $\text{dom}(x) = \text{dom}(y) = \{a, b\}$ and $\text{dom}(z) = \{a, c\}$. The Cartesian product of the domains of these variables are defined by :

$$\text{dom}(x) \times \text{dom}(y) \times \text{dom}(z) = \left\{ \begin{array}{l} (a, a, a), \\ (a, a, c), \\ (a, b, a), \\ (a, b, c), \\ (b, a, a), \\ (b, a, c), \\ (b, b, a), \\ (b, b, c) \end{array} \right\}$$

Definition 4 (Relation)

A *relation* R defined over a sequence of r sets $S_1, S_2, \dots, S_r$ is a subset of the Cartesian product $\prod_{i=1}^r S_i$, so $R \subseteq \prod_{i=1}^r S_i$.

Example 4

For example a relation R_{xyz} defined on $\text{dom}(x) \times \text{dom}(y) \times \text{dom}(z)$ can be :

$$R_{xyz} = \left\{ \begin{array}{l} (a, a, a), \\ (a, a, c), \\ (a, b, c), \\ (b, a, a), \\ (b, a, c), \end{array} \right\} \subset \left\{ \begin{array}{l} (a, a, a), \\ (a, a, c), \\ (a, b, a), \\ (a, b, c), \\ (b, a, a), \\ (b, a, c), \\ (b, b, a), \\ (b, b, c) \end{array} \right\}$$

Definition 5 (Constraint)

A *constraint* c is defined over a set of variables, called the *scope* of c (denoted by $\text{scp}(c)$) and by a *relation* (denoted by $\text{rel}(c)$). This relation describes exactly the set of tuples allowed by c for the variable of its scope. We have:

$$\text{rel}(c) \subseteq \prod_{x \in \text{scp}(c)} \text{dom}^{\text{init}}(x).$$

Definition 6 (Arity)

The *arity* of a constraint c is the number of variables involved in c ($|\text{scp}(c)|$). A constraint is:

- *unary* iff its arity is 1;
- *binary* iff its arity is 2;
- *ternary* iff its arity is 3;
- *n-ary* otherwise;

Definition 7 (Intensional Constraint)

A constraint c is *intensional*, or defined in *intension* if it is described by a Boolean expression or formula.

$$f : \prod_{x \in \text{scp}(c)} \text{dom}(x) \rightarrow \mathbb{B}$$

Example 5

Let the variables v, w, x, y et z with domain $\text{dom}(v) = \text{dom}(w) = \text{dom}(x) = \text{dom}(y) = \text{dom}(z) = \{0, 1, 2\}$:

- $v = w \times 2$ is a binary constraint in intension, its scope is $\{v, w\}$;
- $(x \neq y) \wedge (y \neq z) \wedge (x \neq z)$ is a ternary constraint in intension, its scope is $\{x, y, z\}$.

Definition 8 (Extensional Constraint)

A constraint c is *extensional*, or defined in extension *iff* $\text{rel}(c)$ describes explicitly:

- *positively* by listing the tuples *allowed* by c ;
- *negatively* by listing the tuples *disallowed* by c .

Example 6

If $\text{dom}(x) \times \text{dom}(y) \times \text{dom}(z) = \{0, 1, 2\} \times \{0, 1, 2\} \times \{0, 1, 2\}$ then the ternary constraint from the example 5 can be defined postively as follows:

$$(x, y, z) \in \{(0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)\}$$

and negatively as follows:

$$(x, y, z) \notin \{(0, 0, 0), (0, 0, 1), (0, 0, 2), \dots, (2, 2, 2)\}$$

Definition 9 (Global Constraint)

A *global constraint* is a constraint model that captures a specific semantic relationship and can be applied to an arbitrary number of variables.

Example 7

AllDifferent [Rég94, van01, GMN08] forces all values of specific variables to differ. We can rewrite the second constraint from the example 5 as follows:

$$\text{allDifferent}(x, y, z)$$

Global constraints make modeling more effortless and efficient by providing dedicated filtering algorithms. There are currently over 400 global constraints [VHK06, BCDP07]. Nevertheless, modeling languages offer a core set of global constraints sufficient to model most problems [BLAP20].

Finally, a structure composed of variables and constraints is called a *constraint network*.

Definition 10 (Constraint Satisfaction Problem)

An instance $\mathcal{P}$ of a *Constraint Satisfaction Problem* (CSP), also called *Constraint Network* (CN) is defined by:

- a *finite set of variables*, denoted by $\mathcal{X} = \text{vars}(\mathcal{P})$;
- a *finite set of constraints*, denoted by $\mathcal{C} = \text{ctrs}(\mathcal{P})$, each covering a sub-set of $\mathcal{X}$ such as $\forall c \in \mathcal{C}, \text{scp}(c) \subseteq \mathcal{X}$.

Definition 11 (Support)

For a constraint c :

- an *allowed tuple* by c is an element of $\text{rel}(c)$;
- a *valid tuple* is an element of $V = \prod_{x \in \text{scp}(c)} \text{dom}(x)$;
- a *support* (over c) is a tuple that is both allowed and valid.

Definition 12 (Instantiation)

An *instantiation* I of set of variables $\mathcal{X} = \{x_1, \dots, x_n\}$ of n variables is a set of tuples $(x_1, v_1), \dots, (x_n, v_n)$ such as $\forall i (1 \leq i \leq n), v_i \in \text{dom}^{\text{init}}(x_i)$ and any variable x_i appears only once. An instantiation is *valid* iff $\forall (x, v) \in I, v \in \text{dom}(x)$.

Notation 1 (Restriction of Instantiation)

Let I be an instantiation and $\mathcal{X}$ be a set of variables, we note $I[\mathcal{X}]$ an instantiation of $\text{vars}(I) \cap \mathcal{X}$ defined as $\{(x, a) \in I \mid x \in \mathcal{X}\}$. We call this instantiation the restriction of I over $\mathcal{X}$

Definition 13 (Satisfying Instantiation)

Let I be an instantiation and c a constraint, I satisfying c iff:

- c is covered by I , i.e., $\text{scp}(c) \subseteq \text{vars}(I)$;
- the set of tuples $I[\text{scp}(c)] = \{(x_1, v_1), \dots, (x_r, v_r)\}$ is allowed by c .

Definition 14 (Solution of CSP)

A *solution* of an instance CSP $\mathcal{P}$ is an instantiation of $\mathcal{X}$ such that all its constraints are satisfied. The set of all the solutions is denoted by $\mathcal{S} = \text{sols}(\mathcal{P})$

The classic constraint satisfaction problem consists of determining whether a given constraint network is satisfiable by showing a solution if it is. Deciding whether a CSP is satisfiable is an NP-complete problem [Mac77a]. Other combinatorial tasks may interest: enumerating or counting the set of solutions, calculating an optimal solution according to a given objective, etc. We conclude this section by defining one such task: the constraint optimization problem.

Definition 15 (Constraint Optimization Problem)

An instance $\mathcal{P}$ of a *Constraint Optimization Problem* (COP), also called *Constraint Network under Optimization* (CNO) is defined by:

- a *finite set of variables*, denoted by $\mathcal{X} = \text{vars}(\mathcal{P})$;
- a *finite set of constraints*, denoted by $\mathcal{C} = \text{ctrs}(\mathcal{P})$, each covering a sub-set of $\mathcal{X}$ such as $\forall c \in \mathcal{C}, \text{scp}(c) \subseteq \mathcal{X}$.
- an *objective function* $\mathcal{O} = \text{obj}(\mathcal{X})$ to be maximized or minimized.

A COP instance can be interpreted as a CSP instance with an associated function. This function gives a numerical value to each instance's solutions, thereby quantifying its quality. The objective is to find the solution that maximizes or minimizes this function. For example, a function that maximizes (resp. minimizes) the sum or product of certain variables (possibly associated with coefficients) of the problem.

1.3 Resolution

1.3.1 Consistency and propagation

Different consistency operators, denoted ϕ , provide different levels of inference. Consequently, for a specific constraint network $\mathcal{P}$, achieving different degrees of consistency is possible, leading to more or less extensive filtering of the search space [Mac77b].

Based on the definition of a support (definition 11), the **arc-consistency** is defined as follows:

Definition 16 (Constraint arc-consistency)

A constraint c is said to be **arc-consistent** (AC) if and only if $\forall x \in \text{scp}(c), \forall a \in \text{dom}(x)$, there exists a support on c for (x, a) , i.e., a support τ on c such that $\tau[x] = a$.

Definition 17 (Value arc-inconsistent)

A value (x, a) is **arc-inconsistent** for a constraint c when it has no support on c .

Definition 18 (Algorithm AC)

An **AC** algorithm, for a constraint c , is an algorithm that removes all values that are **arc-inconsistent** on c ; this algorithm is said to force **arc-consistent** on c .

We can now define the **arc-consistency** for the constraint network level.

Definition 19 (Constraint Network AC)

A constraint network $\mathcal{P}$ is **AC** if and only if each constraint of $\mathcal{P}$ is **AC**.

Definition 20 (AC-closure)

Calculating the **AC-closure** on a constraint network $\mathcal{P}$ is the act of removing all the **arc-inconsistent** values of $\mathcal{P}$.

The operator that computes the **AC-closure** of a constraint network is called an **operator of filtering**. Various operators have been proposed in the literature, for instance, **AC3** [Mac77b], **AC2011** [BRYZ05], etc.

1.3.2 Resolution methods

For a constraint network $\mathcal{P}$, utilizing the *generate-and-test* approach — which involves creating a set of full instantiations and linearly checking each — results in an exponential time complexity. This level of complexity is impractical for real-world applications. This section demonstrates how to address this issue by integrating the previous propagation method with a backtracking algorithm. This combination offers a more efficient search space traversal, significantly reducing the time cost.

Backtracking search is a conventional method for addressing **COP/CSP** instances, functioning as a *complete procedure*. It conducts a *depth-first exploration* of the search tree, facilitated by a backtracking mechanism alongside a sequence of decisions and propagations. Contrarily, there are *incomplete search* techniques. While these do not ensure algorithmic completeness, they can be more effective in locating solutions.

Typically, as in *Maintaining Arc Consistency (MAC)* [SF94] that propagates constraints by maintaining the property of **arc consistency**, a binary search tree $\mathcal{T}$ is built: at each internal node of $\mathcal{T}$, (i) a pair (x, v) is selected where x is an unfixed variable, and v is a value in $\text{dom}(x)$, and (ii) two cases (branches) are considered, corresponding to the assignment $x = v$ and the refutation $x \neq v$.

Backtrack search for **COP** relies on **CSP** solving: the principle is to add a special *objective constraint* $\text{obj} < \infty$ to the constraint network (although it is initially trivially satisfied), and to update the limit of this constraint whenever a new solution is found. It means that whenever a solution S is found with cost $B = \text{obj}(S)$, the objective constraint becomes $\text{obj} < B$. Hence, a sequence of better and better solutions is generated (*SATisfiability* is systematically proved with respect to the current limit of the objective constraint) until no more exists (*UNSATisfiability* is eventually proved with respect to the limit imposed by the last found solution), guaranteeing that the last found solution is *optimal*.

1.3.3 Heuristics

1.3.3.1 Variable ordering heuristic

The order in which variables are chosen during the depth-first traversal of the search space is decided by a **variable ordering heuristic**. Each heuristic associates to a variable a score computing *statically*, *dynamically* or *adaptatively*.

Heuristics static : Static heuristics assign a score to variables at the onset of the search. We list a few prominent static heuristics below:

- **rand**: This heuristic randomly selects a variable for branching.
- **lex**: This heuristic opts for the variable that minimizes its lexicographical rank. It follows the variable order as presented in the model. This approach can be valuable if the order in the modeling phase is meticulously chosen.
- **deg**: This heuristic picks the variable with the maximum score determined by its degree, which is the count of constraints in which the variable is involved.

Heuristics dynamic : Dynamic heuristics adapt to the evolving nature of the constraint network. With each search node, changes in variable assignments and domain filtering constantly alter the network’s state. Given this fluidity, it is logical for the heuristic to reassess the score of each variable in response to every network state shift. An example of such a heuristic is **dom**, which selects the variable based on the minimization of its current domain score.

Heuristics adaptive : Finally, there are adaptive heuristics which, in addition to using the current state of the variables and the constraint network, also take into account the history of the constraint network. A classical heuristic is **dom/wdeg** [BHLS04] that aggregates by a division operator the **dom** heuristics with a dynamic degree of the variable **wdeg**. In this manuscript, we also use a recent heuristics called **Frba/dom** [LYL21a] based on the **fail first principle** and exploited two aspects of failure information collected during the search: the failure proportion after the propagations of assignments of variables and the failure length heuristics consider the length of failures, which is the number of fixed variables composing a failure.

1.3.3.2 Value ordering heuristic

Similar to variable selection heuristics, a **value ordering heuristic** is essential to determine the subsequent value to employ. A straightforward heuristic approach involves utilizing the initial value in the domain, symbolized by **First**. This method is frequently adopted due to its robustness. Recently, a heuristic strategy suggesting using the value exerting the most significant effect on the objective function was introduced [FP17], labeled as **Bivs**. For COPs, prioritizing the value observed in the most recent solution is often advantageous. This tactic, known as **solution saving**, is cited in works like [VP17, DCS18].

1.3.4 Restart policies

Restart policies play an important role in modern constraint solvers as they permit to address the heavy-tailed runtime distributions of SAT (Satisfiability Testing) and CSP/COP instances [GSCK00]. In essence, a restart policy corresponds to a function **restart** : $\mathbb{N}^+ \rightarrow \mathbb{N}^+$, that indicates the maximal number of “steps” allowed for the search algorithm at the attempt, called

run. It means that *backtrack search* piloted by a restart policy, builds a sequence of binary search trees $\langle \mathcal{T}_1, \mathcal{T}_2, \dots \rangle$, where $\mathcal{T}_j$ is the search tree explored at run j .

Note that the *cutoff*, which is the maximum number of allowed steps during a run, may correspond to the number of backtracks, the number of wrong decisions [BZF04], or any other relevant measure. In a *fixed* cutoff restart strategy, **restart**(j) is constant whatever is the run j . In a *dynamic* cutoff restart strategy, **restart** increases the cutoff geometrically [Wal99], which guarantees that the whole space of partial solutions will be explored.

1.4 Airport stand allocation problem

In the following sections, we present two issues studied in this manuscript: *Stand Allocation Problem* (SAP) and *Check-in Desk Allocation Problem* (CDAP).

1.4.1 Introduction

At airports, one of the significant combinatorial problems that need to be solved is the **Stand Allocation Problem**. This problem is closely related to the *Gates Allocation Problem* (GAP), and both have been extensively studied since the 1980s [MM85, DS91, DJP08, Sim07].

In the following, we will refer to both problems without distinction. The main objective of the **Stand Allocation Problem** is to find an optimal assignment of aircraft serving different flights to the available stands (*gates*) at the airport. Each flight requires a specific *stand* for various tasks, such as passenger boarding, baggage handling, and refueling.

Airport Term 1 (Stand)

A *stand* is an *aircraft position*. There are two types of possible stands:

- *contact stands* (or *hard stand*) connected to a terminal by a door and a gateway (also called bridge or jetway);
- *remote stands* where a bus is required to reach the terminal.

Notation 2 (Stand)

A stand is noted p . The set of all stands is noted PK .

Figure 1.1 illustrates the difference between contact (underline $G\mathcal{J}$) and remote stands (over line $G\mathcal{J}$).

Airport Term 2 (Flight turnaround (or rotation))

A *flight turnaround* (or *rotation*) comprises at least one arrival or departure flight or both.

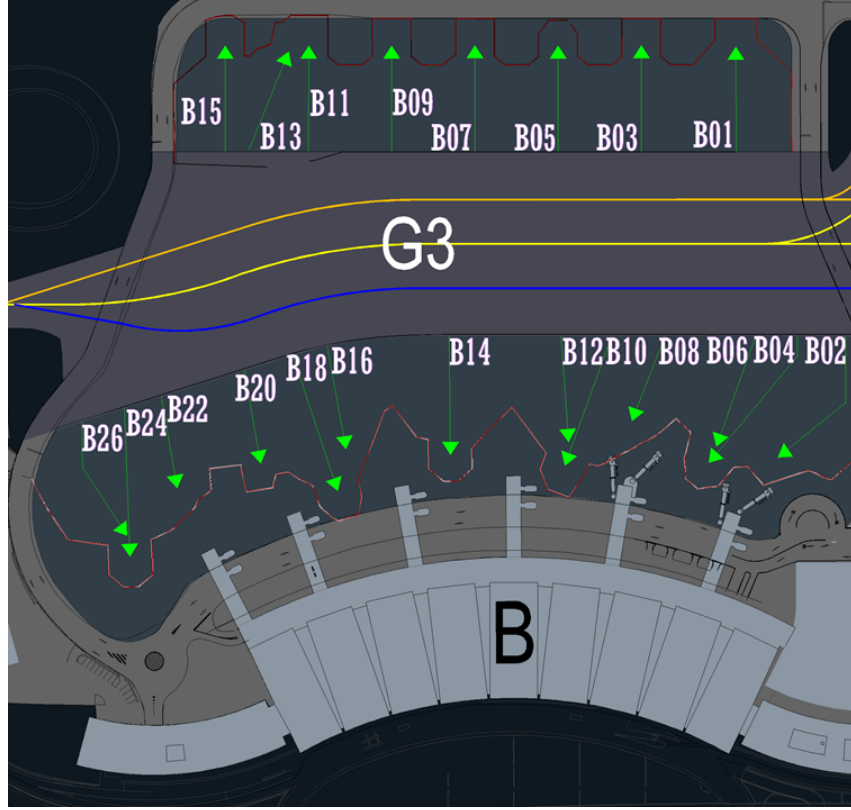


Figure 1.1: Contact and remote parking.

Notation 3 (Rotation)

We note a rotation: ϕ .

The set of all rotations is noted: Φ .

We note a_ϕ and d_ϕ the rotation's start and end time (i.e., the arrival time of the flight at the airport and the departure time of the flight from the airport).

Airport Term 3 (Stand operations)

The *stand operations* of a flight turnaround can be divided into three parts:

- operations about the *arrival flight* composed of the unboarding of passengers and luggages;
- waiting time;
- operations of the *departure flight* composed of the boarding of passengers and luggages.

Note that during these operations, we also have aircraft ground handling operations, such as catering, refueling, cabin services, etc.

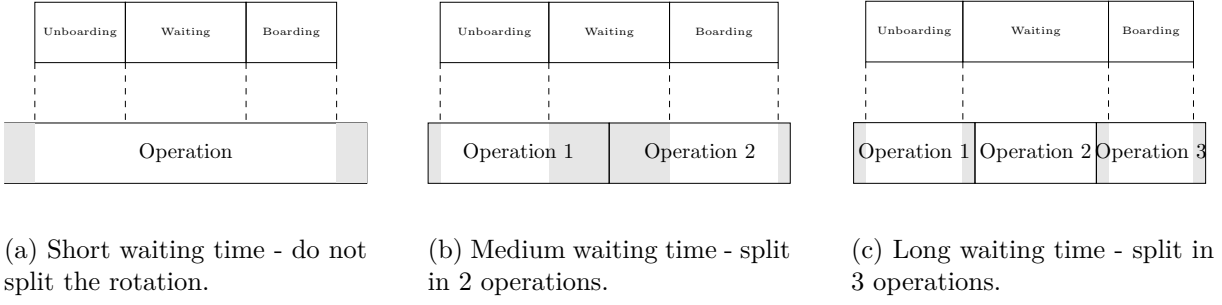


Figure 1.2: Number of operations depending on waiting time.

Notation 4 (Operation)

An operation, denoted as i , is characterized by its arrival at a_i and departure at d_i at the stand for operation i .

Each operation corresponds to an aircraft of a specific type, such as an A320 or B757. The type of aircraft associated with operation i is denoted as k_i .

When referring to the i th operation within rotation ϕ , we use the notation ϕ_i .

The complete collection of flight operations is symbolized as $\mathcal{T}$.

For a specific rotation, ϕ , the corresponding *set of operations* (also termed the *set of tasks*) is represented as $\mathcal{T}_\phi$.

Depending on the waiting time and for operational reasons, the aircraft may be moved to another stand, creating several flight operations. Figures 1.2 illustrate the possible cases. This movement requires a towing tractor (see Figure 1.3) and involves a cost for the operator. Figure 1.2a presents the case where the arrival and the departure flight are the same operations. Figures 1.2b and 1.2c show that the waiting time is enough to consider a decomposition on two or three operations, respectively.



Figure 1.3: A towing tractor.

Of course, stand assignments must be adapted to the airport's services and be practical for passengers. A solution must satisfy a set of strict rules [DDNP07]:

1. one stand is assigned to at most one flight at the same time,
2. space restrictions concerning adjacent stands must be respected,

3. the affinity of specific aircraft for particular aircraft positions must be respected.

We found different single objectives for this problem [DDNP07, BGSS14, DGS20]:

1. minimizing passenger walking distances using *Binary Integer Programming* (BIP) [Bih90, YSC02] or stochastic model [YT07],
2. minimizing the number of off-gate event [VB88],

Or multi-objectives:

1. [LRZ05] use *Integer Linear Programming* (IP) to optimize two objectives:
 - minimizing the sum of the delay penalties,
 - minimizing the total walking distance,
2. [PKB14] uses *binary integer programming* to optimize three objectives:
 - maximizing the gate rest time between two turns,
 - minimizing the cost of towing an aircraft with a long turn,
 - minimizing overall costs, including penalization for not assigning preferred gates to certain turns.
3. [DJP12] use a *Clique Partitionning formulation* for optimize four objectives:
 - maximizing the total assignment preference score,
 - minimizing the number of unassigned flights,
 - minimizing the number of tows,
 - maximizing the robustness of the resulting schedule.

1.4.2 Stand Allocation Formulation

In this section, we consider the formulation of the **gate allocation problem** proposed by [DJP08, GABG15] which we will adapt for a **stand allocation problem** (without any loss of generality) and to the case of Paris Airports.

The previous section explains that the rotation can be decomposed into several operations. At Paris Airports, there can only be two movements for the same aircraft, i.e., a maximum of 3 parking positions, and the conditions for determining whether a rotation must be decomposed depend on certain processing times:

- T_ϕ : minimum transit time; if the duration $d_\phi - a_\phi$ of the rotation (i.e., between the two flights concerning the plane) is less than this time, the plane cannot be moved (towed) to remote parking;
- T_{hang} : minimum time of departure from the hangar; if the rotation is simply a flight arrival, this is the minimum occupation time at the first parking of the rotation
- T_{runway} : minimum time on the runway; if the rotation is simply a flight departure, this is the minimum occupation time at the last parking of the rotation
- T_{arr} : minimum time to process the arrival; this is the minimum occupation time at the first parking of the rotation
- T_{dep} : minimum processing time for departure; this is the minimum occupation time at the last parking of the rotation
- T_{depl} : minimum time of travel; if two tows are required (i.e., if we have 3 operations), this is the minimum time of the middle task

Remark 1

The waiting period exists (i.e., the middle operation) only if the turnaround is longer than $T_{arr} + T_{depl} + T_{dep}$.

Some physical constraints exist, as imposed by the airport infrastructure.

Rule 1 (Capacity)

The **capacity rule** prevent certain aircraft types from being placed on some parking.

Rule 2 (No-Overlap)

The **No-Overlap constraints** reflects the impossibility of assigning two operations (two flights) to the same parking. An operation $i \in \mathcal{T}$ overlaps with another operation $j \in \mathcal{T}$ if $a_i < d_j \wedge a_j < d_i$. The operations overlapping with i is denoted by $\mathcal{O}_i$, and so, contains all operations j overlapping with i .

Rule 3 (Shading constraints)

The **shading constraints** blocks an aircraft's positioning on some nearby parking (regardless of the type of aircraft, e.g., two large aircraft cannot be simultaneously assigned to adjacent stands due to space limitations).

Example 8 (Shading constraints)

For example, Figure 1.4 shows that, if an aircraft is placed on parking $A14$ then the parkings $A16$ are “shaded” and vice versa (it is symmetrical).

Rule 4 (Reduction constraints)

The **reduction constraints** are similar to the shading constraints except that they consider aircraft types. The **reduction constraints** can be defined by 4-tuples (k, p, k', Δ) with $\Delta \subset PK$ being a set of parkings. For every parking p' in Δ , a plane of type k' is allowed on p' if a plane of type k is placed on p .

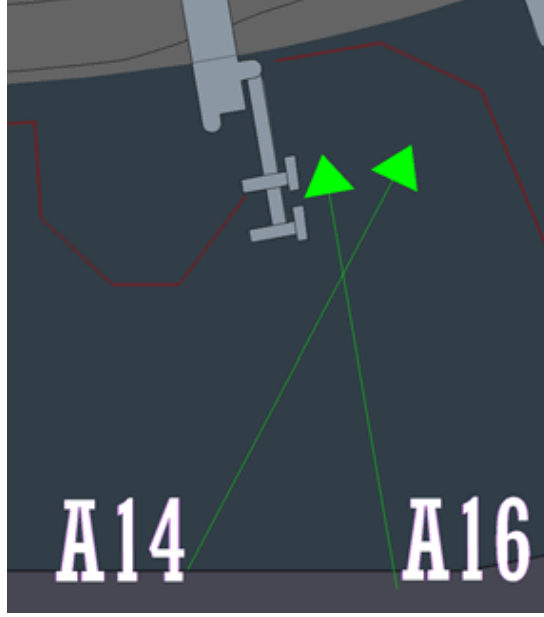


Figure 1.4: Example for the shading constraint. Parking $A14$ is shaded by parking $A16$ and vice-versa.

Example 9 (Reduction constraint)

As an illustration, considering Figure 1.5, shading will only be effective if a specific aircraft type has been placed on $A10$, then the reduction will still allow a subset of aircraft types to be placed on $A8$ and $A12$.

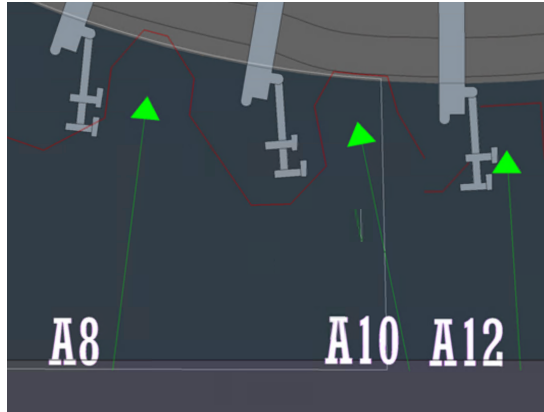


Figure 1.5: Example for the reductions constraint.

Notation 5 (Set of reductions)

We note $\mathcal{D}$ the set of all such reductions (all 4-tuples).

The components for **stand allocation problem formulation** at Paris Airports can be summarized as follows:

- $\mathcal{T}$ the set of operations. The previous section explains that an operation $i \in \mathcal{T}$ is defined by a start time a_i and an end time d_i . For each operation i , we also have the set of operations that overlap with i : $\mathcal{O}_i$.
- PK the set of the stands.
- $PK_i \subset PK$ the set of compatible stands (i.e stands with a capacity compatible with i) for operation i . Because each rotation operation concerns the same flight, the set of compatible stands for a rotation ϕ is the same for each operation i of $\mathcal{T}_\phi$.
- $\mathcal{U} : \mathcal{T} \mapsto \mathcal{T} \cup \{0\}$. $\mathcal{U}(i)$ is the direct successor of the operation i for a given flight, i.e., if a rotation is divided in two operations i and i_2 , then $\mathcal{U}(i) = i_2$. Conventionally, if operation i does not have a successor, then $\mathcal{U}(i)$ equals 0. The end time d_i of an operation $i \in \mathcal{T}$ is supposed to be equal to the start time $a_{\mathcal{U}(i)}$ of its successor if there is one.
- $\mathcal{Q} \subseteq \mathcal{T}^2 \times PK^2$ the set of shadow restrictions. If $(t_{\phi_1,i}, t_{\phi_2,j}, p_1, p_2) \in \mathcal{Q}$ and operation $t_{\phi_1,i}$ is assigned to stand p_1 , then operation $t_{\phi_2,j}$ cannot be assigned to stand p_2 and reciprocally. This quadruplet only exists if operation $t_{\phi_1,i}$ overlaps with $t_{\phi_2,j}$ (i.e., $t_{\phi_2,j} \in \mathcal{O}_{t_{\phi_1,i}}$).
- $\mathcal{D}$ the set of reductions. For a quadruplet $(k, p, k', \Delta) \in \mathcal{D}$ and an operation $i \in \mathcal{T}$ with an aircraft type k , if the operation i is assigned to the parking p , then only operations with an aircraft kind of k' are allowed to parkings of the set Δ .
- $\mathcal{MP} = (m_{i,p})_{\mathcal{T} \times PK}$ the affinity matrix, i.e $m_{i,p}$ is the airline satisfaction realized if operation $i \in \mathcal{T}$ is assigned to stand p .

An assignment $\mathcal{I}$ is a mapping between operations $\mathcal{T}$ and stands PK . The quality $\mathbf{f}(\mathcal{I})$ of an assignment $\mathcal{I}$ is defined as follows:

$$\mathbf{Obj}(\mathcal{I}) = \alpha \mathbf{C}_1(\mathcal{I}) - \beta \mathbf{C}_2(\mathcal{I})$$

where α and β are non negative. $\mathbf{C}_1$ and $\mathbf{C}_2$ are the total operation-stand affinity and the number of towing operations, respectively. We aim to find an assignment maximizing $\mathbf{Obj}(\mathcal{I})$ while respecting operation-stand compatibilities, shadow and reduction restrictions and overlapping constraints.

1.5 Airport check-in desk allocation problem

1.5.1 Introduction

In the previous section, we have presented the stand allocation problem, a classical air transport problem that involves assigning each flight to an available stand while maximizing both passenger conveniences and the airport's operational efficiency. Another classical problem is the *Check-in Desk Allocation Problem* (CDAP), which involves assigning each flight to one or more check-in desks depending on the airline's requirements. Different approaches in MILP (mixed-integer linear programming) have been proposed [YTC04, AR15]. A recent survey [LM22] explains that the airlines demand more check-in from the airport operator than required [Chu96].

This implies problems for the airport operator, who has to satisfy the real need for counters, ensure optimum distribution between all the airlines, and reduce changeover operations to a minimum. To avoid a change of operation for staff of the same flight, check-in desks were

allocated to a flight for its entire check-in time. The authors of [LM22] divide the allocation problem into two sub-problems:

- First, find the optimal number of check-in desks.
- Second, consider to place the resulting polyominoes (see Figure 1.6).

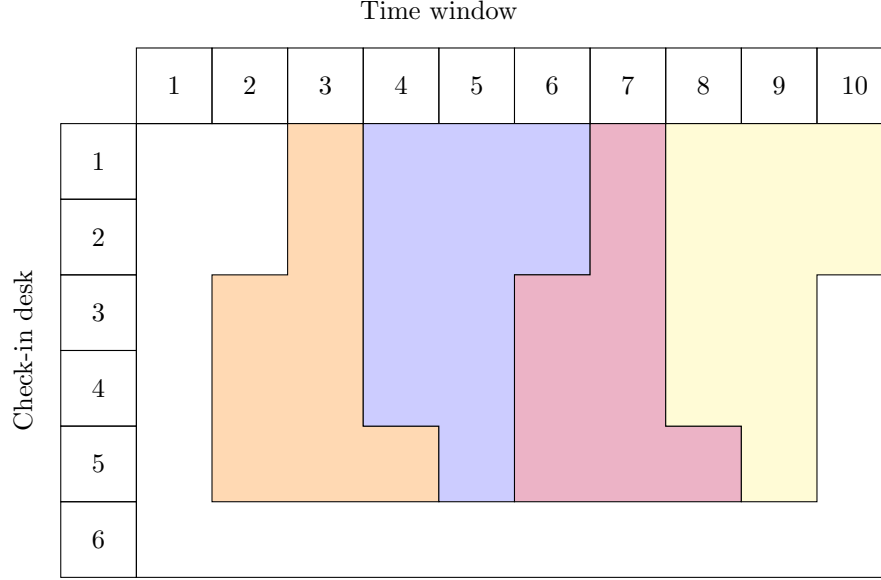


Figure 1.6: Polyominoes in counter allocation.

Although different approaches consider a dynamic number of check-in desks (see Figure 1.7a), in this chapter, we consider the case where the same number of banks is requested for the entire duration of the registration (see Figure 1.7b).

Before formally formulating the problem, we need to introduce a few terms:

Airport Term 4 (Check-in desk (or bank))

A *Check-in desk* (or **bank**) is the check-in and baggage counter and is noted c . We note $\mathcal{CD}$ the set of all check-in desks (i.e., $\forall c, c \in \mathcal{CD}$).

Airport Term 5 (Zones)

A check-in desk belongs to a *zone*. A zone is noted z . The set of all zones is noted $\mathcal{Z}$.

Airport Term 6 (Registration)

A *registration* corresponds to a flight or a set of flights of the same airline.

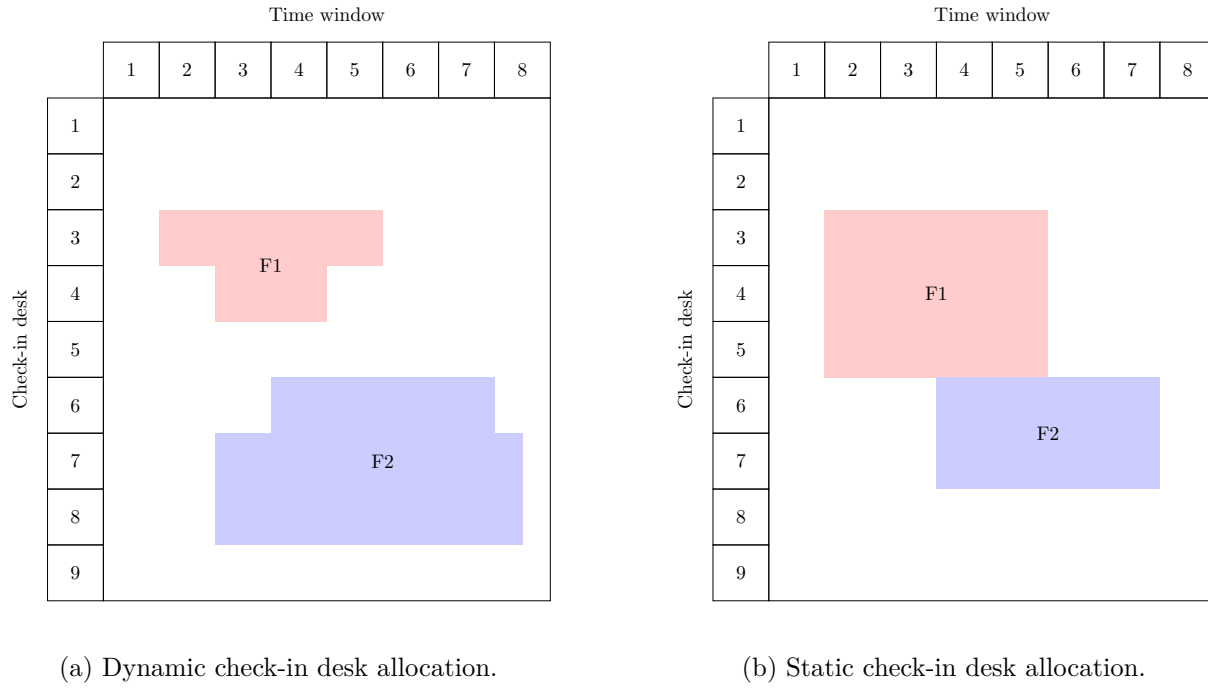


Figure 1.7: Difference between dynamic and static check-in desk allocation

Notation 6 (Registration)

We note a registration ρ . The set of all registrations is noted $\mathcal{R}$. We note a_r and d_r the registration's start time and end time.

Depending on the airline's requirements, registration can be broken down into 1 or more tasks.

Airport Term 7 (Task)

A task t is similar to an operation but has no successor. All tasks of a registration ρ have the same start time and end time.

Example 10 (Task and registration)

Figure 1.8 presents some registration tasks at Orly Airport with 1, 4 and 5 tasks.



Figure 1.8: An example of planning at Orly Airport

Notation 7 (Set of tasks)

The set of all tasks is noted $\mathcal{T}$.

The set of tasks of registration ρ is noted $\mathcal{T}_\rho$. We note ν the maximum number of tasks requested for a registration (i.e., $\max(|\mathcal{T}_\rho|, \forall \rho \in \mathcal{R})$).

Notation 8 (Affinity matrix)

We note $\mathcal{MC} = (m_{t,c})_{\mathcal{T} \times \mathcal{CD}}$ the affinity matrix based on the airlines preferences, i.e $m_{t,c}$ is the airline satisfaction realized if task t is assigned to check-in desk c .

Definition 21 (Compatible check-in desk $c \in \mathcal{CD}_t$ for a task t)

We call a compatible check-in desk $c \in \mathcal{CD}_t$ for a task t the banks that have an affinity with t strictly greater than 0 or a bank at a distance of at most number of requested banks minus 1. For a task t and its registration ρ , we define the set of compatible desk by affinity:

$$\mathcal{CA}_t = \{c \in \mathcal{CD} \mid m_{t,c} > 0\}$$

Now we can define the set of compatible desks by distance:

$$\mathcal{CDIST}_t = \{c_1 \in \mathcal{CD} \mid \exists c_2 \in \mathcal{CA}_t : \text{dist}(c_1, c_2) < |\mathcal{T}_\rho|\}$$

Finally, the set of check-in desks compatible with t is defined by the union of the two previous sets:

$$\mathcal{CD}_t = \mathcal{CA}_t \cup \mathcal{CDIST}_t$$

Definition 22 (Compatible Check-in Desk for a Task)

A check-in desk c is said to be compatible with a task t if it meets one of the following criteria:

1. The bank associated with the desk c has an affinity with task t that is strictly greater than zero ($m_{t,c} > 0$).
2. The bank is within a distance less than the total number of banks requested for the registration ρ (registration of t) minus one from any bank compatible by affinity.

For a given task t and its associated registration ρ , we define:

- The set of desks compatible by affinity as:

$$\mathcal{CA}_t = \{c \in \mathcal{CD} \mid m_{t,c} > 0\}$$

- The set of desks compatible by distance as:

$$\mathcal{CDI}_t = \{c_1 \in \mathcal{CD} \mid \exists c_2 \in \mathcal{CA}_t : \text{dist}(c_1, c_2) < |\mathcal{T}_\rho| - 1\}$$

Consequently, the complete set of check-in desks compatible with task t is determined by the union of desks compatible by affinity and those by distance:

$$\mathcal{CD}_t = \mathcal{CA}_t \cup \mathcal{CDI}_t$$

Example 11 (Compatible check-in desk)

Consider Figure 1.8, focusing on the group of blue tasks positioned on the left side (noted ρ): Four banks are required for this registration (4). For all tasks t in this registration, the set of banks that demonstrate an affinity strictly greater than zero is denoted as $\mathcal{CA}_t$ and equals 1103, 1104, 1105, 1106 in this scenario. Correspondingly, the set of banks located within a maximum distance of three (3) from any bank in $\mathcal{CA}_t$ is represented as $\mathcal{CDI}_t$, which in this case includes the banks 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109.

Remark: It is trivial to observe that allowing banks 1101, 1102, or 1103 as compatible banks for the last registration task would make placement impossible, as there would not be enough banks available to accommodate the preceding tasks. Consequently, in our modeling, a ‘zone constraint’ is implemented to prevent impossible configuration.

In the next section, we present the check-in desk allocation problem formulation.

1.5.2 Check-in Desk Allocation Formulation

In this section, we formally introduce the CDAP as defined at Paris Airports.

Rule 5 (Imposing Consecutive Desks)

The **consecutive constraints** ensure that the check-in desks used for a specific registration ρ are *consecutive*.

Example 12 (Imposing Consecutive Desks)

Figure 1.8 shows an example of an assignment that respects the consecutive constraint.

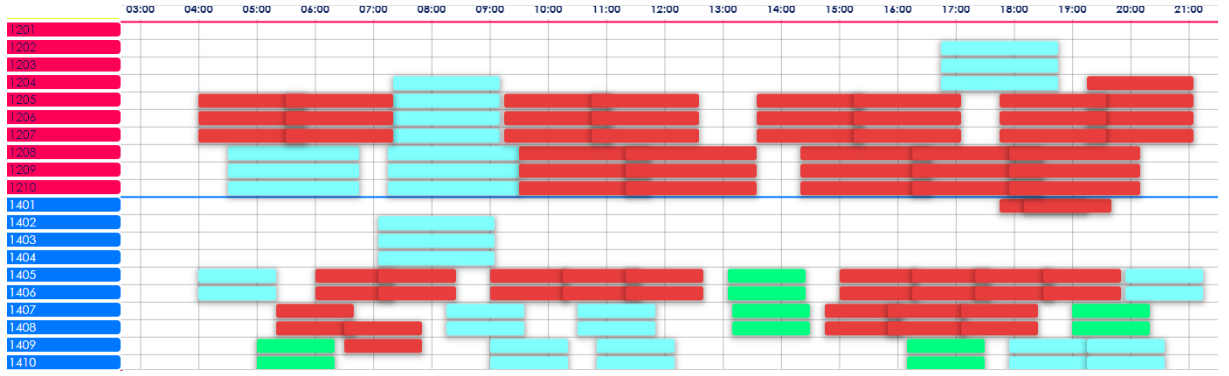


Figure 1.9: Example of a planning that allows overlapping.

Rule 6 (Same zone)

The **same zone constraints** ensure that the check-in desks used for a specific registration ρ come from the same zone.

Example 13 (Same zone)

For example, in Figure 1.9, there are two zones (colored in blue and red); so for registration, we cannot use both a blue and a red bank.

By default, a registration cannot share its assigned banks with another registration if the two registration tasks overlap. The non-overlapping constraint ensures this.

Rule 7 (No-overlap)

The **no-overlap constraint** ensures that a task cannot share its check-in desk with another task if the two tasks are time overlapping (i.e., so at any time, no bank can be shared by two different registrations).

However, for some logistical reason (space) and under certain general conditions (called overlapping rules), some overlapping between flights from the same airline company may be tolerated for a limited period and/or for a limited number of tasks.

Example 14 (Overlapping)

Figure 1.9 presents an example of planning that allows overlapping for 100% of the time and without a limited number of tasks.

Example 15 (Overlapping for a limited number of tasks)

If for example, the number of banks required by a registration is set to 4 and the maximum number of overlapping situations is 2, then only two banks from the four banks associated with the registration can be shared with another registration that shares the same overlapping rule.

Notation 9 (Set of forbidden overlaps)

We will note $\mathcal{F}$ the set of pairs of registrations (ρ_1, ρ_2) that cannot strictly share banks (they may be time overlapping, but no rule exists permitting to have shared banks between them).

Notation 10 (Set of Overlapping Rules)

We will note $\mathcal{OR}$ where each element or is a pair $(\mathcal{R}, t)$ or triplet $(\mathcal{R}, t, m)$. In the former case, $R \subset \mathcal{R}$ represents the set of registrations covered by the rule, while t indicates the permissible duration of overlap stipulated by the rules. In the latter case, an additional parameter m specifies the maximum number of tasks that can overlap. Finally, we note for each rule or and for each registration $\rho \in \mathcal{R}$, $\mathcal{N}_{\rho, or} \subset \mathcal{R}$ the set of neighbors (i.e., registrations that have a temporal overlap with ρ) of ρ considering the rule or .

Frequently, some banks are unavailable for several hours to several days (for example, for maintenance reasons).

Rule 8 (Unavailable constraints)

The **unavailable constraints** ensures that certain banks are not available for some time (which may be periodic). In other words, we must remove the check-in desk from the domain for each task that overlaps with the exclusion period. We note $u = (c, s, e)$ a triplet where c is the check-in desk to exclude, s and e is the start and end time of the excluded period.

Another type of exclusion is to exclude the check-in desk for a given registration regardless of the time.

Rule 9 (Exclusion constraints)

The **exclusion constraints** ensures that certain banks are excluded from certain registrations under some conditions.

Sometimes, employees (from Paris Airports) may want to force a specific set of banks to be associated with some registrations.

Rule 10 (Pre-assigning Banks)

The **pre-assignment constraint** ensures that the user's desired assignment is respected.

Notation 11 (Pre-assignment)

We will note (ρ, j, c) the triplet that represents the pre-assignment of bank c as the j th bank used by registration ρ ; all such triplets will be denoted by P .

The components for check-in desk allocation problem formulation at Paris Airports can be summarized as follows:

- $\mathcal{T}$ the set of tasks.
- $\mathcal{CD}$ the set of check-in desks.
- $\mathcal{R}$ the set of registrations.
- $\mathcal{T}_\rho$ the set of tasks for registration ρ .
- $\mathcal{CD}_\rho \subset \mathcal{CD}$ the set of compatible check-in desks for the registration ρ .
- P the set of pre-assignment.
- $\mathcal{E}_\rho$ the set of excluded check-in desks for a registration ρ .
- $\mathcal{U}$ the set of unavailable rules (i.e., the set of all triplet c, s, e).
- $\mathcal{F}$ the set of forbidden overlaps.
- $\mathcal{OR}$ the set of overlapping rules.
- $\mathcal{MC} = (m_{t,c})_{\mathcal{T} \times \mathcal{CD}}$ the affinity matrix.

An assignment $\mathcal{I}$ is a mapping between tasks $\mathcal{T}$ and check-in desks $\mathcal{CD}$. The quality $\mathbf{f}(\mathcal{I})$ of an assignment $\mathcal{I}$ is defined as follows:

$$\text{Obj}(\mathcal{I}) = \mathbf{C}(\mathcal{I})$$

where $\mathbf{C}$ are the total task-check-in desk affinity. Our objective is to find an assignment maximizing $\text{Obj}(\mathcal{I})$ while respecting task-check-in desk compatibilities, forbidden and allowed overlaps, and user constraints (e.g., excluded check-in desks and pre-assignment).

1.6 Conclusion

This first chapter introduces the concept of the **constraint satisfaction problem**, the underlying **constraint network**, and its components. As an extension of this problem, we have also introduced the **constraint optimization problem**. Next, we introduced two common problems in an airport context: the stand allocation problem and check-in desk allocation problem. In Chapters 3 and 4.2.3, we will see how to model these problems via a COP model. Finally, Chapter 5 presents some methods for resolving these problems.

Chapter 2

Machine learning

Contents

2.1 Introduction	29
2.2 Definitions and notations	30
2.3 Explanation methods for a regression model	36
2.4 Example: How many hotdogs will be sold?	37
2.4.1 Exploratory Data Analysis (EDA)	38
2.4.2 Correlation Analysis	39
2.4.3 Experiments	40
2.4.4 Explanations	41
2.4.5 Conclusion	42
2.5 Machine Learning Usage at Airport	42
2.5.1 Forecasting and simulation	43
2.5.2 Passengers flight prediction	43
2.5.3 Flight delays	43
2.6 Machine learning usage at Paris Airports	44
2.6.1 Room	44
2.6.2 PAX	45
2.6.3 CNX PAX	45
2.6.4 Estimating the shape of PAX presentation curves	45
2.6.5 PAX presentation	46
2.7 Conclusion	47

2.1 Introduction

This chapter focuses on *Machine Learning* (ML) in an airport context. Machine learning is divided into three paradigms:

- *Supervised Learning* : In this paradigm, the *agent* (the *learner*) is provided with a *labeled dataset*, where the *input data* is associated with corresponding *output labels*. The objective is for the algorithm to learn the link between *inputs* and *outputs* so that it can make accurate *predictions* on new, *unseen data*. Typical examples of supervised learning

tasks include *classification*, where the agent assigns inputs to predefined categories, and *regression*, where the learner predicts continuous numerical values.

- *Unsupervised Learning*: In this case, we deal with datasets that are not labeled. The algorithm's task is to identify patterns, relationships, or structures within the data without explicit guidance on the output. *Clustering* is a common example of *unsupervised learning tasks*.
- *Reinforcement Learning*: In *reinforcement learning*, an agent interacts with an environment and learns by receiving *feedback* in the form of *rewards* or *penalties* based on its actions. The agent aims to discover the best actions or policies that maximize the cumulative reward over time.

This manuscript will concentrate on the first paradigm, as the other two are not covered here. The chapter begins by introducing different types of models used in machine learning. We then explore various methods for explaining these models. Before concluding, we present the usage of machine learning in the airport context, focusing on the Paris Airport case.

2.2 Definitions and notations

This section introduces the concepts commonly used to describe a learning task or learning problem.

Definition 23 (Attribute)

We consider a *set of attributes* $\{a_1, a_2, \dots, a_n\}$ where each *attribute* a_i (or *feature*) takes its value v_i in a domain D_i . The values can be a *totally ordered set of numbers* (**real numbers** $\mathbb{R}$, or **integers** $\mathbb{Z}$), *categorical* (in this case, the values are not necessarily ordered, e.g., $D_i = \{\text{AirFrance}, \text{EasyJet}, \text{Emirates}\}$) or **Boolean** (D_i is $\mathbb{B} = \{0, 1\}$).

Definition 24 (Instance)

An *instance* is an observation $\chi = v_1, v_2, \dots, v_n$ where each v_i is an element of the domain D_i .

Definition 25 (Training and Test Set)

An *example* is a pair (χ, y) where χ is an instance and y is the *label* or an associated decimal value. A *training set* is a list E of examples used to train the model. By contrast, the *test set* tests its performance.

Definition 26 (Decision function)

Let E be a trainet set: A *decision function* is a function $f : E \mapsto Y$ mapping each example (χ, y) of E to $y = f(\chi)$.

Definition 27 (Binary classifier)

A *binary classifier* is a decision function where Y is composed of two classes (i.e., $f : E \mapsto \mathbb{B}$).

Example 16 (Hotdog or Not Hotdog)

To explain the binary classifier, let us take the example of a model that determines whether an image represents a hotdog. The positive class is *hotdog*, the negative class is *not hotdog*^{*ab*}.

^aThanks to the series Silicon Valley for this funny idea <https://www.youtube.com/watch?v=vIci3C4JkL0>

^bA real dataset of images for this task is available on Kaggle: <https://www.kaggle.com/datasets/dansbecker/hot-dog-not-hot-dog>

The previous example shows us that using a binary classifier can have some limitations. There are also *multi-class classifiers* that can add other classes like *pizza*, *hamburger*, etc. However, this kind of classifier is outside the scope of this manuscript.

Definition 28 (Regressor)

A *regressor* (or *regression model*) is a decision function where Y is real number from $\mathbb{R}$ (i.e., $f : E \mapsto \mathbb{R}$).

Remark 2

Unless otherwise stated, the models used in this manuscript are *regression models*.

We now introduce *decision tree* and **boosted decision tree**. The latter is the principal kind of model used in this manuscript.

Definition 29 (Decision Tree)

A *decision tree* (in case of regression) $\mathcal{T}$ over $a_1, \dots, a_n$ is a *binary tree* where each *node* is labelled with a *condition* on input features $a_1, \dots, a_n$ and each *leaf* is labelled by a *real number* (from $\mathbb{R}$).

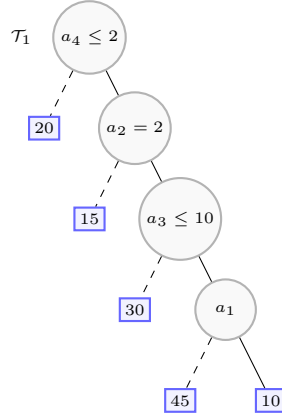


Figure 2.1: A decision tree

Definition 30 (Condition)

A *condition* in a *decision tree* can take different forms:

- $a_i \leq v_j$ with v_j a number when a_i is a *numerical attribute*.
- $a_i = v_j$ or $a_i \in \{v_1, v_3, v_6\}$ when a_i is a *categorical attribute*.
- a_i when a_i is a *Boolean attribute*.

Notation 12 (Output of a decision tree $\mathcal{T}$ over an instance χ)

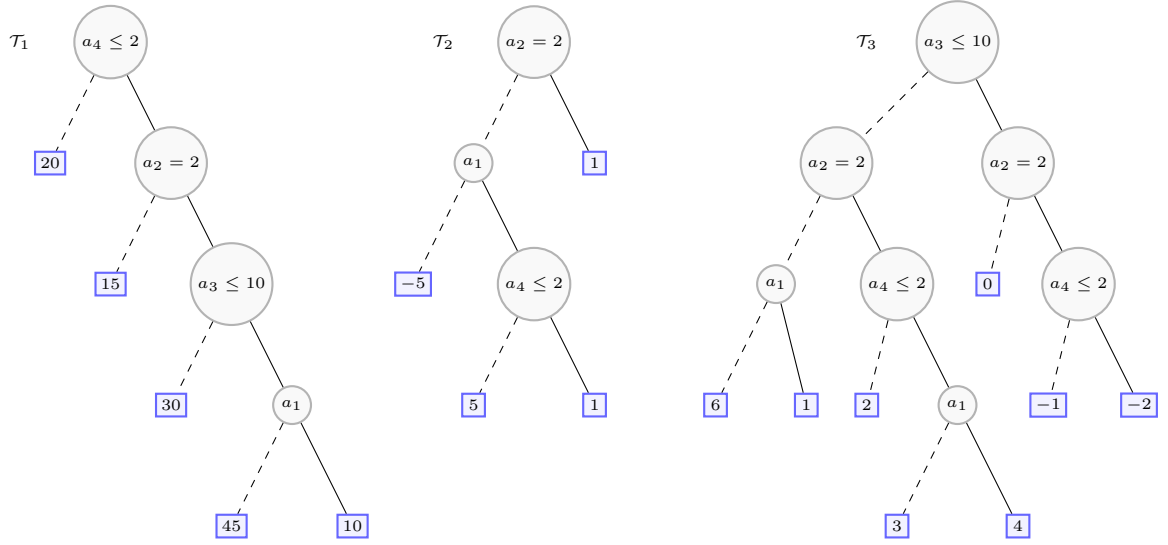
We note the *output* of the decision tree $\mathcal{T}$ over the instance χ : $\mathbf{o}(\mathcal{T}, \chi)$.

Example 17 (Decision Tree)

Figure 2.1 shows a basic decision tree.

Circular nodes represent conditions, while square nodes represent leaves. The left (dashed) arc (resp. the right (plain) arc) outgoing from any (circle) node labeled by a condition c corresponds to the case c is false (resp. true). Consider an instance $\chi = (a_1 = 1, a_2 = 2, a_3 = 9, a_4 = 0)$, we always take the right child, as the condition is verified and the decision tree output is 10.

Because using a single tree may create a *weak model*, there are different methods for combining these models and creating a model called *stacked model* or *ensemble model* [Bre96, Fri02, Bre01].

Figure 2.2: An example of *decision forest* $\mathcal{F}$ for a regression task with 3 trees $\mathcal{T}_1, \mathcal{T}_2$ and $\mathcal{T}_3$.**Definition 31 (Decision Forest)**

A *decision forest* represents a collective term for models constituted by multiple *decision trees*. Such a forest can be denoted as $\mathcal{F} = \{\mathcal{T}_1, \dots, \mathcal{T}_m\}$, where each individual tree, $\mathcal{T}_i$ ($i \in [m]$), functions as a binary tree. The *prediction* derived from the forest is an *aggregation* of the predictions (also referred to as outputs) of its inherent decision trees. This prediction can be symbolized as $\mathbf{o}(\mathcal{F}, \chi)$ or $\hat{y}$. The models that form part of this ensemble are termed *constituent models*. Collectively, they make up the *ensemble model*, often referred to as the *strong model*.

Example 18 (Decision Forest)

For example, in a multi-class classification random forest^a (a type of decision forest), each tree votes for a single class, and the random forest prediction is the most represented class. In a *regression gradient boosted decision tree* (another type of decision forest), each tree outputs a real, and the *gradient boosted tree prediction* is the sum of those values.

Now consider the decision forest on Figure 2.2 composed of 3 trees and the same instance χ as the example 17, we have $\mathbf{o}(\mathcal{T}_1, \chi) = 10$, $\mathbf{o}(\mathcal{T}_2, \chi) = 1$, $\mathbf{o}(\mathcal{T}_3, \chi) = -2$. If we consider **sum** as an *aggregation function*, the output for the decision forest is $\mathbf{o}(\mathcal{F}, \chi) = 9$

^aThis example is taken from <https://developers.google.com/machine-learning/decision-forests/intro-to-decision-forests-real?hl=en>

For training, a random forest uses *bagging*. Each tree in the random forest trains on a subset of training examples *sampled with replacement*.

Another method is to use *gradient boosting* applied to decision forest, which formed a *gradient boosting decision tree* [Fri02, RG15].

Definition 32 (Gradient boosting)

Gradient boosting is a training algorithm where each *constituent model* is trained iteratively for improving the quality of the *main model* (or the *strong model*).

Definition 33 (Gradient Boosting Decision Tree)

A *Gradient Boosting Decision Tree* (GBDT) is a kind of *decision forest* in which:

- *training* used *gradient boosting*.
- each *weak model* is a *decision tree*.

At each step, a new weak model is trained to predict the *error* of the current *strong model* (i.e., it learns a correction for the strong model learned so far). The weak model is then added to the strong model.

$$\mathcal{F}_{i+1} = \mathcal{F}_i + \mathcal{T}_i$$

where:

- $\mathcal{F}_i$ is a strong model at step i
- $\mathcal{T}_i$ is a weak model (a **decision tree**) at step i

The iteration stops when the maximum number of iterations is reached.

GBDT is very efficient on many machine learning tasks, such as multi-class classification [Li10], click prediction [RDR07] and, learning to rank [Bur10].

Now that we have described what a regression model is, we can present well-known regression quality measures based on residuals (errors between predicted and actual values) to assess the regression quality.

Notation 13 (Notation for metrics)

For a set of samples E , we note:

- $\hat{y}_i$ the predicted value for the i -th sample.
- y_i the true value for the i -th sample.
- $\bar{y}$ the average of the target variable.

Based on this notation, we can define the error.

Definition 34 (Error)

The *error* between the *predicted value* and the *true value* is defined as:

$$\epsilon_i = y_i - \hat{y}_i$$

Based on this *error* formulation, we can define the *mean absolute error* (MAE) based on *Manhattan distance*.

Definition 35 (Mean absolute error)

Mean Absolute Error (MAE) measures the average absolute difference between the *actual* and *predicted values*. It provides a straightforward representation of the average prediction error. This measure is less sensitive to extreme values in the data. For a set of samples E , the **MAE** is defined as:

$$\text{MAE}(E) = \frac{1}{|E|} \sum_{i=1}^{|E|} |\epsilon_i|$$

Definition 36 (Mean squared error)

Mean Squared Error (MSE) measures the average of the squared difference between the actual and predicted values. Unlike **MAE**, **MSE** penalizes the most significant errors more heavily. For a set of samples E , the **MSE** is defined as:

$$\text{MSE}(E) = \frac{1}{|E|} \sum_{i=1}^{|E|} (\epsilon_i)^2$$

Definition 37 (Root Mean squared error)

Root Mean Squared Error (RMSE) is the square root of **MSE** and represents the typical magnitude of the prediction errors. **RMSE** is particularly useful when we want to interpret prediction errors in the same unit as the target variable. For a set of samples, the **RMSE** is noted **RMSE(E)** and defined as:

$$\text{RMSE}(E) = \sqrt{\text{MSE}(E)}$$

Finally, the *coefficient of determination* (R^2) is the last measure.

Definition 38 (Coefficient of determination)

Coefficient of determination is a statistical measure that represents the proportion of the *variance* for a dependent variable that is explained by variables in a regression model and is defined as follows:

$$R^2(E) = 1 - \frac{\sum_{i=1}^{|E|} (\epsilon_i)^2}{\sum_{i=1}^{|E|} (y_i - \bar{y})^2}$$

2.3 Explanation methods for a regression model

Several tree-based ML models were presented in the previous section. These models are inherently interpretable, but the interpretation can become more complex as the trees get larger and more complicated tree methods like random forests or boosted trees are used.

Due to the extensive use of machine learning in various applications, much research has focused on explainability in recent years. **eXplainable AI (XAI)** plays a crucial role in making complex AI models understandable to humans. XAI aims to develop effective methods and approaches for interpreting learning models and providing **explanations** for the decisions made [HDM⁺11, RSG16, FH17, LL17, GMR⁺18, IMM18, SDC19, HEKK19, Mil19, SDC19]. We now introduce some XAI concepts based on the presentation by [BADDs⁺20].

Definition 39 (Model agnostic)

The purpose of *model-agnostic* techniques are to extract specific information from a model's prediction process and can be implemented with any model.

Definition 40 (Explanation by simplification)

It is a sub-category of *model-agnostic* post-hoc methods. We found *local explanations* or methods that consist of *rule extraction*.

For example, **Local Interpretable Model-agnostic Explanations (LIME)** [RSG16] is a *model-agnostic* approach that creates interpretable explanations for individual predictions by approximating the model's behavior. It perturbs the input data and observes how the predictions change to determine *feature importance*.

Definition 41 (Feature relevance explanation methods)

Feature relevance explanation methods for posthoc explainability aim to describe the functioning of a *black-box model* by *ranking* or *measuring* the *importance* (*influence*) of each feature in the *prediction output* by the model to be explained.

SHapley Additive exPlanations (SHAP)[LL17] is a popular method in the **feature relevance explanation methods** category. SHAP values are based on cooperative game theory and attribute the prediction outcome to each feature in the **input data**.

LIME and **SHAP** are two model-agnostic approaches and have the advantage in theory of avoiding the need to access the model's internal. Otherwise, it has been shown that SHAP or LIME can produce similar *feature attributions* for different predictions.

We can note that, recently, some generic agnostic approaches that generate *symbolic explanations* have been proposed [BCMT21a].

Definition 42 (Model precise)

In *model-precise (model-specific)* methods the internal structure of the concrete ML model is used for reasoning and generating explanations.

In the case of classification, we can refer for example [SCD18, IM21, ABB⁺22b, ABB⁺22a]

2.4 Example: How many hotdogs will be sold?

In this section, we will extend the *hotdogs* example in the context of a regression task to answer the question: *how many hotdogs will be sold?*. For this purpose, we will use a notional dataset specially created for this manuscript. We randomly generated a dataset of 100,000 rows. To make the randomly generated data more realistic, we gave a higher probability of having more hotdogs sold when the day had specific characteristics. The source code used to generate the data is available on [Gitlab](#).

To approach this example, we will follow the same procedure as the rest of this document. Here is an outline of the steps we will take:

1. **Exploratory Data Analysis (EDA)**: We will begin by studying various figures and statistics about the dataset to gain insights into its characteristics. This will include examining the distribution of the *target variable (the number of hotdogs sold)* and analyzing the relationships between different *attributes* and the *target variable*.
2. **Correlation Analysis**: Next, we will investigate the correlation between the various attributes (features) and the target variable. This will help us to identify which features significantly impact the *number of hotdogs sold*.
3. **Experiments**: We will explore different regression models to predict the *number of hotdogs sold* based on the available features. This may involve decision tree and boosted tree.
4. **Explanations**: Finally, we will generate explanations for the predictions made by the regression models. This will involve techniques such as feature importance analysis to understand which features contribute the most to the prediction.

We aim to build a regression model that can effectively predict the *number of hotdogs sold* based on the given dataset. Additionally, we will gain insights into the factors influencing hotdog sales and explain the model's predictions. All these experiments are available on [Gitlab](#) ().

Name	Description	Type	Example
Season	The season of the sales day.	Categorical	Winter
Temperature	The temperature (Celcius) on hotdog sales day.	Numeric	18
Day	The day of the week of the sales day.	Categorical	Saturday
Holiday	The sale day is a public holiday or not.	Boolean	True
Event	Event or not.	Boolean	False
Price	The price of the hotdog.	Numeric	2.25
Quantity	The number of hotdogs sold.	Numeric	150

Table 2.1: Attributes for the sold hotdog prediction task

2.4.1 Exploratory Data Analysis (EDA)

Table 2.1 presents the dataset’s features used in the example. The dataset consists of 100,000 rows, each one representing a sample with various features. For example, they are the *season* encoded with a categorical value that represents different seasons (e.g., winter, spring, summer, autumn), a number indicating the *temperature* on the respective day, and some Booleans for indicating whether the day is a holiday or if the day has a special event, such as a football match, concert, or any other event.

Figure 2.3 shows the number of hotdogs sold considering if the day has a special event (1, bars on the right) or not (0, bars on the left) and if it is a holiday day (*orange* bars) or not (*blue* bars). The bar plot shows that the most hotdogs were sold on holidays with an event.”

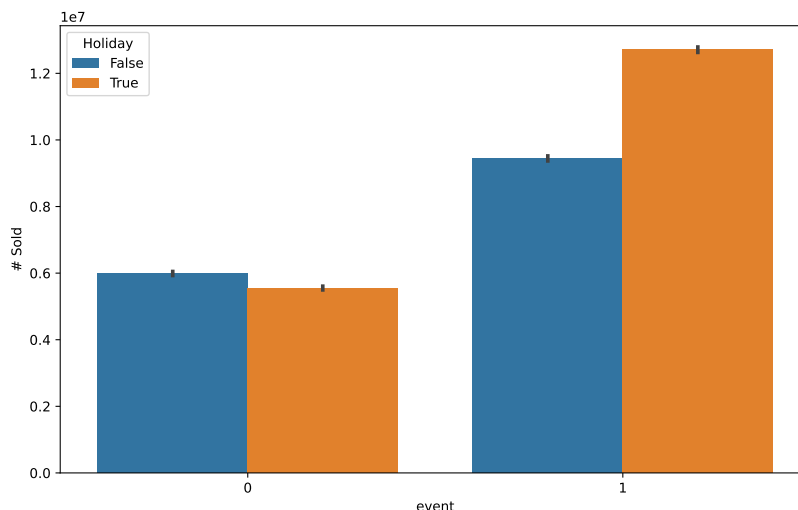


Figure 2.3: Count of hotdogs sold on holiday in event feature.

Figure 2.4 shows the number of hotdogs sold considering the day of the week and if the day has an event (*orange* bars) or not (*blue* bars). The bar plot shows that most hotdogs were sold during the weekend.

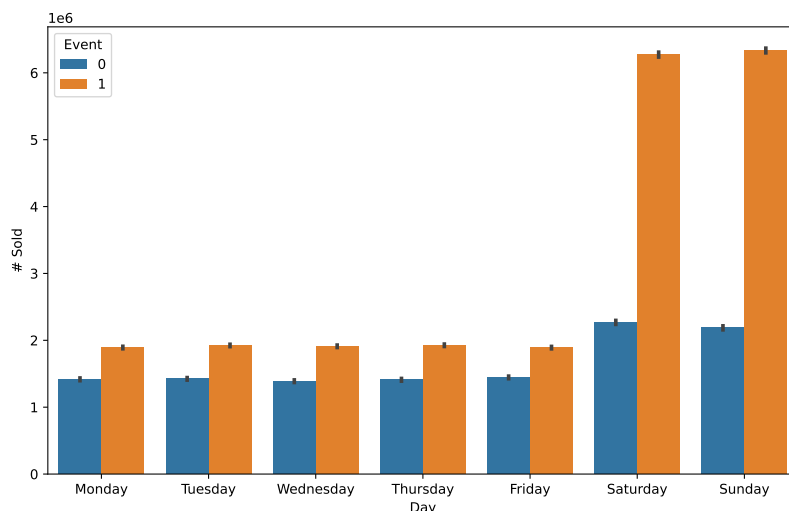


Figure 2.4: Count of hotdogs sold on an event in day feature.

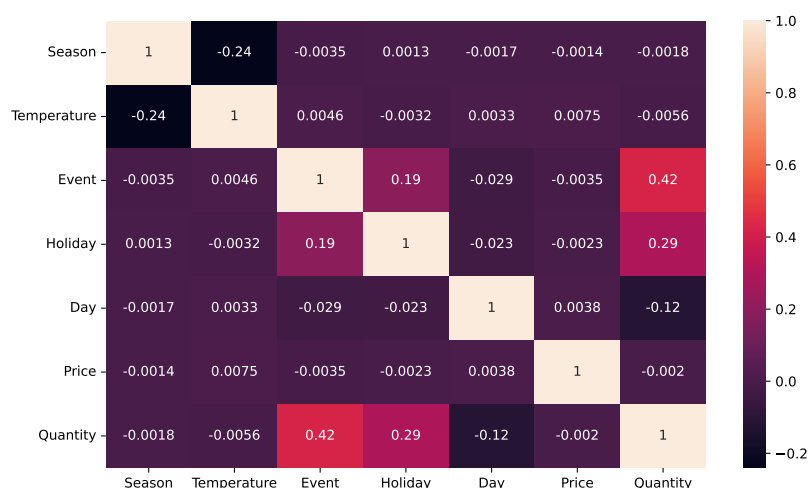


Figure 2.5: Results of statistical correlation measure.

2.4.2 Correlation Analysis

We use the well-known *Pearson correlation coefficient*, which is a statistical measure that calculates the linear relationship between two variables.

It is a number between -1 and 1 that measures the *strength* and *direction* of the relationship between two variables.

Figure 2.5 shows a *heatmap* where x -axis and y -axis present all the variables of the dataset. The values in the heatmap cells indicate the Pearson correlation coefficient between the corresponding variables. Values range from -1 to 1 , with 1 indicating a perfect positive correlation, -1 indicating a perfect negative correlation, and 0 indicating no correlation. The heatmap re-

	MAE	MSE	RMSE
Model			
$\mathcal{T}_l$	123.78	27386.16	165.48
$\mathcal{T}_o$	123.13	26988.00	164.27
LightGBM	108.02	19806.64	140.73

Table 2.2: Model Performance Metrics for Predicting hotdog Sales.

veals a positive correlation between *quantity* and *event* features with a correlation coefficient of 0.42 while the correlation is weaker between *quantity* and the *holiday* variable (0.29). Finally, the correlation between *quantity* and *season* or *temperature* seems close to 0.

2.4.3 Experiments

In this section, we describe the experiments conducted to predict the number of hotdogs sold using two different machine learning models: **Decision Tree** and **LightGBM**. The objective of these experiments is to compare the performance of the models in predicting hotdog sales based on various features present in the dataset.

2.4.3.1 Experiment Environment

The experiments were conducted on a computer equipped with an *Intel® Core™ i9-10900 CPU* running at *2.80GHz*, with 20 cores, and 64 GB of RAM. The environment was set up using the *conda* environment and *Python 3.11*.

2.4.3.2 Experimental procedure and evaluation metrics

To evaluate the performance of the models, we employed *k-fold* cross-validation with $k=10$. *K-fold* cross-validation is a technique where a dataset is divided into k subsets, with each subset serving as a test set once while the remaining $k - 1$ subsets serve as the training set to evaluate the performance of a machine learning model. This process was repeated ten times, ensuring each fold served as training and validation data. For model evaluation, we used metrics presented in the last section: [MAE](#) and [RMSE](#).

2.4.3.3 Results

Table 2.2 summarizes the results of the experiments for both the **Decision Tree** and **LightGBM** models presenting the average performance metrics over the 10 iterations of cross-validation.

The first column lists the model names:

- $\mathcal{T}_l$ corresponds to a decision tree with the usage of *label encoding* for categorical values.
- $\mathcal{T}_o$ corresponds to a decision tree with the usage of *one hot encoding* for categorical values.
- **LightGBM** corresponds to a **LightGBM** model with all default values maintained.

The presented table displays the average values of the evaluation metrics, namely [MAE](#), [MSE](#) and [RMSE](#), computed across the 10 iterations of cross-validation. The results show that the **LightGBM** model consistently outperformed both **Decision Tree** models across all three metrics.

2.4.4 Explanations

This section uses [SHAP](#) to interpret the Hotdog Sales Model. We use the training dataset, the testing dataset, and the *model* produced by the first iteration of the cross-validation. However, similar figures can be produced for the other iterations⁷.

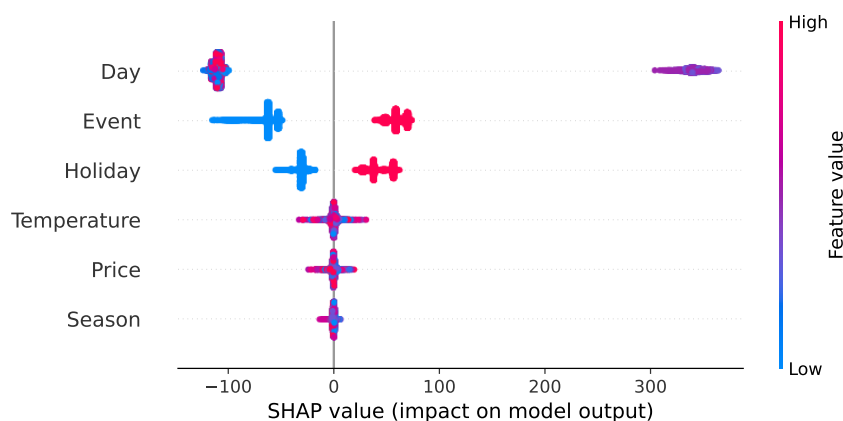


Figure 2.6: Summary plot of SHAP values for hotdog sales model.

Each *feature* in Figure 2.6, is represented by a horizontal bar on the chart. The position of the bar along the x-axis indicates the magnitude of the feature’s impact on the model’s predictions. The bar’s color reflects the value of the corresponding feature, with blue representing lower feature values and red indicating higher values. For example, a lower value (i.e., the Boolean value 0) for the feature *Event* impacts the model output negatively while a high value for the feature *Holiday* (i.e., the Boolean value 1) impacts the model output positively. Finally, the SHAP values largely confirm the coefficients of Pearson from section 2.4.2 and the importance of the impact of the *Event* and *Holiday* variables.

Season	Temperature	Event	Holiday	Day	Price	Quantity	$\hat{y}$	Error
Winter	3.5	0	0	Saturday	4.25	1111	515.805646	595.194354
Winter	6.0	0	0	Wednesday	4.25	136	135.992558	0.007442

Table 2.3: Instances with the minimal and maximal error.

Table 2.3 presents two examples. The first row represents the example with the maximal error, while the second row represents the example with the minimal error. Figures 2.7 and 2.8 offer a more granular view of the Hotdog Sales model’s predictions for examples with minimal error (resp. maximal error). Figure 2.7 shows the impact of each feature on the model output, for instance, the first row on Table 2.3. The mapping between the categorical feature *Day* and this integer code uses the lexicographic order of the *Day* (e.g., *Friday* = 0, *Monday* = 1, *Saturday* = 2, ..., *Wednesday* = 6). The starting point is the model’s average prediction on the training set (336.658). Each bar represents the contribution of a particular feature to the prediction. If we sum the model’s average prediction with each bar value, we obtain the model output (135.993). Figure 2.8 is similar to Figure 2.7 but for the example with the maximal error.

⁷Recall that these experiments can be accessed on Gitlab ()

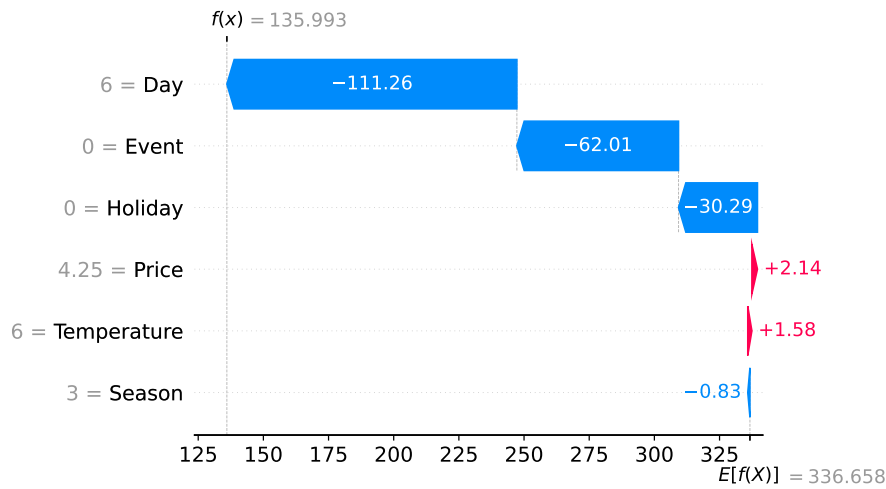


Figure 2.7: Waterfall plot for the instance with the minimal error.

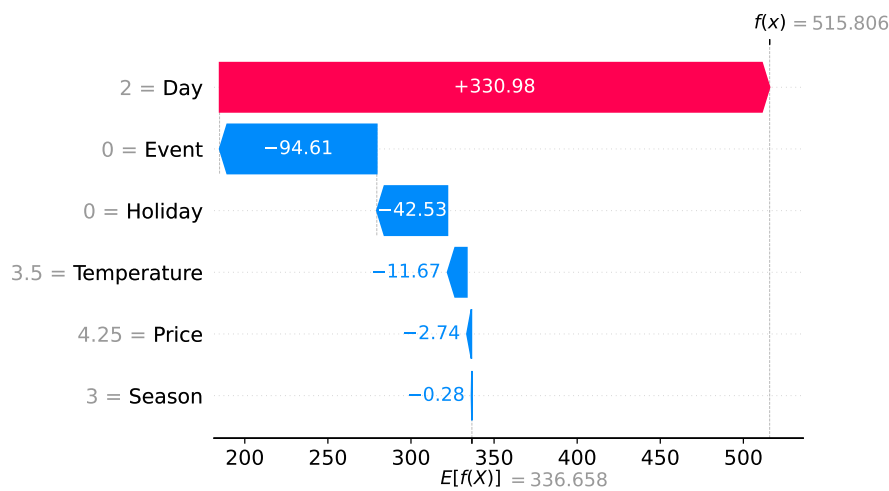


Figure 2.8: Waterfall plot for the instance with the maximal error.

2.4.5 Conclusion

The hotdog sales example is a practical and relatable demonstration of the processes and methodologies associated with machine learning that will be used in this document. It presents a fictive scenario where predictive analytics can be applied while ensuring comprehensive coverage of key steps in data analytics, from exploratory data analysis and correlation investigation to model experimentation and interpretation.

2.5 Machine Learning Usage at Airport

In the context of airports, artificial intelligence (AI) can be utilized to address various important issues, such as optimization (see Chapter 1), simulation, and prediction.

2.5.1 Forecasting and simulation

AI-powered solutions, such as Veovo and Amorph.aero, have been adopted by several airports to conduct forecasts and simulations for various scenarios and situations. These simulations help to anticipate potential management challenges and allow for proactive measures. Simulations are especially advantageous in addressing a crucial challenge: managing passenger flow, particularly within large airports and hubs. The primary goal is to prevent congestion, reduce queue delays, and enhance the overall passenger experience.

Airport simulations involve exploring various scenarios and assessing their outcomes regarding congestion, delays, and other factors. These simulations can be as simple as predicting passenger flow patterns for a given day or simulating queues with different numbers of open lines at security checkpoints, various authorities (police controls, customs, and immigration), and check-in counters. Additionally, “what-if” scenarios are simulated to understand the potential impact of specific changes or events. Various AI techniques, including machine learning and planning tools, are employed for these simulations, utilizing real historical and simulation data [SMMFPMM21].

2.5.2 Passengers flight prediction

Predicting passenger flow, also known as Short-Term Traffic Prediction, is a critical aspect not only in airport transport [LHC03, MJB⁺20, BDFS22, LLL⁺23] but also in domains like Bus Transport [LC17, KKS⁺19, NGSH22]. Deep learning approaches have been proposed to consider the temporal features and seasonality of the data to make accurate predictions [LEJ⁺21].

In airport transport, predicting various types of information is crucial for optimizing costs and reducing waiting times. This includes predicting the number of **Passenger (PAX)** on a flight, which is sometimes not readily shared by airlines, and the number of **Passengers with Reduced Mobility (PRM)** for better treatment and to avoid delays. Additionally, predicting the presentation curve of passengers at security checkpoints is vital for efficient airport operations.

2.5.3 Flight delays

Flight delays have long been a significant concern for airlines and airports. Several studies have explored different models to predict flight delays, such as support vector machines, random forests, and logistic regression [NMBS18, NG17]. Some studies focus on predicting departure flight delays, while others tackle arrival delays [VAA⁺17]. In [IEM21a], the authors compared different machine-learning methods (random forest, **logistic regression**, *Bayesian naive classifier*, and decision trees) for delay prediction on arrival. In [Tan21], seven binary models are compared. In [YZL⁺21], the authors have proposed several stacked approaches for the *Boston Logan International Airport flight* with data from January to December 2019.

Factors affecting flight delays have also been extensively examined. Weather conditions play a crucial role and some studies have concentrated on weather data as the primary feature for predicting delays [YDS⁺20], for instance, at Frankfurt airport [MHR08]. Additionally, the impact of flight connections and other factors has been investigated [WSW03].

In [EM20], a support vector machine (SVM) model is used. Based on 20 days, this latter study examines some causes of air traffic delays at the three major airports in New York City. The survey in [CLFZ22] proposes a deep learning approach for flight delay prediction considering a multi-airport scenario. Regarding regression-based processes, the authors [RB14] have proposed methods based on classification and regression with random forests for US airports.

Notably, only one study has attempted to predict takeoff delay with a regression model, specifically for the *Maastricht Upper Area Control Centre* (MUAC). This study utilized static and dynamic variables, including variables, to capture congestion levels at departure airports by representing the number of delayed flights in the periods leading up to a flight's departure [CBN⁺19]. Thus, they have a variable representing the number of delayed flights 60, 30, and 15 minutes before flight departure. The comparison of different models has shown that certain approaches, such as the **LightGBM** model [KMF⁺17], exhibit marginally better performance compared to *Recurrent Neural Network* (RNN) models like *Long Short Time Memory* (LSTM) [HS97] for flight delay prediction.

2.6 Machine learning usage at Paris Airports

Paris Airports developed a complex process, called **Airport Operation Plan** (AOP), to estimate presentation curves at critical resources such as security checkpoints. AOP collects information about each flight and its progress from operational applications like **Airport Operational Database** (AODB), **Root Mean Squared Error** (RMSE) that assigns the resource *check-in desk*, *stand*) to the flight for the boarding or unboarding process, and **Baggage Reconciliation System** (BRS) that checks if the luggage is on the same flight as its owner passenger. It also collects data from METEO France (weather conditions), GOOGLE (road access time to the platform), and sensors deployed in the airport.

Concerning sensors, there are, for example, **XOVIS** to measure waiting times in specific areas and **FLUXPAX**⁸. AOP consumes FLUXPAX data from scans at different milestones to obtain information on the number of passengers seen, namely:

- Number of passengers seen at the check-in desk.
- Number of passengers seen at security checkpoint.
- Number of passengers on board.
- Proportion of passengers observed in fast-track.

Every minute, the flight information is updated from the data warehouse and sensors. Not all data are present all the time. The information on a flight tends to be more accurate as we approach its departure (or arrival). AOP fills in the hole in the data using different ML models, described in the following. Each of these models uses the **FastTree** model from the **ML.NET** library [AAB⁺19]. It is an implementation of the *DART* algorithm [RG15].

The objective of the AOP is to predict how resources will be consumed. To do this, the AOP computes how each flight consumes a resource for each slot of 5 minutes. AOP aggregates the consumption of each slot and generates the consumption curves. There are 288 slots by day.

We first presented each model of AOP, and in chapters 6 and 7, we will present our main contributions to AOP.

2.6.1 Room

The first step is to determine the *boarding* or *unboarding* room. If the *room* is known when the AOP retrieves the information, it is considered reliable; else, a ML model based on the last 400 days determines the room.

⁸FLUXPAX is a system for scanning passenger tickets.

2.6.2 PAX

Before attempting to predict the presentation curves of passengers over time at the security checkpoint, we must first determine the number of passengers (PAX) on the flights. However, passenger behavior varies; they only go through the check-in desk if they have luggage. Thus, the number of passengers checking in at a desk cannot accurately indicate the number of passengers. Nonetheless, obtaining this number is crucial for predicting the passenger presentation curve at the security checkpoint, which helps to determine the number of lines that should be opened and reduce waiting time. This is why if the information on the number of passengers on the flight is not considered reliable (i.e., from the airline) then a *machine learning* model based on a flight history similar to the previous model (400 days) is used to determine the number of passengers on the flight.

2.6.3 CNX PAX

Knowing the CNX PAX is not sufficient to determine the presentation of the passenger of the resources of type *transit*. For each *departing flight*, it is necessary to know the origin of the PAX by terminal/unboarding hall and the type of customs circuit. In symmetry, for each *arriving flight*, the destination of the PAX by terminal/boarding hall and type of customs circuit must be known. The AOP manipulates groups of PAX composed of the following information:

- a source terminal
- a provenance room
- a destination terminal
- a destination room
- a custom circuit
- the number of PAX in the group

If the correspondence information is known, the groups of PAX can be easily formed considering the *terminal*, the *hall*, the *custom circuit*, and this *rotation*.

Otherwise, a prediction calculates how each flight distributes the connection circuits.

2.6.4 Estimating the shape of PAX presentation curves

Security checkpoints are categorized into three types, based on the passengers going through them: those for local flights, connecting flights, and both. At Paris Airports, managing human resources at the checkpoints involves two levels. The first strategic level predicts passenger flows at the checkpoints 45 days beforehand to determine the necessary number of agents. Minor adjustments can be made 20 days before, and minor changes are possible until the week preceding the flight. However, the later the change is made, the costlier it is. The second level, the tactical level, involves distributing agents in real-time at the checkpoint [MJB⁺20].

When a PAX presents himself at a resource, his presentation time will depend on the nature of the resource and possibly on his status/behavior. For local departure passengers, their appearance follows a “bell-shaped” curve in the hours leading up to the official departure time (SOBT). The characteristics of this curve, including its slopes and spread, are determined by flight data such as flight type, destination, and airline processes. To estimate this curve, AOP employs ML based on FLUXPAX records from the previous 400 days. For arrival passengers, their presence is quickly detected after the actual landing time, plus a period depending on the length of the unboarding circuit. After this, the presentation of passengers is usually evenly spread out

over a fixed period. For connecting passengers, they primarily follow the behavior of an arrival passenger or, in some cases, a departure passenger if the **rotation** is longer.

2.6.5 PAX presentation

From all the flight and pax data collected and learned, the AOP can determine the *passenger presentation* at each resource at any point in time. This involves calculating the *number of passengers* presenting themselves per *time slot* at each resource for a particular day.

From the previous section, we dispose of the following data:

- The number of **PAX** (real or estimated).
- The groups of **connecting pax** (real or estimated).
- The **shape of the curves** for each potential resource (estimated).
- The *terminal* and the *hall*.

However, the missing information is regarding the passenger movement through the different resources. There are three types of passengers to consider:

- *Local departure passengers*: For these passengers, it is necessary to know which *security checkpoint* resource they use, if any, as well as which *border police checkpoint*.
- *Arrival local passengers*: Similarly, for *arrival local passengers* with an arrival *terminal*, an arrival *hall*, and an arrival circuit which *border police checkpoint* is used, if any.
- *Correspondence passengers*: For *correspondence passengers* coming in on a **terminal**, a **hall**, and an *arriving customs route* and departing on a specific *terminal* and *hall* and with a specific *custom route*, the *security checkpoint* resource and *border police checkpoint*, if any.

It is impossible to learn this information through the captured data, as there are no inbound scans and no distinction made between **local passengers** and **correspondence passengers**. Thus, the AOP employs a parameter table to describe the circuits as presented in Table 2.4.

Arr Site	Arr Circ	Dep Site	Dep Circ	FT Status	Res	Prev Res	Type	Ratio	Pres Curve
C2EK	int	C2EK	dom	*	T2EN2		D	1	
		C2EK	schen	*	T2EN2		D	1	
		C2EK	int	*	T2EN2	T2ED	D	1	
		C2EK	*	*	T2EN2		A	1	0,0,1,1,1,1,1,1,1

Table 2.4: An example of circuit configuration for AOP.

Each line of the Table 2.4 describes a group of passengers and is composed of the following columns:

- *Arr site*: an *arrival terminal* and an *arrival hall*.
- *Arr cir*: the custom circuit of the arrival passengers.
- *Dep site*: a *departure terminal* and a *departure hall*.
- *Dep cir*: the custom circuit of the departure passengers.
- *FastTrack status*: yes, no, or whatever.
- *Res*: the name of the *resource* (e.g., *security checkpoint* or *border control police checkpoint*).
- *Prev Res*: the name of the previous resource, if any.

- *Type*: type of curve (e.g., *arrival curve* or *departure curve*).
- *Ratio*: the ratio of passengers following this circuit.
- *Pres Curve*: if present a manual suite of values representing the curve.

For example, the first line indicates that all the passengers (ratio of 100%) boarding at the hall *K* of terminal *2E*, whatever their customs or fast-track status, pass by the security checkpoint *T2EN2*. Moreover, international passengers first pass through the departure border police *T2ED*. Next, the passengers follow the learning departing curves for the security checkpoint *T2EN2*. However, international passengers from terminal *2E*, hall *K* and departing on a flight of any customs status pass through the security checkpoint *T2EN2*. They follow a curve based on arrival flight (*A*) described in the last column. The first two *zeros* indicate two steps without passengers (i.e., 10 minutes without passengers) followed by a constant flow of passengers. The *ratio* column expresses that, in some cases, connecting passengers do not necessarily follow the planned connecting circuit. For example, we can indicate that 80% of passengers follow the connecting circuit and that the remaining 20% exit at the arrival resources and then return to the departure resources.

Now, from the previous tables and all data (real or estimated), we can produce the curve of any resource at any time by summing the *local* curves for each flight on each resource the flight consumes to form the overall curve for the day.

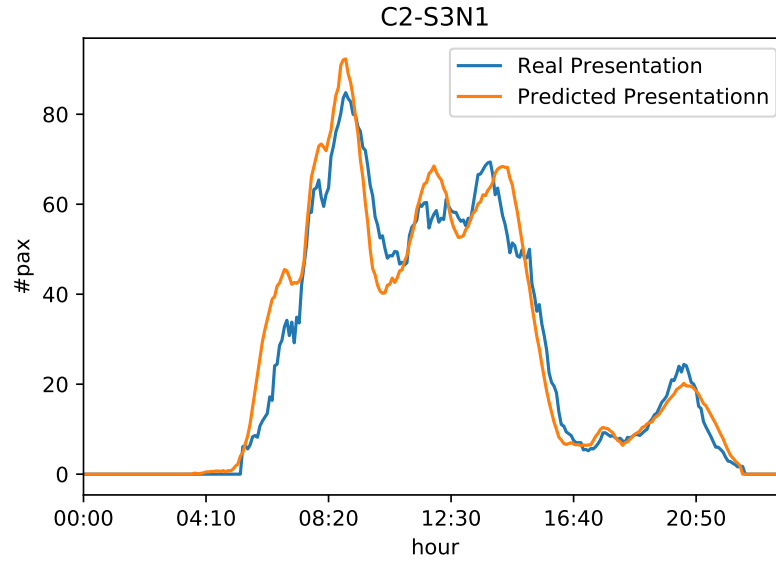


Figure 2.9: Evolution of real and predicted presentation for C2-S3N1.

Figures 2.9 and 2.10 show the predicted curves and the actual curve for two security checkpoints of Terminal 2. As we can see, the predicted curve is relatively close to the actual curve. AOP offers an efficient prediction of airport users' consumption of airport resources.

2.7 Conclusion

This chapter explores the concepts around ML and its application at Paris Airports. Machine learning algorithms can make predictions and draw insights by analyzing vast amounts of flight

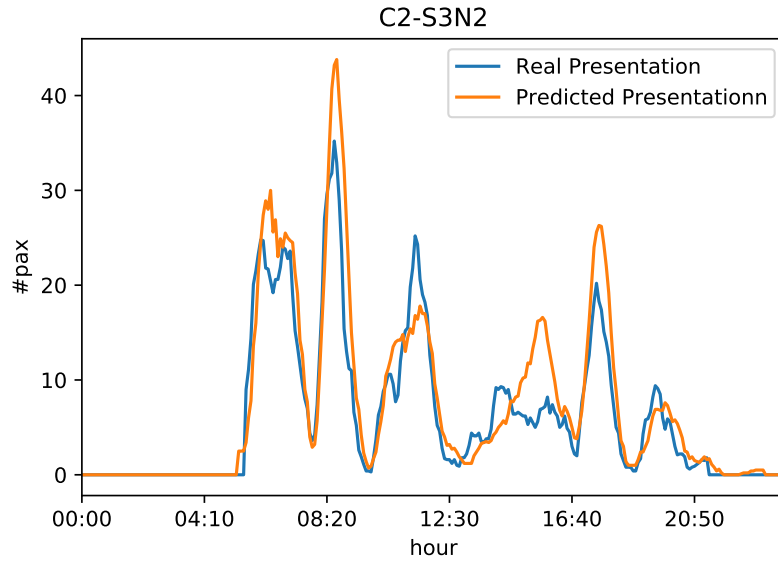


Figure 2.10: Evolution of real and predicted presentation for C2-S3N2.

and passenger data. The AOP system at Paris Airports is an excellent example of this application, as it employs machine learning to calculate passenger presentations at each resource for a given day.

Chapter 6 presents a new application introduced in AOP that involves predicting *passengers with reduced mobility* to better allocate resources and facilitate their travel experience. The second use case, presented in Chapter 7 focuses on predicting *offblock delays* to minimize the impact on passengers and airline operations.

Part II

Optimization of airport resources

Chapter 3

Modeling of the Airport check-in desk assignment problem

Contents

3.1 Introduction	51
3.2 Modeling	52
3.2.1 A first COP formulation	53
3.2.2 Some refinements	59
3.3 Experimental environment	61
3.3.1 Vocabulary	61
3.3.2 Experiments on HPC	62
3.3.3 Experiments on Paris Airport information system	62
3.3.4 Analysis and reproducibility	62
3.4 Experiments	64
3.4.1 Instances	64
3.4.2 Comparison between ACE and ACEURANCETOUIX	65
3.4.3 Modelizations and solver configurations	65
3.4.4 Scoring	67
3.4.5 No-overlapping family	68
3.4.6 Overlapping family	71
3.4.7 Overlapping with a limited number of tasks family	75
3.4.8 Comparison with the ADP algorithm	80
3.5 Conclusion	84

3.1 Introduction

In the world of air travel, airports are crucial for connecting passengers and flights to their desired destinations. As air travel becomes more widespread, optimizing airport operations is crucial. One of the challenges airports face is assigning **check-in desks** to airlines and flights, which is crucial for efficient passenger handling. This task is complex due to various constraints and objectives, such as passenger wait times, resource utilization, and airline satisfaction. As for the previous chapter, the application of **COP** offers a promising solution to address this problem.

This chapter presents different **modeling** for this problem with different methods and **solvers**. We start by formally describing the model developed for the *Check-in Desk Allocation Problem* (CDAP) in a higher ‘**mathematical**’ form. Next, we present experimental environments used for all this part of the manuscript. Finally, we present experiments on different environments.

3.2 Modeling

For modeling **CNO**, (**COP**), several modeling languages or libraries exist such as, e.g., OPL [P. 99], MiniZinc [NSB⁺07, SBF10], Essence [FGJ⁺07] and PyCSP³ [LS20]. Our choice is the recently developed Python library PyCSP³ that permits to generate specific instances (after providing ad hoc data) in XCSP3 format [BLAP16, BLAP20], which is recognized by some well-known CP solvers such as ACE (AbsCon Essence) [Lec23], Oskar [Osc12], Choco [PFL16], and Picat-SAT [ZKF17].

Subsequently, we employ the data itself (for example, ρ) to serve as both the data and their corresponding indexes. Additionally, we adopt the notation $a[i, b]$ or $a[i][b]$ interchangeably to access the cells within a matrix denoted as a . Lastly, we employ the notation $\text{ntask}(\rho)$ to retrieve the count of tasks associated with the rotation ρ (aeronautical term. 2, page. 14).

Table 3.1 recalls all the notations for the problem of check-in desk allocation presented in Section 1.5 (page. 5).

Category	Notation	Description
Check-in	c	Check-in desk.
	$\mathcal{CD}$	The set of all check-in desks.
Zones	z	A zone of check-in desks (i.e., a group of check-in desks)
	$\mathcal{Z}$	The set of all zones.
Registration	ρ	A registration.
	$\mathcal{R}$	The set of all registrations.
	a_r and d_r	The registration’s start and end times.
Tasks	$t_{\rho,i}$	i th task of a registration ρ .
	$\mathcal{T}$	Set of tasks.
	$\mathcal{T}_\rho$	Tasks of the registration ρ
	$\mathcal{CD}_t$	The set of compatible check-in desks for the task t .
Constraints	P	The set of pre-assignment.
	$\mathcal{E}_\rho$	The set of excluded check-in desks for a registration ρ .
	$\mathcal{U}$	The set of unavailable rules.
	$\mathcal{F}$	The set of forbidden overlaps.
	$\mathcal{OR}$	The set of overlapping rules.
	$\mathcal{MC}$	The affinity matrix.

Table 3.1: Notations for the check-in desk allocation problem.

Second, we proceed with the introduction of **variables** in our model. Each **registration** must utilize **check-in desks** following its specific strategy. Instead of considering domains encompassing all possible **banks**, we initially restrict the domains only to include the **banks** that align with the strategy associated with each **registration**. These reduced domains for **check-in desks** compatible with the registration’s strategy are denoted as $\mathcal{D}_{x,\rho}$ for each reg-

istration ρ . Similarly, the domains for **variables** representing **rewards** for airlines are also restricted to values corresponding to the permissible check-in desks. We denote this domain for each registration ρ as $\mathcal{D}_{r,\rho}$. Additionally, we introduce a fictive bank f with a reward value of 0.

We need two (2-dimensional) arrays of **variables** to represent assigned registration and associated rewards:

- x is a matrix of $|\mathcal{R}| \times \nu$ variables having the set of values $\mathcal{D}_{x,\rho}$; $x[\rho][j]$ represents the index (code) of the check-in desk assigned to the j th task of the registration ρ .
- r is a matrix of $|\mathcal{R}| \times \nu$ variables having the set of values $\mathcal{D}_{r,\rho}$; $r[\rho][j]$ represents the satisfaction of the airline for the j th task of the registration ρ .

Third, we need to introduce the constraints in our model. Given the nature of the problem (and data), it is natural to post so-called *extensional constraints*, which explicitly enumerate either the *allowed tuples* (**positive table**) or the *disallowed tuples* (**negative table**) for a sequence of variables (representing the scope of a constraint). Over the last decade, efficient algorithms have been developed to handle such table constraints [Lec11, LLY15, DHL⁺16, VLS17].

3.2.1 A first COP formulation

3.2.1.1 Common constraints

Let us consider the variables previously introduced, the problem of check-in desk allocation can be formulated as follows:

Modelization 1 (Common constraints)

$$x[\rho, j] = c, \forall (\rho, j, c) \in \mathcal{P} \quad (C_1)$$

$$(x[\rho, j] = x[\rho, j+1] - 1) \vee (x[\rho, j] = f \wedge x[\rho, j+1] = f), \forall \rho \in \mathcal{R}, \forall j \in \text{ntask}(\rho) \quad (C_2)$$

$$(x[\rho_1, i] \neq x[\rho_2, j]) \vee (x[\rho_2, j] = f), \forall \rho_1, \rho_2 \in \mathcal{F}, \forall i \in \text{ntask}(\rho_1), \forall j \in \text{ntask}(\rho_2) \quad (C_3)$$

$$x[\rho, t], r[\rho, t] \in \{(c, m_{t,c}), \forall (c, m_{t,c}) \in \mathcal{MC}, \forall t \in \mathcal{T}_\rho, \forall \rho \in \mathcal{R}\} \cup \{f, 0\} \quad (C_4)$$

Constraints C_1 ensure that each pre-assignment of $\mathcal{P}$ is respected. Constraints C_2 ensure that the chosen check-in desks for registration are consecutive or use the fictive check-in desk for each registration task (see Rule 5, page. 25). The introduction of holes in the domains (e.g., *useless check-in desks*) makes it possible to manage this by imposing that a task must be equal to the following task minus one and not including useless check-in desks in the domain. In this way, we insert a hole representing the *zone's separation*. Constraints C_3 prevent two overlapping registrations from being assigned to the same check-in desk (as presented by Rule 7, page. 25). Constraints C_4 use **table constraint** to map the check-in desk with this weight. We use the affinity matrix defined previously.

Example 19

Consider a set of five registrations, represented as $\mathcal{R} = \{\rho_1, \dots, \rho_5\}$, and a group of eight check-in desks, denoted as $\mathcal{CD} = \{c_1, \dots, c_8\}$.

Two zones are identified: z_1 contains desks from c_1 to c_4 , and z_2 encompasses desks from c_5 to c_8 ($\mathcal{Z} = \{z_1, z_2\}$).

For ρ_1 and ρ_2 , we have three tasks, whereas for ρ_3 , ρ_4 , and ρ_5 , there are two tasks. The associated time windows for each registration are visualized in Figure 3.1. The affinity matrix is provided in Table 3.2.

Based on the given data, we introduce:

- The matrix x of dimension 5×3 . Here, five signifies the total registrations, and three represents the maximum number of tasks in a given registration. Each variable has a consistent domain: $\mathcal{D}_x = \{1, \dots, 9, 10\}$. The initial values, ranging from 1 to 4, correspond to desks c_1 through c_4 . The value 5 is a gap between the zones z_1 and z_2 . Subsequent values, 6 through 9, map to desks c_5 to c_8 . The value 10 designates an auxiliary check-in desk.
- The matrix r has identical dimensions, with a domain ranging from 0 to 100, represented as $\mathcal{D}_r = \{0, \dots, 100\}$.

Considering a pre-assignment $\mathcal{P}$ for ρ_1 , and adhering to the constraints from Modeling 1, we have:

$$\begin{aligned} x[\rho_1, 1] &= 1 \\ x[\rho_1, 2] &= 2 \\ x[\rho_1, 3] &= 3 \end{aligned}$$

Incorporating the constraint C_2 for registration ρ_1 , we derive:

$$\begin{aligned} (x[\rho_1, 1] = x[\rho_1, 2] - 1) \vee (x[\rho_1, 1] = 10 \wedge x[\rho_1, 2] = 10) \\ (x[\rho_1, 2] = x[\rho_1, 3] - 1) \vee (x[\rho_1, 2] = 10 \wedge x[\rho_1, 3] = 10) \end{aligned}$$

For other registrations, the same procedure is applied. Given the overlap between tasks of ρ_1 with those of ρ_2 and ρ_4 , additional constraints are stipulated:

$$\begin{aligned} (x[\rho_1, 1] \neq x[\rho_2, 1]) \wedge (x[\rho_1, 2] \neq x[\rho_2, 1]) \wedge (x[\rho_1, 3] \neq x[\rho_2, 1]) \wedge \\ (x[\rho_1, 1] \neq x[\rho_2, 2]) \wedge (x[\rho_1, 2] \neq x[\rho_2, 2]) \wedge (x[\rho_1, 3] \neq x[\rho_2, 2]) \wedge \end{aligned}$$

$$\begin{aligned} (x[\rho_1, 1] \neq x[\rho_4, 3]) \wedge (x[\rho_1, 2] \neq x[\rho_4, 3]) \wedge (x[\rho_1, 3] \neq x[\rho_4, 3]) \wedge \\ (x[\rho_1, 1] \neq x[\rho_4, 1]) \wedge (x[\rho_1, 2] \neq x[\rho_4, 1]) \wedge (x[\rho_1, 3] \neq x[\rho_4, 1]) \wedge \\ (x[\rho_1, 1] \neq x[\rho_4, 2]) \wedge (x[\rho_1, 2] \neq x[\rho_4, 2]) \wedge (x[\rho_1, 3] \neq x[\rho_4, 2]) \end{aligned}$$

Finally, using the constraint C_4 and referring to the affinity matrix from Table 3.2, we post:

$$x[\rho_1, 1], r[\rho_1, 1] \in \{(1, 90), (2, 80), (3, 70), (4, 20), (6, 20), (7, 20), (8, 20), (9, 20), (10, 0)\}$$

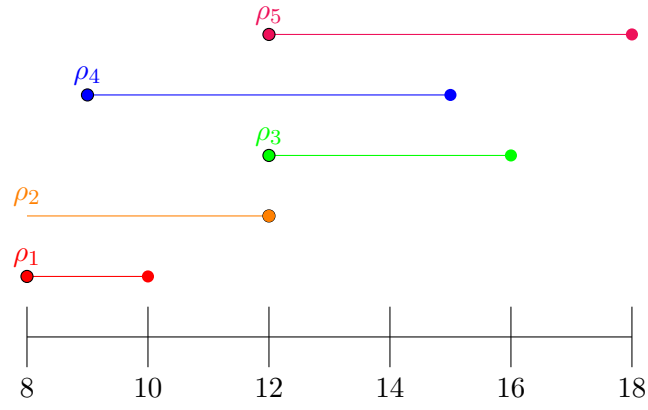


Figure 3.1: Time intervals for check-in desk allocation example.

	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5
c_1	90	20	85	65	10
c_2	80	20	75	60	10
c_3	70	20	55	50	20
c_4	20	80	35	60	30
c_5	20	60	40	35	45
c_6	20	55	45	30	60
c_7	20	20	30	25	90
c_8	20	20	20	20	85
f	0	0	0	0	0

Table 3.2: Rewards for Check-in Desk Assignments

Some constraints and variables are only present if the associated rules are in the data. This is the case for overlap rules with a limited number of tasks, exclusion constraints, and unavailability constraints. These constraints are developed in the following sections.

3.2.1.2 Overlap rules with a limited number of tasks

Modelization 2 (Overlap rules with a limited number of tasks)

$$\begin{aligned}
 od[or, \rho, b, \rho_1] = 0 \Leftrightarrow & \bigwedge_{bb=0}^{\mathbf{ntasks}(\rho_1)} (x[\rho, b] \neq x[n, bb]) \vee x[\rho, b] = f, \\
 & \forall or \in \mathcal{OR}, \\
 & \forall \rho \in \mathcal{R}, \\
 & \forall b \in \mathbf{ntask}(\rho), \\
 & \forall \rho_1 \in \mathcal{N}_{\rho, or}
 \end{aligned} \tag{C_5}$$

$$\begin{aligned}
 \text{Sum}(\{od[or, \rho, b, \rho_1] > 0, \forall \rho_1 \in \mathcal{N}_{\rho, or}, \forall b \in \mathbf{ntasks}(\rho)\}) \leq m, \\
 & \forall or \in \mathcal{OR}, \\
 & \forall \rho \in \mathcal{R}
 \end{aligned} \tag{C_6}$$

As detailed earlier, overlapping rules can be set up to tolerate registrations using the same check-in desk. There are three methods of implementing these rules for two registrations:

- if no rule exists between these registrations, or if a rule exists but is incompatible with the overlap period, then overlap is prohibited using a non-overlap constraint (see Rule C_3).
- if a rule exists as a pair (i.e., without specifying the m in the rules), overlapping is tolerated, and no constraint is added.
- if a rule exists as a triplet, specific constraints are added to represent this particular case.

We add a new matrix of $|\mathcal{OR}| \times |\mathcal{R}| \times |\nu| \times |n|$ variables called od . n is the maximum number of possible neighbors (i.e., $\max(\{|\mathcal{N}_{\rho, or}|, \forall or \in \mathcal{OR}, \forall \rho \in \mathcal{R}_{or}\})$). The domain of a variable $od[or, \rho, b, \rho_1]$ is a binary domain composed of the value 0 and the overlapping duration between the registration ρ and ρ_1 considering the rule or .

Constraints C_5 ensure that the overlap time between a registration ρ and one of its neighboring registration n is equal to 0 if and only if the two tasks use different check-in desks or one of them uses the dummy bank. Based on the assignment in the x matrix, this constraint is used to determine whether two registrations overlap or not. We now must introduce constraints considering the maximum number of overlaps possible on a registration.

In Constraint C_6 , $od[or, \rho, b, \rho_1] > 0$ is true when the registrations ρ and ρ_1 overlap on task b of ρ . We ensure that the sum of these booleans for each task b of a registration ρ and the set of tasks of these neighbors is less than or equal to m from the or rule.

Example 20

Suppose we have overlapping rules, denoted as or , that allow a maximum overlap of 2 hours between two tasks from registrations ρ_2 and ρ_4 . This rule can be represented as the triplet $or = (\rho_2, \rho_4, 120, 2)$. Here, the set ρ_2, ρ_4 indicates the registrations subject to this overlapping rule. To express this rule in the context of our initial overlapping constraint definition, we can form a matrix, od , with dimensions $1 \times 5 \times 3 \times 1$:

- The first dimension is 1, signifying the single overlapping rule.
- The second dimension represents the total number of registrations in our example, 5.
- The third dimension relates to the maximum number of tasks associated with a registration.
- Lastly, the fourth dimension indicates the maximum count of neighboring registrations. In this instance, it is equivalent to $\max(\{|\mathcal{N}_{\rho_2, or}|\})$, which equals 1, given that $\mathcal{N}_{\rho_2, or}$, comprises just ρ_4 .

$$\begin{aligned}
 od[1, 2, 1, 1] &= 0 \Leftrightarrow \bigwedge_{bb=1}^2 (x[2, 1] \neq x[4, bb]) \vee x[2, 1] = 10, \\
 od[1, 2, 2, 1] &= 0 \Leftrightarrow \bigwedge_{bb=1}^2 (x[2, 2] \neq x[4, bb]) \vee x[2, 2] = 10, \\
 od[1, 2, 3, 1] &= 0 \Leftrightarrow \bigwedge_{bb=1}^2 (x[2, 3] \neq x[4, bb]) \vee x[2, 3] = 10,
 \end{aligned}$$

For the previous constraints :

- They are associated with the first overlapping rule, represented by the index 1 in the matrix od 's first dimension.
- They are related to registration ρ_2 , signified by the index 2 in od 's second dimension.
- Each constraint corresponds to a specific task of ρ_2 , denoted by an index in the matrix od 's third dimension.
- The concluding index designates the primary neighbor (index 1) of ρ_2 , which is ρ_4 .

For each task of ρ_2 we ensure that the overlap time with each task of ρ_4 is equal to 0 if and only if variables for the tasks use different check-in desks or one of them uses the dummy bank.

Regarding the Constraints C_6 , we count for each registration of each rule the number of variables with a value strictly greater than 0, and we ensure that this sum is less than the value defined by the rule.

3.2.1.3 Exclusion constraints and Unavailability constraints

Modelization 3 (Exclusion constraints)

$$x[\rho, i] \notin \mathcal{E}_\rho, \forall \rho \in \mathcal{R}, \forall i \in \text{ntasks}(\rho) \quad (C_7)$$

Modelization 4 (Unavailability constraints)

$$x[\rho, i] \neq c, \forall \rho \in \mathcal{U}_c \quad (C_8)$$

In Constraint C_7 , we post for each task of each registration a negative table that excluded the check-in desk from $\mathcal{E}_\rho$. For Constraint C_8 , we introduce the set of registrations that overlap with an unavailability rule u (e.g., $\mathcal{U}_c = \{\rho, \forall \rho \in \mathcal{R} \mid \text{overlap}(\rho, u)\}$). For each created set, we add an inequality between each registration task and the rule's check-in desk.

Example 21

Building upon our preceding example, let's consider a scenario where certain check-in desks are excluded for the registration ρ_1 , represented by the set $\mathcal{E}_{\rho_1}$. To encapsulate this, we introduce negative table constraints (see Constraint C_7) for each task associated with ρ_1 as follows:

$$\begin{aligned} x[\rho_1, 1] &\notin \mathcal{E}_{\rho_1} \\ x[\rho_1, 2] &\notin \mathcal{E}_{\rho_1} \\ x[\rho_1, 3] &\notin \mathcal{E}_{\rho_1} \end{aligned}$$

This ensures that the tasks of ρ_1 do not get assigned to the restricted check-in desks listed in $\mathcal{E}_{\rho_1}$.

Now, if check-in desk c_1 is unavailable and this unavailability overlaps with ρ_4 , we can assert the following constraints:

$$\begin{aligned} x[\rho_4, 1] &\neq 1 \\ x[\rho_4, 2] &\neq 1 \end{aligned}$$

3.2.1.4 Objective Function

Finally, we can use the matrix r for posting the objective function that maximizes the satisfaction of the airlines:

Modelization 5 (Objective function)

$$\text{maximize} \quad \sum_{\substack{\rho \in \mathcal{R} \\ j \in 1..\text{ntasks}(\rho)}} r[\rho, j] \quad (C_9)$$

3.2.2 Some refinements**3.2.2.1 Gathering Binary Difference Constraints**

We will now strengthen this natural formulation by reformulating the set of constraints (C_3) using the `AllDifferentExcept` constraint. This latter enforces all variables to take distinct values, except those assigned to a special (joker) value (here it is our fictive bank f).

$$\text{AllDifferentExcept}(\{x[\rho_1], x[\rho_2]\}, f) \quad (C_{10})$$

For each pair ρ_1, ρ_2 in the forbidden overlaps $\mathcal{F}$. Note that we used the notation $x[\rho_1]$ and $x[\rho_2]$ for a shortcut that integrates the entire second dimension of the matrix into the constraint (i.e., each task of ρ_1 or ρ_2). This formulation considerably reduces the number of constraints about no-overlapping tasks, as the previous formulation needs a quadratic number of `not-equal` constraints.

Note that there are several possible representations or propagators in the `ACE` solver to manage this constraint:

- `AllDifferentExcept` can be represented with a `Cardinality` constraint where each value (except the excluded value) from the union of the domain of the scope of the constraint must be present at most once (called `Card`).
- `AllDifferentExcept` can be represented by decomposing the constraint into binary constraints (called `Bin`).
- `AllDifferentExcept` can use a weak propagator that not preserve the arc-consistency (called `ADEWeak`)

Example 22

Building on the scenarios presented in Example 19, we can refine the inequalities and introduce the subsequent constraints:

$$\begin{aligned} &\text{AllDifferentExcept}(\{x[\rho_1], x[\rho_2]\}, 10) \\ &\text{AllDifferentExcept}(\{x[\rho_1], x[\rho_4]\}, 10) \end{aligned}$$

3.2.2.2 Gathering *AllDifferentExcept* constraints

Even though the formulation above notably reduces the number of posted constraints, the solver remains too slow to find acceptable results (bounds) in a reasonable amount of time. We have

thus gathered all `AllDifferentExcept` constraints into a unique, pragmatic constraint called `GatherAllDifferentExcept`. For this particular constraint, we use a specific fast propagator (see Algorithm 1) that performs a limited form of filtering (i.e., does not enforce *generalized arc consistency*). This is a very pragmatic approach, equivalent to the initial set of binary constraints but faster (only one constraint being posted). We use a data structure that allows for two variables x and y to check whether x and y appear conjointly in the same `AllDifferentExcept` constraint. When x is assigned, we can thus find easily the variables to filter.

Algorithm 1: Propagator for `GatherAllDifferent`

```

Input:  $x$  // A variable  $x$  to filter.
Output: boolean // Returns false if an inconsistency is detected

1 if  $\text{dom}(x).size() = 1$  then
2    $v \leftarrow x.\text{dom}.\text{singleValue}()$ 
3   if  $v = \text{exceptValue}$  then
4     return true
5   end
6    $t \leftarrow \text{neighboursOf}(x)$ 
7   if  $\text{futvars}.size() < t.size()$  then
8      $\text{tab} \leftarrow \text{futvars}$ 
9   else
10     $\text{tab} \leftarrow t$ 
11  end
12  for  $i = \text{tab}.size() - 1$  to 0 do
13     $y \leftarrow \text{tab}[i]$ 
14    if  $(y \neq x) \wedge \text{sharedConstraint}(x, y) \wedge \text{not } y.\text{dom}.\text{removeValueIfPresent}(v)$ 
15      then
16        return false
17    end
18  end
19 return true

```

3.2.2.3 Refinement of the Overlapping constraint

To reduce the dimensions of the od matrix, an alternative approach involves utilizing not the cardinality of $\mathcal{R}$ for the second dimension, but only the size of the set of registrations governed by rules. This size is represented by the union of all registrations involved in the rules, denoted as $\mathcal{R}_{OR} = \bigcup_{\forall \mathcal{R}_{or} \in \mathcal{OR}} \mathcal{R}_{or}$. We can also use a true binary domain (i.e., $\{0,1\}$) to avoid the reification constraint introduced by $od[or, \rho, b, \rho_1] > 0$. Consequently, Constraint C_6 can be reformulated in two ways.

First, in Constraint C_{11} , we exploit the binary domain to sum the variables directly. This eliminates the need for reification and results in the following form:

Modelization 6 (Sum)

$$\begin{aligned}
\text{Sum}(\{od[or, \rho, b, \rho_1], \forall \rho_1 \in \mathcal{N}_{\rho, or}, \forall b \in \text{ntasks}(\rho)\}) &\leq m, \\
\forall or \in \mathcal{OR}, & \\
\forall \rho \in \mathcal{R} &
\end{aligned} \tag{C_{11}}$$

Alternatively, in Constraint C_{12} , we make use of a **Count** constraint along with the binary domain, taking advantage of the associated propagator.

Modelization 7 (Count)

$$\begin{aligned}
\text{Count}(\{od[or, \rho, b, \rho_1], \forall \rho_1 \in \mathcal{N}_{\rho, or}, \forall b \in \text{nt}(\rho)\}, v = 1) &\leq m, \\
\forall or \in \mathcal{OR}, & \\
\forall \rho \in \mathcal{R} &
\end{aligned} \tag{C_{12}}$$

3.3 Experimental environment

3.3.1 Vocabulary

This section introduces some of the vocabulary used to identify the data manipulated during the various campaigns in the chapters that follow. A **campaign** contains all the **experimental data** that has been collected and defines the configuration of the **solver execution environment** (temporal and spatial boundaries, machine configuration, etc.). During a campaign, **solvers** are evaluated. Note that different configurations of the same **solver** are considered different **solvers**. A **campaign** is characterized by the set of **input files** used for it. In this context, all **solvers** are run on the same set of files. Finally, an experiment corresponds to the execution of a given **solver** on a given **input file** so that the set of experiments correspond to the Cartesian product of the **set of input files** and the **set of solvers**. Each experiment is characterized by those **measurements** that are relevant to the analyses to be carried out, such as execution time or memory used by the **solver** (and many others, depending on the application).

Example 23

Let us consider a **campaign** in which we want to compare the solvers S^1 and S^2 on two instances `CarSequencing.xml` and `RLFAP.xml`^a. The set of **input files** comprises these two files. The **experiments** for this campaign are as follows:

- The execution of S^1 on `CarSequencing.xml` ;
- The execution of S^1 on `RLFAP.xml` ;
- The execution of S^2 on `CarSequencing.xml` ;
- The execution of S^2 on `RLFAP.xml` ;

^aWe will see later what these ‘XML’ files correspond to.

3.3.2 Experiments on HPC

All the campaigns described in this manuscript are executed on the same conditions. We dispose of a cluster of computers equipped with 128 GB of RAM and two quad-core Intel XEON E5-2637 (3.5 GHz); for a specific node, only one **experiment** is run at a time. When we used instances from the XCSP library, the time limit was set at 2400 seconds (time commonly used in XCSP competitions [BLAP20]), otherwise the time is set to 300 seconds, the time limit fixed by Paris Airports. When the associated **campaign** is presented, a description of the **instances** used will



be provided. The experiments executed on the cluster are illustrated by the logo

3.3.3 Experiments on Paris Airport information system

Concerning the campaigns executed on the Paris Airports Information System, we dispose of a server equipped with 64 GB of RAM and two 10-core Intel Xeon Silver 4210R (2.40GHz). Except in the case of experiments designed to test load, each experiment was run alone on the server (i.e., without any other parallel resolution). Note that the solver is stopped when no more improvement has been made for 5 seconds (since the last solution was found). Since the choice of stopping the solver after 5 seconds makes it non-deterministic, we run each experiment 10 times.

3.3.4 Analysis and reproducibility

In the context of computer science research (and more generally, in any field requiring the design of software programs), it is necessary to carry out experiments to ensure that the produced programs work as intended. In particular, one needs to ensure that its resources remain reasonable. To do so, software solutions such as *run solver* [Rou11] have been developed to measure and limit the consumption of the temporal and spatial resources of the program under test. However, more than respecting these limits is generally required to assess the program’s behavior. Collecting additional statistics, usually provided by the program (e.g., via software logs) or tools such as *runsolver* is often necessary. The collected data must then be aggregated to evaluate the quality of the program’s results through a statistical analysis.

Statistics provide many mathematical tools, and choosing one over another may introduce biases in the results or their analysis. Thus, many erroneous analyses have been identified over

the years. One of the most famous such analyses is an article on economics by Reinhart and Rogoff [RR10], for which analytical errors were detected only three years later [HAP14]. As a countermeasure to such errors, the principles of *transparent science* and *reproducible results* are increasingly being applied. They have been the subject of an OECD recommendation [PF07], and scientists have introduced many approaches favoring reproducibility in the context of computer experiments [FFR16, KPD18]. Towards this direction, it is recommended to open the source of the software program (or, at least, provide its binaries) and to make available the data used to evaluate it (e.g., using software forges). It is also important to analyze the results reproducibly, which can be done using tools such as RMarkdown⁹ or Jupyter Notebooks¹⁰.

In the research community of *problem solving* (e.g., in the *SATisfiability Problem* (SAT), *Pseudo-Boolean* (PB), *Constraint Programming* (CP) or *Quantified Boolean Formula* (QBF) communities), there is not a significant difference between how the different solvers are executed. Indeed, solvers are often required to provide *Command Line Interface* (CLI) that meet the environment requirements in which they are being executed (e.g., during competitions). As such, the main difference between these programs is their actual implementation. Also, most of the data collected when running the solvers remains almost the same (e.g., runtime, memory usage, etc.). In this context, the creation of a tool that can run the program, collect the data it produces, and analyze it would have multiple advantages: testing new features is more straightforward, both in terms of execution and analysis, and the reproducibility of the results is automatically ensured.

Based on these observations, we have developed *Metrics* [FWW21] (*mETRICS* stands for *rEproducible softWare peRformance analysIs in perfeCt Simplicity*), a Python library aiming to unify and make easier the analysis of solver experiments. Metrics aims to provide a complete toolchain from the execution of the solver to the analysis of its performance. Currently, this library contains two main components: *scalpel* (*sCALPEL* stands for *extraCting dAta of exPeriments from softwarE Logs*) and *wallet* (*wALLET* stands for *Automated tooL for expLoiting Experimental resulTs*). On the one hand, *scalpel*, is designed to simplify the retrieval of experimental data. It can handle a wide variety of inputs, including CSV, XML, JSON files or even the solver's output, thanks to a description file provided by the user. This makes the tool easy to configure and highly flexible. On the other hand, *wallet* provides a nice interface for drawing commonly used plots (such as scatter or cactus plots) and computing statistics about the execution of the different solvers (in particular, their score using classical performance measures). The design of *wallet* makes easier the integration of the analysis in Jupyter Notebooks, which can easily be shared online (for instance, GitHub or Gitlab can render such files), which also favors the reproducibility of the analysis.

In the upcoming chapters, statistics, tables, or graphs produced for each campaign are accompanied by a clickable logo  that directs to the corresponding campaign analysis. A campaign includes all the information required to enable the reader to replicate the experiments, data extraction, analysis and production of the figures presented in this thesis. Moreover, we use the open-source tool RunSolver¹¹ to control the resources used by our solvers or models.

⁹<https://rmarkdown.rstudio.com/>

¹⁰<https://jupyter.org/>

¹¹<http://www.cril.univ-artois.fr/~roussel/runsolver/>

3.4 Experiments

Within this section, we undertake a variety of experiments to compare the proposed models and explore various configurations for the ACE solver.

3.4.1 Instances

Airport	Terminals	Date		#Flights	#Tasks	#OR	#OR Nbmax
ORY	1,2,3,4	2023-05-08	2023-05-14	2285	3903	21	21
ORY	1,2,3,4	2023-06-19	2023-06-25	1559	2966	22	22
ORY	1,2,3,4	2023-06-26	2023-07-02	1669	3212	22	22
ORY	1,2,3,4	2023-07-03	2023-07-09	1686	3267	22	22
CDG	T2B T2D	2023-07-03	2023-07-09	767	1672	0	0
CDG	T1	2023-07-03	2023-07-09	647	2120	8	8
CDG	T1,T2,T3	2023-08-21	2023-08-27	1811	5077	13	0
CDG	T1,T2,T3	2023-09-11	2023-09-17	4274	10298	14	0
CDG	T1,T2,T3	2023-09-18	2023-09-24	4278	10216	14	0

Table 3.3: General information about the check-in planning.

Table 3.3 presents some factual information about the different planning used for these experiments. The first two columns indicate the area of the planning (i.e., **Airport** and **Terminals** concerning the planning). The third column gives the date of the planning. Finally, the last two columns display the number of flights and tasks. For each planning, we have created three families of problems:

- **no-overlapping**: This family does not contain overlapping rules. Therefore, the constraints presented in Section 3.2.1.2 are absent.
- **overlapping**: In this family, there are overlapping rules without a maximum number of tasks. This family will be similar to the **no-overlapping** family but with fewer **AllDifferentExcept** constraints.
- **overlapping-nbmax**: The overlapping rules contain a maximum number of tasks in this family. Note that this family contains the various models proposed for the constraint overlapping with a limited number of tasks (i.e., modeling 2, 6 and 7).

For our experiments, we also compare three types of decomposition. The first type of decomposition is **no-decomposition**, which gathers all the planning in a unique instance. The second decomposition is **partial-decomposition**, which keeps terminals together and decomposes by day. Finally, the third decomposition decomposes by terminals and by day and is called **full-decomposition**. For decompose by day, we exploit the **strategy rules** (i.e., airline satisfaction). By default, terminals are separated but grouped if a strategy covers several terminals.

The first set of instances $\mathcal{I}_1$ is composed of all families of instances and represents 1541 instances:

$$\left\{ \mathcal{I}_t^f \mid \begin{array}{l} f \in \{\text{no-overlapping}, \text{overlapping}, \text{overlapping-nbmax}\} \\ t \in \{\text{checkin}\} \end{array} \right\} \quad (\mathcal{I}_1)$$

3.4.2 Comparison between ACE and ACEURANCETOUIX

For technical reasons, we do not use ACE directly in the Paris Airport environment, but its JUniverse¹² adapter, ACEURANCETOUIX¹³. The main difference between ACE and ACEURANCETOUIX is how to read the XCSP file. This first campaign compares ACE and ACEURANCETOUIX to show that the cost of using ACEURANCETOUIX is negligible and used the solver configurations defined by the set Ψ_1 . Note that we have activated the annotation for each configuration to guide which variables must be assigned first.

$$\left\{ \begin{array}{l} \mathcal{S} \in \{\text{ACE}, \text{ACEURANCETOUIX}\} \\ \mathcal{S}_{valh}^{varh} \\ varh \in \{\text{Frba}\} \\ valh \in \{\text{First}\} \end{array} \right\} \quad (\Psi_1)$$

Summary of the experiments 1 (ACE vs ACEURANCETOUIX - , ,)

	$\mathcal{I}_1$
	Ψ_1
	5 minutes
	32 GB

	Linux CentOS Stream 8.3
	Two quadcore Intel XEON E5-2637 (3.5 GHz)
	128 GB

Figures 3.2 and 3.3 show scatter plots comparing the value of the first bound found and the time to found first bound between ACE and ACEURANCETOUIX respectively. These scatter plots show that ACEURANCETOUIX found precisely the same first bound as ACE with an often better time. Next, we will only use ACEURANCETOUIX.

3.4.3 Modelizations and solver configurations

This section introduces experiments that evaluated our proposed model using various solver configurations. Because the different families are not comparable, we separate each analysis of each family into a specific section.

We shall primarily utilize tables for these experiments to display the initial (respectively, final) bound and the corresponding time for each configuration and instance. Upon decomposing the instances, it becomes imperative to aggregate each sub-problem's bounds to ascertain the entire problem's bound. Using the illustration provided in Figure 3.4, if seven solvers produce bounds at different times (*black point*), and the fourth solver is the last to generate a bound, and we

¹²<https://github.com/crillab/juniverse>

¹³<https://github.com/crillab/aceurancetourix>

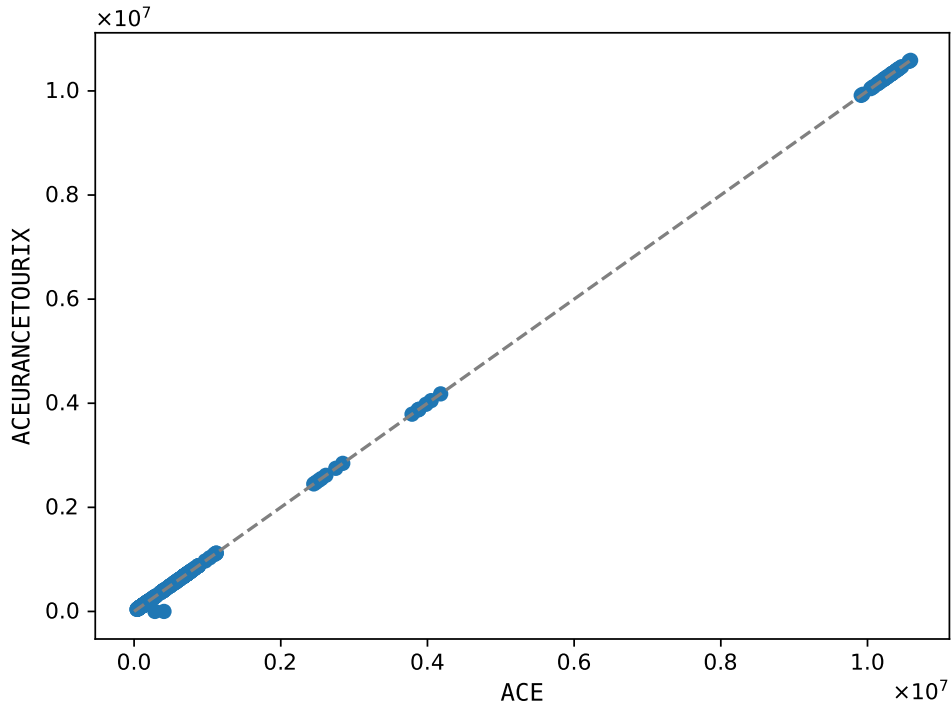


Figure 3.2: Comparison of first bound found by ACE and ACEURANCETOURIX.

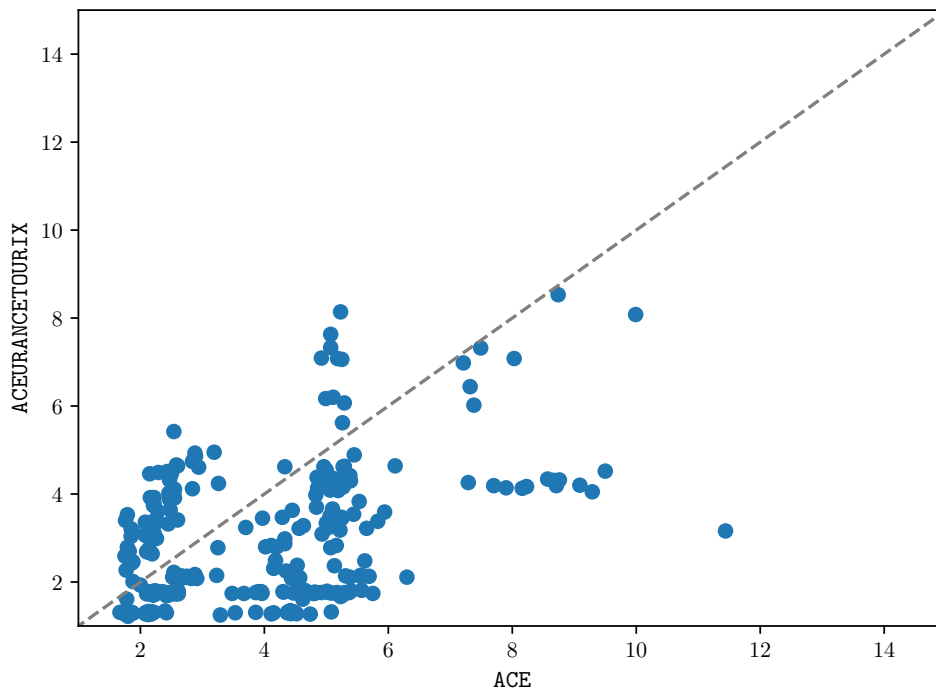


Figure 3.3: Comparison of the time to find the first bound between ACE and ACEURANCETOURIX.

accumulate the final bounds produced by each solver up to, or just before, the moment when the fourth solver's bound is established (*red point*).

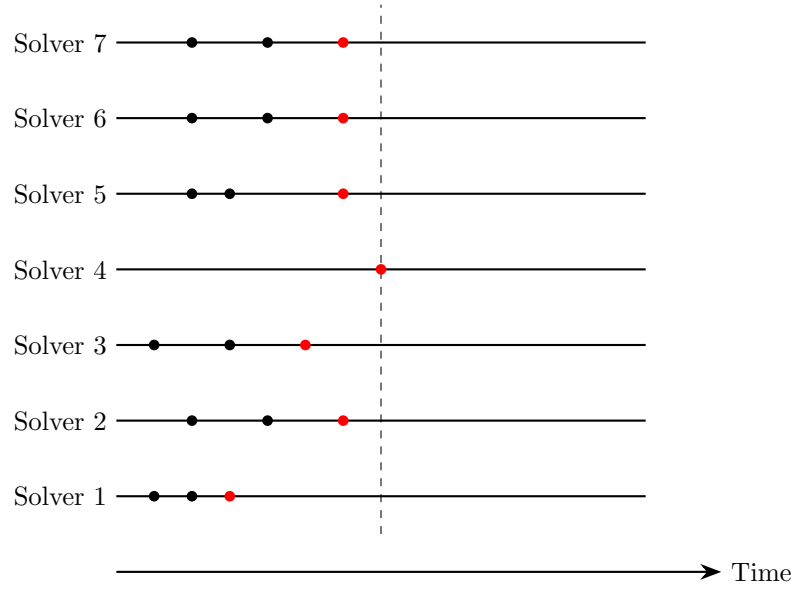


Figure 3.4: Example of the aggregation of bounds to find the global bound when the problem is decomposed.

For these experiments, we use the set of solvers defined as follows:

$$\left\{ \begin{array}{l} \text{decompo} - \text{ade} - \text{ale}_{\text{valh}}^{\text{varh}} \\ \text{decompo} \in \{\text{Full}, \text{Part}, \text{None}\} \\ \text{ade} \in \{\text{Card}, \text{Bin}, \text{ADEWeak}, \text{Gather}\} \\ \text{ale} \in \{\text{i2e}, \emptyset\} \\ \text{varh} \in \{\text{Frba}, \text{Wdeg}\} \\ \text{valh} \in \{\text{First}, \text{SO}, \text{Bivs}, \text{Bivs} + \text{SO}\} \end{array} \right\} \quad (\Psi_2)$$

Where **decompo** is the decomposition used for solving the instance, **ade** corresponds to the algorithm for the **AllDifferentExcept** constraint. **i2e** for **ale** indicates that *intension constraints* have been converted to *extension constraints* (when this is the case, the maximum arity of converted constraints is 4). **varh** is the variable order heuristic used by the solver (see Section 1.3.3.1, page. 13). Finally, **valh** is the value order heuristic used by the solver, **First** and **Bivs** [FP17] are the heuristics described in Section 1.3.3.2, page. 13. The designation **SO** stands for *Static Order*, a specialized value order heuristic wherein the check-in desk is sequenced according to this weight from strategy. The term **Bivs** + **SO** illustrates a fusion of both **Bivs** and **SO**.

3.4.4 Scoring

To select the best solvers presented, we define a score that allows us to keep only the best. Recall that all results are available online. First, we present the different notations we use to evaluate experimental results. Given a set $\mathcal{I}$ of instances and a set Ψ of solvers,

$$b_{s,t}^i$$

corresponds to the best bound obtained by solver $s \in \Psi$ on instance $i \in \mathcal{I}$ in t second(s), where $t \in [0, \dots, T]$ and T is the timeout. Note that we consider only the time for each solver of the

first bound (t_f) and the last bound t_l . We use a similar notation to refer to the time taken by solver s to produce its first bound (resp. last bound) on instance i :

$$t_s^i$$

We can now define the value corresponding to the largest (best) bounds obtained by a set of solvers Ψ on a given instance i at time t where t is equal to t_f or t_l .

$$\text{maxb}_t^i = \max_{s \in \Psi} b_{s,t}^i$$

We also define the notation for the minimum time taken by a set of solvers Ψ to find the first solution of an instance i .

$$\text{mint}_t^i = \min_{s \in \Psi} t_s^i$$

We can now present the score obtained by a solver $s \in \Psi$ on an instance $i \in \mathcal{I}$ at time t where t is equal to t_f or t_l :

$$n_{s,t}^i = \begin{cases} 1, & \text{if } b_{s,t}^i = \text{maxb}_t^i \wedge t_s^i = \text{mint}_t^i \\ 0.5, & \text{if } b_{s,t}^i = \text{maxb}_t^i \\ 0 & \text{else} \end{cases}$$

Finally, the score obtained by the solvers is defined as follows:

$$n_s = \sum_{i \in \mathcal{I}} n_{s,t_f}^i + \sum_{i \in \mathcal{I}} n_{s,t_l}^i \quad (3.1)$$

3.4.5 No-overlapping family

This section concerns experiments on the family of instances **no – overlapping**. We define a new set of instances for this family noted $\mathcal{I}_2$, a subset of $\mathcal{I}_1$. It is important to note that the results are for the instances presented in the Table 3.3, and that we are therefore applying the aggregation procedure (when we have decomposed the instance) described in Section 3.4.3.

$$\{\mathcal{I}_{\text{no-overlapping}}^{\text{checkin}}\} \quad (\mathcal{I}_2)$$

Summary of the experiments 2 (no – overlapping - , ,)

$\mathcal{I}_2$
 Ψ_2
5 minutes
32 GB

Linux CentOS Stream 8.3
Two quadcore Intel XEON E5-2637 (3.5 GHz)
128 GB

Solver	Score
Part – Gather $_{SO}^{Frba}$	2.5
Full – Gather $_{SO}^{Wdeg}$	2.0
Full – ADEWeak – $i2e_{SO}^{Wdeg}$	1.5
Full – Bin – $i2e_{SO}^{Wdeg}$	1.5
None – ADEWeak – $i2e_{SO}^{Frba}$	1.0
None – Bin – $i2e_{SO}^{Frba}$	1.0
None – Card – $i2e_{SO}^{Frba}$	1.0
None – Gather – $i2e_{First}^{Frba}$	1.0
Part – ADEWeak $_{SO}^{Wdeg}$	1.0
Full – Card – $i2e_{SO}^{Wdeg}$	0.5
Part – Bin – $i2e_{Bivs}^{Frba}$	0.5
Part – Card – $i2e_{First}^{Frba}$	0.5

Table 3.4: Ranking of the solvers for no – overlapping family.

Table 3.4 displays the scores derived from equation 3.1 for each solver, emphasizing the optimal pairs formed by decomposition and the **AllDifferent** algorithm configurations. Consequently, each **AllDifferent** algorithm appears thrice, once for every decomposition.

We can notice that the 2 solvers occupying the first places are based on the **Gather** method for **AllDiffExcept** constraint presented in Section 3.2.2.2. Comprehensive details of each solver can be further explored in Table 3.5. This table presents, for every instance and solver, the initial and final bounds, along with the time required to obtain these bounds. The **Part – Gather $_{SO}^{Frba}$** configuration seems to falter with specific instances, notably CDG T1,2,3 during the period of August 21-27 2023. However, generally, the **Gather** configurations always allow you to obtain a bound, distinguishing themselves from configurations based on **Card**, **Bin**, or **ADEWeak**.

It is noteworthy to highlight the efficacy of the **Full – Gather $_{SO}^{Wdeg}$** configuration. It excels by rapidly procuring the best initial solutions for instances 3, 4, and 6 (as sequenced in the table) when utilizing the full decomposition. Additionally, it achieves the best final bound for instance

2. Yet, its efficacy seems somewhat diminished for the Orly airport scenarios, as evident from the outcomes for instances 7, 8, and 9.

Term	Period	Solver	First B (Time (s))	Last B (Time (s))
CDG T1	0703-0709	Part – Gather $_{SO}^{Frba}$	23,465,000 (6.43s)	23,550,000 (7.38s)
CDG T1	0703-0709	Part – ADEWeak $_{SO}^{Wdeg}$	23,450,000 (6.58s)	23,550,000 (7.87s)
CDG T1	0703-0709	None – ADEWeak – $i2e_{SO}^{Frba}$	23,120,000 (8.37s)	23,550,000 (80.14s)
CDG T1	0703-0709	None – Card – $i2e_{SO}^{Frba}$	23,120,000 (11.97s)	23,550,000 (129.4s)
CDG T1	0703-0709	None – Bin – $i2e_{SO}^{Frba}$	23,120,000 (17.26s)	23,550,000 (57.1s)
CDG T1	0703-0709	Part – Bin – $i2e_{SO}^{Frba}$	22,830,000 (11.61s)	23,510,000 (254.37s)
CDG T1	0703-0709	Part – Card – $i2e_{SO}^{Frba}$	19,625,000 (9.66s)	23,550,000 (32.15s)
CDG T1	0703-0709	None – Gather – $i2e_{First}^{Frba}$	15,895,000 (6.76s)	23,550,000 (195.27s)
CDG T1,2,3	0821-0827	Full – Bin – $i2e_{SO}^{Wdeg}$	20,779,400 (10.26s)	20,941,500 (76.8s)
CDG T1,2,3	0821-0827	Full – ADEWeak – $i2e_{SO}^{Wdeg}$	20,501,000 (6.22s)	20,941,500 (57.77s)
CDG T1,2,3	0821-0827	Full – Gather $_{SO}^{Wdeg}$	20,478,300 (6.21s)	20,941,500 (51.25s)
CDG T1,2,3	0821-0827	Full – Card – $i2e_{SO}^{Wdeg}$	TO	TO
CDG T1,2,3	0821-0827	Part – Bin – $i2e_{SO}^{Frba}$	18,273,900 (24.17s)	19,143,600 (299.17s)
CDG T1,2,3	0821-0827	Part – ADEWeak $_{SO}^{Wdeg}$	17,802,500 (9.19s)	19,022,100 (282.91s)
CDG T1,2,3	0821-0827	Part – Gather $_{SO}^{Frba}$	17,202,400 (4.85s)	19,272,700 (82.22s)
CDG T1,2,3	0821-0827	None – ADEWeak – $i2e_{SO}^{Frba}$	17,138,900 (30.98s)	17,152,500 (280.27s)
CDG T1,2,3	0821-0827	None – Bin – $i2e_{SO}^{Frba}$	17,138,900 (53.4s)	17,732,800 (297.86s)
CDG T1,2,3	0821-0827	None – Gather – $i2e_{First}^{Frba}$	10,926,000 (15.22s)	13,183,900 (297.0s)
CDG T1,2,3	0821-0827	Part – Card – $i2e_{First}^{Frba}$	TO	TO
CDG T1,2,3	0821-0827	None – Card – $i2e_{SO}^{Frba}$	TO	TO
CDG T1,2,3	0911-0917	Full – Gather $_{SO}^{Wdeg}$	61,802,700 (12.38s)	61,908,700 (231.76s)
CDG T1,2,3	0911-0917	Full – ADEWeak – $i2e_{SO}^{Wdeg}$	61,802,700 (13.93s)	61,944,700 (243.01s)
CDG T1,2,3	0911-0917	Full – Bin – $i2e_{SO}^{Wdeg}$	61,802,700 (111.37s)	61,802,700 (111.37s)
CDG T1,2,3	0911-0917	Part – ADEWeak $_{SO}^{Wdeg}$	60,018,100 (18.49s)	60,483,600 (241.98s)
CDG T1,2,3	0911-0917	Part – Gather $_{SO}^{Frba}$	55,270,000 (15.33s)	57,689,400 (288.85s)
CDG T1,2,3	0911-0917	None – Gather – $i2e_{First}^{Frba}$	48,654,700 (27.32s)	48,654,700 (27.32s)
CDG T1,2,3	0911-0917	Full – Card – $i2e_{SO}^{Wdeg}$	TO	TO
CDG T1,2,3	0911-0917	Part – Card – $i2e_{First}^{Frba}$	TO	TO
CDG T1,2,3	0911-0917	Part – Bin – $i2e_{SO}^{Frba}$	TO	TO
CDG T1,2,3	0911-0917	None – Card – $i2e_{SO}^{Frba}$	TO	TO
CDG T1,2,3	0911-0917	None – Bin – $i2e_{SO}^{Frba}$	TO	TO
CDG T1,2,3	0911-0917	None – ADEWeak – $i2e_{SO}^{Frba}$	TO	TO
CDG T1,2,3	0918-0924	Full – Gather $_{SO}^{Wdeg}$	61,677,400 (11.65s)	61,727,400 (246.09s)
CDG T1,2,3	0918-0924	Full – Bin – $i2e_{SO}^{Wdeg}$	61,677,400 (121.21s)	61,677,400 (121.21s)
CDG T1,2,3	0918-0924	Full – ADEWeak – $i2e_{SO}^{Wdeg}$	61,671,400 (11.14s)	61,733,400 (264.55s)
CDG T1,2,3	0918-0924	Part – ADEWeak $_{SO}^{Wdeg}$	59,863,500 (21.15s)	60,332,700 (289.24s)
CDG T1,2,3	0918-0924	Part – Gather $_{SO}^{Frba}$	55,440,000 (13.53s)	57,761,900 (283.9s)
CDG T1,2,3	0918-0924	None – Gather – $i2e_{First}^{Frba}$	48,710,100 (35.64s)	48,710,100 (35.64s)
CDG T1,2,3	0918-0924	Full – Card – $i2e_{SO}^{Wdeg}$	TO	TO
CDG T1,2,3	0918-0924	Part – Card – $i2e_{First}^{Frba}$	TO	TO
CDG T1,2,3	0918-0924	Part – Bin – $i2e_{SO}^{Frba}$	TO	TO
CDG T1,2,3	0918-0924	None – Card – $i2e_{SO}^{Frba}$	TO	TO
CDG T1,2,3	0918-0924	None – Bin – $i2e_{SO}^{Frba}$	TO	TO
CDG T1,2,3	0918-0924	None – ADEWeak – $i2e_{SO}^{Frba}$	TO	TO
CDG T2BD	0703-0709	Full – ADEWeak – $i2e_{SO}^{Wdeg}$	19,149,020 (6.2s)	19,149,020 (6.2s)
CDG T2BD	0703-0709	Full – Card – $i2e_{SO}^{Wdeg}$	19,149,020 (7.47s)	19,149,020 (7.47s)
CDG T2BD	0703-0709	Part – ADEWeak $_{SO}^{Wdeg}$	19,149,020 (7.61s)	19,149,020 (7.61s)
CDG T2BD	0703-0709	Full – Gather $_{SO}^{Wdeg}$	19,129,020 (5.85s)	19,149,020 (6.52s)
CDG T2BD	0703-0709	Full – Bin – $i2e_{SO}^{Wdeg}$	19,129,020 (8.59s)	19,149,020 (9.59s)
CDG T2BD	0703-0709	Part – Gather $_{SO}^{Frba}$	19,059,020 (6.83s)	19,149,020 (7.78s)
CDG T2BD	0703-0709	None – Card – $i2e_{SO}^{Frba}$	18,959,020 (13.89s)	19,219,020 (179.22s)
CDG T2BD	0703-0709	None – ADEWeak – $i2e_{SO}^{Frba}$	18,959,020 (17.63s)	19,219,020 (114.99s)
CDG T2BD	0703-0709	None – Bin – $i2e_{SO}^{Frba}$	18,959,020 (25.7s)	19,219,020 (89.55s)
CDG T2BD	0703-0709	Part – Bin – $i2e_{SO}^{Frba}$	18,719,020 (11.84s)	19,149,020 (107.02s)
CDG T2BD	0703-0709	Part – Card – $i2e_{First}^{Frba}$	13,419,629 (9.12s)	19,149,020 (84.11s)

Continued on next page

Term	Period	Solver	First B (Time (s))	Last B (Time (s))
CDG T2BD	0703-0709	None – Gather – $i2c_{\text{First}}^{\text{Frba}}$	12,619,755 (12.09s)	19,219,020 (286.29s)
ORY	0508-0514	Full – Gather – $i2c_{\text{SO}}^{\text{Wdeg}}$	40,693,200 (7.74s)	41,993,600 (299.79s)
ORY	0508-0514	Full – ADEWeak – $i2c_{\text{SO}}^{\text{Wdeg}}$	40,450,200 (6.93s)	41,965,200 (248.13s)
ORY	0508-0514	Full – Bin – $i2c_{\text{SO}}^{\text{Wdeg}}$	40,373,500 (10.67s)	41,853,400 (288.15s)
ORY	0508-0514	Part – Gather – $i2c_{\text{SO}}^{\text{Frba}}$	40,192,400 (9.66s)	41,846,700 (253.73s)
ORY	0508-0514	Part – ADEWeak – $i2c_{\text{SO}}^{\text{Wdeg}}$	39,604,200 (11.18s)	41,536,000 (295.32s)
ORY	0508-0514	Full – Card – $i2c_{\text{SO}}^{\text{Wdeg}}$	39,580,600 (11.64s)	41,637,000 (294.19s)
ORY	0508-0514	Part – Bin – $i2c_{\text{Bivs}}^{\text{Frba}}$	39,557,200 (28.89s)	40,619,100 (297.22s)
ORY	0508-0514	None – ADEWeak – $i2c_{\text{SO}}^{\text{Frba}}$	39,518,900 (26.71s)	39,519,200 (297.1s)
ORY	0508-0514	None – Bin – $i2c_{\text{SO}}^{\text{Frba}}$	39,518,900 (48.88s)	39,581,600 (294.31s)
ORY	0508-0514	None – Card – $i2c_{\text{SO}}^{\text{Frba}}$	39,518,900 (100.36s)	39,518,900 (100.36s)
ORY	0508-0514	Part – Card – $i2c_{\text{First}}^{\text{Frba}}$	34,299,700 (26.86s)	39,306,600 (299.98s)
ORY	0508-0514	None – Gather – $i2c_{\text{First}}^{\text{Frba}}$	34,010,700 (14.88s)	34,736,700 (298.67s)
ORY	0619-0625	Part – Bin – $i2c_{\text{Bivs}}^{\text{Frba}}$	28,728,414 (22.57s)	29,186,279 (281.61s)
ORY	0619-0625	Part – Gather – $i2c_{\text{SO}}^{\text{Frba}}$	28,408,874 (4.49s)	29,596,588 (292.85s)
ORY	0619-0625	None – ADEWeak – $i2c_{\text{SO}}^{\text{Frba}}$	28,386,081 (16.45s)	28,450,581 (297.51s)
ORY	0619-0625	None – Bin – $i2c_{\text{SO}}^{\text{Frba}}$	28,386,081 (33.77s)	28,583,781 (294.07s)
ORY	0619-0625	None – Card – $i2c_{\text{SO}}^{\text{Frba}}$	28,386,081 (64.93s)	28,388,381 (296.73s)
ORY	0619-0625	Part – ADEWeak – $i2c_{\text{SO}}^{\text{Wdeg}}$	28,138,018 (8.22s)	29,458,052 (262.75s)
ORY	0619-0625	Full – Gather – $i2c_{\text{SO}}^{\text{Wdeg}}$	26,899,903 (7.91s)	27,705,238 (298.28s)
ORY	0619-0625	Full – Bin – $i2c_{\text{SO}}^{\text{Wdeg}}$	26,731,207 (13.59s)	27,698,141 (286.09s)
ORY	0619-0625	Full – ADEWeak – $i2c_{\text{SO}}^{\text{Wdeg}}$	26,564,490 (7.59s)	27,690,838 (237.92s)
ORY	0619-0625	Part – Card – $i2c_{\text{First}}^{\text{Frba}}$	25,564,544 (11.31s)	29,077,973 (289.32s)
ORY	0619-0625	None – Gather – $i2c_{\text{First}}^{\text{Frba}}$	25,548,051 (9.84s)	26,692,886 (299.95s)
ORY	0619-0625	Full – Card – $i2c_{\text{SO}}^{\text{Wdeg}}$	TO	TO
ORY	0626-0702	Part – Gather – $i2c_{\text{SO}}^{\text{Frba}}$	30,418,573 (8.64s)	31,144,690 (294.97s)
ORY	0626-0702	Part – Bin – $i2c_{\text{Bivs}}^{\text{Frba}}$	30,265,997 (23.49s)	30,776,981 (299.11s)
ORY	0626-0702	None – ADEWeak – $i2c_{\text{SO}}^{\text{Frba}}$	29,889,358 (18.79s)	29,914,458 (277.4s)
ORY	0626-0702	None – Card – $i2c_{\text{SO}}^{\text{Frba}}$	29,889,358 (44.32s)	29,897,558 (131.52s)
ORY	0626-0702	None – Bin – $i2c_{\text{SO}}^{\text{Frba}}$	29,889,358 (59.0s)	30,035,958 (293.54s)
ORY	0626-0702	Part – ADEWeak – $i2c_{\text{SO}}^{\text{Wdeg}}$	29,607,819 (8.74s)	30,993,466 (293.89s)
ORY	0626-0702	Full – Gather – $i2c_{\text{SO}}^{\text{Wdeg}}$	28,011,791 (7.11s)	29,155,236 (267.2s)
ORY	0626-0702	Full – ADEWeak – $i2c_{\text{SO}}^{\text{Wdeg}}$	27,655,885 (6.39s)	29,161,343 (298.09s)
ORY	0626-0702	Full – Bin – $i2c_{\text{SO}}^{\text{Wdeg}}$	27,629,682 (13.89s)	29,022,830 (273.92s)
ORY	0626-0702	Part – Card – $i2c_{\text{First}}^{\text{Frba}}$	27,140,765 (16.68s)	30,437,262 (299.94s)
ORY	0626-0702	None – Gather – $i2c_{\text{First}}^{\text{Frba}}$	26,965,458 (20.31s)	27,787,658 (299.49s)
ORY	0626-0702	Full – Card – $i2c_{\text{SO}}^{\text{Wdeg}}$	TO	TO
ORY	0703-0709	Part – Gather – $i2c_{\text{SO}}^{\text{Frba}}$	30,338,194 (7.78s)	31,355,600 (237.55s)
ORY	0703-0709	Part – Bin – $i2c_{\text{Bivs}}^{\text{Frba}}$	30,287,419 (21.35s)	30,964,399 (297.16s)
ORY	0703-0709	None – ADEWeak – $i2c_{\text{SO}}^{\text{Frba}}$	30,115,897 (29.81s)	30,161,797 (276.36s)
ORY	0703-0709	None – Card – $i2c_{\text{SO}}^{\text{Frba}}$	30,115,897 (44.44s)	30,148,797 (290.04s)
ORY	0703-0709	None – Bin – $i2c_{\text{SO}}^{\text{Frba}}$	30,115,897 (55.2s)	30,302,397 (297.37s)
ORY	0703-0709	Part – ADEWeak – $i2c_{\text{SO}}^{\text{Wdeg}}$	29,681,127 (8.55s)	31,225,383 (242.41s)
ORY	0703-0709	Full – Gather – $i2c_{\text{SO}}^{\text{Wdeg}}$	28,068,100 (7.11s)	29,235,350 (257.21s)
ORY	0703-0709	Full – Bin – $i2c_{\text{SO}}^{\text{Wdeg}}$	27,946,300 (14.28s)	29,118,736 (284.97s)
ORY	0703-0709	Full – ADEWeak – $i2c_{\text{SO}}^{\text{Wdeg}}$	27,725,879 (7.02s)	29,243,660 (287.32s)
ORY	0703-0709	Part – Card – $i2c_{\text{First}}^{\text{Frba}}$	27,609,302 (18.6s)	30,578,464 (298.47s)
ORY	0703-0709	None – Gather – $i2c_{\text{First}}^{\text{Frba}}$	27,492,302 (11.32s)	28,392,602 (299.41s)
ORY	0703-0709	Full – Card – $i2c_{\text{SO}}^{\text{Wdeg}}$	TO	TO

Table 3.5: Results for no-overlapping family.

3.4.6 Overlapping family

This section concerns experiments on the family of instances **overlapping**. We define a new set of instances for this family noted $\mathcal{I}_3$, a subset of $\mathcal{I}_1$. This family contains all the instances from Table 3.3 except the instance that concerns the terminal T2B and D because this scenario does not contain overlapping rules.

$$\{\mathcal{I}_{\text{overlapping}}^{\text{checkin}}\} \quad (\mathcal{I}_3)$$

Summary of the experiments 3 (overlapping - , ,)

$\mathcal{I}_3$

Ψ_2

5 minutes

32 GB

Linux CentOS Stream 8.3

Two quadcore Intel XEON E5-2637 (3.5 GHz)

128 GB

Solver	Score
Full – ADEWeak – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	3.5
Full – Bin – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	3.5
Full – Gather – $i2e_{\text{Bivs}+\text{SO}}^{\text{Wdeg}}$	3.5
Full – Card – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	3.0
Part – Gather – $i2e_{\text{Bivs}}^{\text{Wdeg}}$	2.5
Part – ADEWeak – $i2e_{\text{SO}}^{\text{Wdeg}}$	1.5
Part – Bin – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	1.0
Part – Card – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	1.0
None – ADEWeak – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	0.5
None – Bin – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	0.5
None – Card – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	0.5
None – Gather – $i2e_{\text{Bivs}+\text{SO}}^{\text{Frba}}$	0.5

Table 3.6: Ranking of the solvers for **overlapping** family.

Like the previous section, Table 3.7 presents for each instance and each solver the first bound (resp. last) and the time for obtaining this bound. **Gather** based configurations are in more difficulty on this family but still contribute the 4 best first bounds and always get a bound, unlike the **Card** approach, which is in timeout with full decomposition for the last two instances.

Term	Period	Solver	First B (Time (s))	Last B (Time (s))
CDG T1	0703-0709	Part – Gather – $i2e_{Bivs}^{wdeg}$	27,675,000 (5.65s)	27,675,000 (5.65s)
CDG T1	0703-0709	Part – Card – $i2e_{Bivs+SO}^{Frba}$	27,675,000 (10.28s)	27,675,000 (10.28s)
CDG T1	0703-0709	Part – Bin – $i2e_{Bivs+SO}^{Frba}$	27,675,000 (11.23s)	27,675,000 (11.23s)
CDG T1	0703-0709	Part – ADEWeak – $i2e_{SO}^{wdeg}$	27,660,000 (7.54s)	27,675,000 (8.18s)
CDG T1	0703-0709	None – Gather – $i2e_{Bivs+SO}^{Frba}$	27,655,000 (7.75s)	27,675,000 (41.73s)
CDG T1	0703-0709	None – Card – $i2e_{Bivs+SO}^{Frba}$	27,655,000 (18.07s)	27,675,000 (167.12s)
CDG T1	0703-0709	None – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	27,655,000 (18.23s)	27,675,000 (83.32s)
CDG T1	0703-0709	None – Bin – $i2e_{Bivs+SO}^{Frba}$	27,655,000 (18.44s)	27,675,000 (150.58s)
CDG T1,T2,T3	0821-0827	Full – Bin – $i2e_{Bivs+SO}^{Frba}$	29,397,000 (5.4s)	29,397,000 (5.4s)
CDG T1,T2,T3	0821-0827	Full – Gather – $i2e_{Bivs+SO}^{wdeg}$	29,397,000 (6.33s)	29,397,000 (6.33s)
CDG T1,T2,T3	0821-0827	Full – Card – $i2e_{Bivs+SO}^{Frba}$	29,397,000 (6.4s)	29,397,000 (6.4s)
CDG T1,T2,T3	0821-0827	Full – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	29,397,000 (6.48s)	29,397,000 (6.48s)
CDG T1,T2,T3	0821-0827	None – Bin – $i2e_{Bivs+SO}^{Frba}$	28,028,300 (16.73s)	28,028,300 (16.73s)
CDG T1,T2,T3	0821-0827	None – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	28,028,300 (17.04s)	28,028,300 (17.04s)
CDG T1,T2,T3	0821-0827	None – Card – $i2e_{Bivs+SO}^{Frba}$	28,028,300 (17.44s)	28,028,300 (17.44s)
CDG T1,T2,T3	0821-0827	None – Gather – $i2e_{Bivs+SO}^{Frba}$	28,028,300 (17.99s)	28,028,300 (17.99s)
CDG T1,T2,T3	0821-0827	Part – Gather – $i2e_{Bivs}^{wdeg}$	28,018,300 (7.15s)	28,018,300 (7.15s)
CDG T1,T2,T3	0821-0827	Part – Card – $i2e_{Bivs+SO}^{Frba}$	28,018,300 (8.21s)	28,018,300 (8.21s)
CDG T1,T2,T3	0821-0827	Part – Bin – $i2e_{Bivs+SO}^{Frba}$	28,018,300 (8.95s)	28,018,300 (8.95s)
CDG T1,T2,T3	0821-0827	Part – ADEWeak – $i2e_{SO}^{wdeg}$	25,324,800 (4.56s)	28,018,300 (8.17s)
CDG T1,T2,T3	0911-0917	Full – Card – $i2e_{Bivs+SO}^{Frba}$	94,281,600 (11.11s)	94,281,600 (11.11s)
CDG T1,T2,T3	0911-0917	Full – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	94,281,600 (11.44s)	94,281,600 (11.44s)
CDG T1,T2,T3	0911-0917	Full – Bin – $i2e_{Bivs+SO}^{Frba}$	94,281,600 (12.3s)	94,281,600 (12.3s)
CDG T1,T2,T3	0911-0917	Full – Gather – $i2e_{Bivs+SO}^{wdeg}$	94,281,600 (12.57s)	94,281,600 (12.57s)
CDG T1,T2,T3	0911-0917	Part – Bin – $i2e_{Bivs+SO}^{Frba}$	92,925,300 (13.2s)	92,925,300 (13.2s)
CDG T1,T2,T3	0911-0917	Part – Card – $i2e_{Bivs+SO}^{Frba}$	92,925,300 (15.47s)	92,925,300 (15.47s)
CDG T1,T2,T3	0911-0917	None – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	92,925,300 (122.5s)	92,925,300 (122.5s)
CDG T1,T2,T3	0911-0917	None – Card – $i2e_{Bivs+SO}^{Frba}$	92,925,300 (152.94s)	92,925,300 (152.94s)
CDG T1,T2,T3	0911-0917	None – Bin – $i2e_{Bivs+SO}^{Frba}$	92,925,300 (154.13s)	92,925,300 (154.13s)
CDG T1,T2,T3	0911-0917	None – Gather – $i2e_{Bivs+SO}^{Frba}$	92,925,300 (160.36s)	92,925,300 (160.36s)
CDG T1,T2,T3	0911-0917	Part – ADEWeak – $i2e_{SO}^{wdeg}$	92,826,300 (9.22s)	92,925,300 (15.85s)
CDG T1,T2,T3	0918-0924	Full – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	93,895,800 (7.02s)	93,895,800 (7.02s)
CDG T1,T2,T3	0918-0924	Full – Gather – $i2e_{Bivs+SO}^{wdeg}$	93,895,800 (10.57s)	93,895,800 (10.57s)
CDG T1,T2,T3	0918-0924	Full – Card – $i2e_{Bivs+SO}^{Frba}$	93,895,800 (11.47s)	93,895,800 (11.47s)
CDG T1,T2,T3	0918-0924	Full – Bin – $i2e_{Bivs+SO}^{Frba}$	93,895,800 (12.78s)	93,895,800 (12.78s)
CDG T1,T2,T3	0918-0924	Part – Gather – $i2e_{Bivs}^{wdeg}$	92,575,000 (13.03s)	92,575,000 (13.03s)
CDG T1,T2,T3	0918-0924	Part – Card – $i2e_{Bivs+SO}^{Frba}$	92,575,000 (13.85s)	92,575,000 (13.85s)
CDG T1,T2,T3	0918-0924	Part – Bin – $i2e_{Bivs+SO}^{Frba}$	92,575,000 (14.27s)	92,575,000 (14.27s)
CDG T1,T2,T3	0918-0924	None – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	92,575,000 (146.55s)	92,575,000 (146.55s)
CDG T1,T2,T3	0918-0924	None – Card – $i2e_{Bivs+SO}^{Frba}$	92,575,000 (153.01s)	92,575,000 (153.01s)
CDG T1,T2,T3	0918-0924	None – Gather – $i2e_{Bivs+SO}^{Frba}$	92,575,000 (153.67s)	92,575,000 (153.67s)
CDG T1,T2,T3	0918-0924	None – Bin – $i2e_{Bivs+SO}^{Frba}$	92,575,000 (160.5s)	92,575,000 (160.5s)
CDG T1,T2,T3	0918-0924	Part – ADEWeak – $i2e_{SO}^{wdeg}$	92,404,000 (8.9s)	92,575,000 (21.76s)
ORY	0508-0514	Full – Gather – $i2e_{Bivs+SO}^{wdeg}$	45,080,700 (7.57s)	45,760,300 (294.2s)
ORY	0508-0514	Part – Gather – $i2e_{Bivs}^{wdeg}$	44,782,000 (9.3s)	45,522,500 (296.62s)
ORY	0508-0514	Full – Bin – $i2e_{Bivs+SO}^{Frba}$	44,241,300 (13.95s)	45,535,000 (298.93s)
ORY	0508-0514	Full – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	44,237,800 (8.59s)	45,650,000 (291.81s)
ORY	0508-0514	Full – Card – $i2e_{Bivs+SO}^{Frba}$	44,173,300 (21.38s)	45,185,300 (299.41s)
ORY	0508-0514	Part – Bin – $i2e_{Bivs+SO}^{Frba}$	43,960,500 (25.97s)	44,955,800 (295.39s)
ORY	0508-0514	Part – Card – $i2e_{Bivs+SO}^{Frba}$	43,927,800 (67.37s)	44,220,200 (291.41s)
ORY	0508-0514	None – Gather – $i2e_{Bivs+SO}^{Frba}$	43,924,300 (16.23s)	43,987,800 (279.05s)
ORY	0508-0514	None – Bin – $i2e_{Bivs+SO}^{Frba}$	43,924,300 (78.59s)	43,958,300 (292.36s)
ORY	0508-0514	None – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	43,924,300 (155.74s)	43,927,800 (286.65s)
ORY	0508-0514	Part – ADEWeak – $i2e_{SO}^{wdeg}$	43,668,100 (9.56s)	45,739,700 (282.01s)
ORY	0508-0514	None – Card – $i2e_{Bivs+SO}^{Frba}$	TO	TO
ORY	0619-0625	Part – Gather – $i2e_{Bivs}^{wdeg}$	32,359,753 (8.89s)	32,697,569 (298.68s)
ORY	0619-0625	Part – Bin – $i2e_{Bivs+SO}^{Frba}$	32,246,332 (15.79s)	32,477,257 (298.36s)

Continued on next page

Term	Period	Solver	First B (Time (s))	Last B (Time (s))
ORY	0619-0625	Part – Card – $i2e_{Bivs+SO}^{Frba}$	32,195,732 (29.66s)	32,355,857 (296.22s)
ORY	0619-0625	None – Gather – $i2e_{Bivs+SO}^{Frba}$	32,178,032 (12.91s)	32,288,732 (298.29s)
ORY	0619-0625	None – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	32,178,032 (60.12s)	32,199,532 (273.07s)
ORY	0619-0625	None – Bin – $i2e_{Bivs+SO}^{Frba}$	32,178,032 (63.89s)	32,251,532 (294.19s)
ORY	0619-0625	None – Card – $i2e_{Bivs+SO}^{Frba}$	32,178,032 (222.87s)	32,178,132 (249.84s)
ORY	0619-0625	Part – ADEWeak – $i2e_{SO}^{wdeg}$	31,808,957 (8.61s)	32,833,969 (266.73s)
ORY	0619-0625	Full – Gather – $i2e_{Bivs+SO}^{wdeg}$	29,094,850 (7.15s)	29,501,666 (248.71s)
ORY	0619-0625	Full – Bin – $i2e_{Bivs+SO}^{Frba}$	29,042,154 (16.07s)	29,334,254 (293.52s)
ORY	0619-0625	Full – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	29,030,954 (8.38s)	29,366,554 (298.82s)
ORY	0619-0625	Full – Card – $i2e_{Bivs+SO}^{Frba}$	TO	TO
ORY	0626-0702	Part – Bin – $i2e_{Bivs+SO}^{Frba}$	34,034,441 (19.93s)	34,264,416 (288.25s)
ORY	0626-0702	Part – Card – $i2e_{Bivs+SO}^{Frba}$	33,926,041 (35.76s)	34,194,491 (287.51s)
ORY	0626-0702	None – Gather – $i2e_{Bivs+SO}^{Frba}$	33,923,641 (12.82s)	34,010,741 (271.01s)
ORY	0626-0702	None – Bin – $i2e_{Bivs+SO}^{Frba}$	33,923,641 (56.88s)	33,997,641 (291.91s)
ORY	0626-0702	None – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	33,923,641 (73.2s)	33,940,941 (267.72s)
ORY	0626-0702	None – Card – $i2e_{Bivs+SO}^{Frba}$	33,923,641 (224.23s)	33,923,741 (283.24s)
ORY	0626-0702	Part – Gather – $i2e_{Bivs}^{wdeg}$	33,882,869 (8.34s)	34,542,394 (292.68s)
ORY	0626-0702	Part – ADEWeak – $i2e_{SO}^{wdeg}$	33,345,559 (4.89s)	34,674,500 (294.63s)
ORY	0626-0702	Full – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	30,768,488 (9.16s)	31,192,488 (278.73s)
ORY	0626-0702	Full – Bin – $i2e_{Bivs+SO}^{Frba}$	30,695,888 (12.76s)	31,143,288 (298.64s)
ORY	0626-0702	Full – Gather – $i2e_{Bivs+SO}^{wdeg}$	30,597,871 (7.87s)	31,242,091 (298.68s)
ORY	0626-0702	Full – Card – $i2e_{Bivs+SO}^{Frba}$	TO	TO
ORY	0703-0709	Part – Gather – $i2e_{Bivs}^{wdeg}$	34,265,778 (8.4s)	34,726,489 (179.51s)
ORY	0703-0709	Part – Bin – $i2e_{Bivs+SO}^{Frba}$	34,032,158 (21.0s)	34,475,983 (292.12s)
ORY	0703-0709	None – Gather – $i2e_{Bivs+SO}^{Frba}$	34,024,258 (13.67s)	34,168,558 (293.58s)
ORY	0703-0709	None – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	34,024,258 (64.59s)	34,035,658 (287.31s)
ORY	0703-0709	None – Bin – $i2e_{Bivs+SO}^{Frba}$	34,024,258 (75.03s)	34,137,658 (297.52s)
ORY	0703-0709	None – Card – $i2e_{Bivs+SO}^{Frba}$	34,024,258 (236.94s)	34,024,258 (236.94s)
ORY	0703-0709	Part – Card – $i2e_{Bivs+SO}^{Frba}$	33,954,858 (41.11s)	34,296,783 (289.48s)
ORY	0703-0709	Part – ADEWeak – $i2e_{SO}^{wdeg}$	33,669,264 (8.87s)	34,836,995 (294.46s)
ORY	0703-0709	Full – Gather – $i2e_{Bivs+SO}^{wdeg}$	30,766,270 (7.92s)	31,338,086 (298.57s)
ORY	0703-0709	Full – Bin – $i2e_{Bivs+SO}^{Frba}$	30,730,880 (17.55s)	31,154,280 (298.43s)
ORY	0703-0709	Full – ADEWeak – $i2e_{Bivs+SO}^{Frba}$	30,721,480 (9.07s)	31,204,280 (291.87s)
ORY	0703-0709	Full – Card – $i2e_{Bivs+SO}^{Frba}$	TO	TO

Table 3.7: Results for overlapping family.

3.4.7 Overlapping with a limited number of tasks family

This section concerns experiments on the family of instances `overlapping – nbmax`. We define a new set of instances for this family noted $\mathcal{I}_4$, a subset of $\mathcal{I}_1$. Like the previous section, this family contains only instances from Table 3.3 contains overlapping rules with a limited number of tasks. Each instance of this family has been modeled with 3 variants. The variant `Sum` corresponds to the modeling from model 2. The second variant `minod` corresponds to modeling 6, and finally the last variant `minod – count` corresponds to modeling 7.

$$\left\{ \mathcal{I}_{\text{checkin}}^{\text{overlapping-nbmax}} \right\} \quad (\mathcal{I}_4)$$

Summary of the experiments 4 (overlapping – nbmax -  , )	
	$\mathcal{I}_4$
	Ψ_2
	5 minutes
	32 GB
	Linux CentOS Stream 8.3
	Two quadcore Intel XEON E5-2637 (3.5 GHz)
	128 GB

Table 3.8 presents scores on the same principle as the previous sections. Once again, for this family, we note that `Gather` configurations contribute to obtaining the best first bound. (2 times) and best last bound (3 times). The configuration `Part – BinBivs+SOFrba` contributes thrice to the best first bound but never obtain both the best first and best last bound. The `ADEWeak` and `Card` approaches can lead to timeouts. Concerning the modeling, we notice that the first solver is always (except once) a solver using one of the proposed refinements of Constraint C_6 : `minod` or `minod – count`.

Solver	Model	Score
Part – Gather $_{S0}^{Frba}$	sum	1.5
Part – Gather $_{S0}^{Frba}$	minod-count	1.5
Full – Gather $_{S0}^{Frba}$	minod-count	1.0
Part – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	minod	0.5
Part – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.5
Part – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	sum	0.5
Full – Gather $_{S0}^{Frba}$	sum	0.5
Full – Gather $_{S0}^{Frba}$	minod	0.5
Part – Bin $_{Bivs+S0}^{Frba}$	sum	0.5
Part – Gather $_{S0}^{Frba}$	minod	0.5
Part – Bin $_{Bivs+S0}^{Frba}$	minod-count	0.0
None – Card – $i2e_{Bivs+S0}^{Frba}$	sum	0.0
Part – Card – $i2e_{Bivs+S0}^{Frba}$	minod	0.0
Part – Bin $_{Bivs+S0}^{Frba}$	minod	0.0
Part – Card – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.0
Part – Card – $i2e_{Bivs+S0}^{Frba}$	sum	0.0
None – Gather – $i2e_{Bivs+S0}^{Frba}$	sum	0.0
None – Gather – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.0
None – Gather – $i2e_{Bivs+S0}^{Frba}$	minod	0.0
Full – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	minod	0.0
None – Card – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.0
Full – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.0
None – Bin – $i2e_{Bivs+S0}^{Frba}$	sum	0.0
None – Bin – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.0
None – Bin – $i2e_{Bivs+S0}^{Frba}$	minod	0.0
None – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	sum	0.0
None – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.0
None – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	minod	0.0
Full – Card – $i2e_{Bivs+S0}^{Frba}$	sum	0.0
Full – Card – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.0
Full – Card – $i2e_{Bivs+S0}^{Frba}$	minod	0.0
Full – Bin – $i2e_{Bivs+S0}^{Frba}$	sum	0.0
Full – Bin – $i2e_{Bivs+S0}^{Frba}$	minod-count	0.0
Full – Bin – $i2e_{Bivs+S0}^{Frba}$	minod	0.0
Full – ADEWeak – $i2e_{Bivs+S0}^{Frba}$	sum	0.0
None – Card – $i2e_{Bivs+S0}^{Frba}$	minod	0.0

Table 3.8: Ranking of the solvers for family **overlapping** – nbmax.

Term	Period	Solver	Model	First B (Time (s))	Last B (Time (s))
CDG T1	0703-0709	Part - Bin - $i2e_{Bivs+SO}^{Frba}$	minod-count	23,975,000 (14.13s)	24,255,000 (243.59s)
CDG T1	0703-0709	Part - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	sum	23,955,000 (10.56s)	24,305,000 (121.85s)
CDG T1	0703-0709	Part - Gather - $i2e_{Bivs+SO}^{Frba}$	minod	23,945,000 (9.41s)	24,275,000 (21.12s)
CDG T1	0703-0709	Part - Bin - $i2e_{Bivs+SO}^{Frba}$	minod	23,945,000 (13.06s)	24,255,000 (245.19s)
CDG T1	0703-0709	Part - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod	23,935,000 (9.42s)	24,305,000 (117.87s)
CDG T1	0703-0709	Part - Bin - $i2e_{Bivs+SO}^{Frba}$	sum	23,905,000 (13.46s)	24,255,000 (267.92s)
CDG T1	0703-0709	Part - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod-count	23,880,000 (9.43s)	24,305,000 (102.05s)
CDG T1	0703-0709	Part - Card - $i2e_{Bivs+SO}^{Frba}$	minod	23,865,000 (11.22s)	24,195,000 (280.39s)
CDG T1	0703-0709	Part - Card - $i2e_{Bivs+SO}^{Frba}$	sum	23,850,000 (12.17s)	24,145,000 (243.84s)
CDG T1	0703-0709	Part - Card - $i2e_{Bivs+SO}^{Frba}$	minod-count	23,835,000 (11.89s)	24,195,000 (277.17s)
CDG T1	0703-0709	None - Gather - $i2e_{Bivs+SO}^{Frba}$	minod	23,760,000 (13.06s)	24,005,000 (295.14s)
CDG T1	0703-0709	None - Gather - $i2e_{Bivs+SO}^{Frba}$	sum	23,760,000 (14.62s)	23,995,000 (261.53s)
CDG T1	0703-0709	None - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod-count	23,760,000 (17.27s)	23,955,000 (82.81s)
CDG T1	0703-0709	None - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod	23,760,000 (19.44s)	23,955,000 (84.84s)
CDG T1	0703-0709	None - Gather - $i2e_{Bivs+SO}^{Frba}$	minod-count	23,760,000 (19.93s)	23,995,000 (249.34s)
CDG T1	0703-0709	None - Bin - $i2e_{Bivs+SO}^{Frba}$	minod-count	23,760,000 (23.12s)	23,955,000 (83.01s)
CDG T1	0703-0709	None - Bin - $i2e_{Bivs+SO}^{Frba}$	minod	23,760,000 (24.65s)	23,955,000 (84.76s)
CDG T1	0703-0709	None - Card - $i2e_{Bivs+SO}^{Frba}$	minod	23,760,000 (25.72s)	23,955,000 (182.94s)
CDG T1	0703-0709	None - Bin - $i2e_{Bivs+SO}^{Frba}$	sum	23,760,000 (26.06s)	23,955,000 (93.18s)
CDG T1	0703-0709	None - Card - $i2e_{Bivs+SO}^{Frba}$	sum	23,760,000 (28.85s)	23,955,000 (208.16s)
CDG T1	0703-0709	None - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	sum	23,760,000 (30.35s)	23,955,000 (127.44s)
CDG T1	0703-0709	None - Card - $i2e_{Bivs+SO}^{Frba}$	minod-count	23,760,000 (37.92s)	23,955,000 (199.87s)
CDG T1	0703-0709	Part - Gather - $i2e_{Bivs+SO}^{Frba}$	minod-count	23,755,000 (8.98s)	24,275,000 (19.38s)
CDG T1	0703-0709	Part - Gather - $i2e_{Bivs+SO}^{Frba}$	sum	23,610,000 (8.4s)	24,275,000 (17.26s)
ORY	0508-0514	Full - Gather - $i2e_{Bivs+SO}^{Frba}$	minod-count	43,232,500 (32.41s)	43,788,200 (293.06s)
ORY	0508-0514	Full - Gather - $i2e_{Bivs+SO}^{Frba}$	minod	43,219,800 (34.64s)	43,786,000 (286.24s)
ORY	0508-0514	Full - Gather - $i2e_{Bivs+SO}^{Frba}$	sum	43,082,500 (16.82s)	43,786,400 (298.86s)
ORY	0508-0514	Full - Bin - $i2e_{Bivs+SO}^{Frba}$	minod	42,992,600 (44.35s)	43,356,500 (298.9s)
ORY	0508-0514	Full - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod	42,936,500 (39.58s)	43,456,300 (287.57s)
ORY	0508-0514	Full - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod-count	42,930,900 (46.37s)	43,461,700 (299.98s)
ORY	0508-0514	Full - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	sum	42,870,900 (30.59s)	43,433,100 (299.55s)
ORY	0508-0514	Full - Bin - $i2e_{Bivs+SO}^{Frba}$	minod-count	42,864,200 (44.05s)	43,356,100 (271.06s)
ORY	0508-0514	Full - Bin - $i2e_{Bivs+SO}^{Frba}$	sum	42,813,300 (37.45s)	43,356,000 (293.43s)
ORY	0508-0514	Full - Card - $i2e_{Bivs+SO}^{Frba}$	minod-count	42,163,400 (49.43s)	43,155,900 (291.41s)
ORY	0508-0514	Full - Card - $i2e_{Bivs+SO}^{Frba}$	minod	42,111,700 (52.09s)	43,125,900 (269.58s)
ORY	0508-0514	Part - Gather - $i2e_{Bivs+SO}^{Frba}$	minod-count	42,071,100 (47.79s)	43,593,900 (299.92s)
ORY	0508-0514	Full - Card - $i2e_{Bivs+SO}^{Frba}$	sum	42,017,600 (40.84s)	43,083,300 (276.04s)
ORY	0508-0514	Part - Bin - $i2e_{Bivs+SO}^{Frba}$	minod-count	41,844,300 (62.31s)	42,739,900 (282.22s)
ORY	0508-0514	Part - Bin - $i2e_{Bivs+SO}^{Frba}$	minod	41,822,700 (61.4s)	42,779,600 (297.56s)
ORY	0508-0514	Part - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod	41,687,600 (67.69s)	42,820,400 (293.8s)
ORY	0508-0514	Part - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod-count	41,685,900 (57.31s)	42,834,900 (290.86s)
ORY	0508-0514	Part - Bin - $i2e_{Bivs+SO}^{Frba}$	sum	41,671,600 (40.02s)	42,779,900 (298.23s)
ORY	0508-0514	Part - Card - $i2e_{Bivs+SO}^{Frba}$	minod	41,655,300 (123.04s)	41,827,100 (299.98s)
ORY	0508-0514	Part - Card - $i2e_{Bivs+SO}^{Frba}$	sum	41,648,200 (111.55s)	41,778,000 (297.95s)
ORY	0508-0514	Part - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	sum	41,648,100 (43.01s)	42,598,500 (298.28s)
ORY	0508-0514	Part - Card - $i2e_{Bivs+SO}^{Frba}$	minod-count	41,646,900 (108.49s)	41,806,900 (289.74s)
ORY	0508-0514	Part - Gather - $i2e_{Bivs+SO}^{Frba}$	minod	41,632,600 (48.03s)	43,587,800 (298.1s)
ORY	0508-0514	None - Gather - $i2e_{Bivs+SO}^{Frba}$	minod-count	41,626,300 (137.47s)	41,627,000 (285.84s)
ORY	0508-0514	None - Gather - $i2e_{Bivs+SO}^{Frba}$	minod	41,626,300 (182.33s)	41,626,800 (268.04s)
ORY	0508-0514	None - Bin - $i2e_{Bivs+SO}^{Frba}$	minod	41,626,300 (242.98s)	TO
ORY	0508-0514	None - Bin - $i2e_{Bivs+SO}^{Frba}$	minod-count	41,626,300 (246.32s)	41,626,600 (292.84s)
ORY	0508-0514	Part - Gather - $i2e_{Bivs+SO}^{Frba}$	sum	41,286,700 (16.09s)	43,597,800 (294.74s)
ORY	0508-0514	None - Card - $i2e_{Bivs+SO}^{Frba}$	minod	TO	TO
ORY	0508-0514	None - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod	TO	TO
ORY	0508-0514	None - Card - $i2e_{Bivs+SO}^{Frba}$	minod-count	TO	TO
ORY	0508-0514	None - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	minod-count	TO	TO
ORY	0508-0514	None - Card - $i2e_{Bivs+SO}^{Frba}$	sum	TO	TO
ORY	0508-0514	None - Bin - $i2e_{Bivs+SO}^{Frba}$	sum	TO	TO
ORY	0508-0514	None - ADEWeak - $i2e_{Bivs+SO}^{Frba}$	sum	TO	TO
ORY	0508-0514	None - Gather - $i2e_{Bivs+SO}^{Frba}$	sum	TO	TO

Continued on next page

Term	Period	Solver	Model	First B (Time (s))	Last B (Time (s))
ORY	0619-0625	Part – Bin $\frac{Frba}{Bivs+SO}$	minod	31,068,885 (53.73s)	31,186,296 (298.94s)
ORY	0619-0625	Part – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod	31,045,985 (48.75s)	31,296,298 (298.47s)
ORY	0619-0625	Part – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod-count	31,036,289 (50.65s)	31,243,794 (299.41s)
ORY	0619-0625	Part – Bin $\frac{Frba}{Bivs+SO}$	sum	30,944,419 (32.93s)	31,185,896 (298.56s)
ORY	0619-0625	Part – Bin $\frac{Frba}{Bivs+SO}$	minod-count	30,940,919 (40.91s)	31,186,396 (298.59s)
ORY	0619-0625	Part – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	sum	30,888,319 (33.14s)	31,217,294 (295.26s)
ORY	0619-0625	Part – Card – $i2e\frac{Frba}{Bivs+SO}$	minod	30,887,519 (74.51s)	31,099,981 (273.03s)
ORY	0619-0625	None – Gather – $i2e\frac{Frba}{Bivs+SO}$	minod	30,864,719 (65.74s)	30,917,319 (294.45s)
ORY	0619-0625	None – Gather – $i2e\frac{Frba}{Bivs+SO}$	minod-count	30,864,719 (67.48s)	30,915,319 (291.03s)
ORY	0619-0625	None – Gather – $i2e\frac{Frba}{Bivs+SO}$	sum	30,864,719 (86.57s)	30,877,419 (299.95s)
ORY	0619-0625	None – Bin – $i2e\frac{Frba}{Bivs+SO}$	minod	30,864,719 (103.67s)	30,898,219 (298.22s)
ORY	0619-0625	None – Bin – $i2e\frac{Frba}{Bivs+SO}$	sum	30,864,719 (126.81s)	30,867,919 (295.02s)
ORY	0619-0625	None – Bin – $i2e\frac{Frba}{Bivs+SO}$	minod-count	30,864,719 (140.81s)	30,883,419 (281.55s)
ORY	0619-0625	None – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod-count	30,864,719 (173.22s)	30,866,419 (273.74s)
ORY	0619-0625	None – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod	30,864,719 (210.59s)	30,866,119 (268.11s)
ORY	0619-0625	Part – Card – $i2e\frac{Frba}{Bivs+SO}$	minod-count	30,860,919 (52.33s)	31,100,481 (275.4s)
ORY	0619-0625	Part – Gather $\frac{Frba}{SO}$	minod	30,854,897 (37.29s)	31,610,865 (297.32s)
ORY	0619-0625	Part – Card – $i2e\frac{Frba}{Bivs+SO}$	sum	30,851,219 (48.39s)	31,099,981 (294.05s)
ORY	0619-0625	Part – Gather $\frac{Frba}{SO}$	minod-count	30,833,460 (30.33s)	31,624,665 (299.84s)
ORY	0619-0625	Part – Gather $\frac{Frba}{SO}$	sum	30,303,466 (15.11s)	31,607,265 (276.91s)
ORY	0619-0625	Full – Bin – $i2e\frac{Frba}{Bivs+SO}$	minod-count	28,583,212 (41.63s)	28,838,202 (299.82s)
ORY	0619-0625	Full – Bin – $i2e\frac{Frba}{Bivs+SO}$	minod	28,580,112 (40.2s)	28,830,802 (295.76s)
ORY	0619-0625	Full – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod-count	28,553,916 (33.23s)	28,860,794 (299.57s)
ORY	0619-0625	Full – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod	28,546,416 (37.81s)	28,856,894 (285.31s)
ORY	0619-0625	Full – Bin – $i2e\frac{Frba}{Bivs+SO}$	sum	28,528,736 (36.15s)	28,820,807 (295.9s)
ORY	0619-0625	Full – Gather $\frac{Frba}{SO}$	minod	28,489,131 (32.01s)	29,199,542 (295.13s)
ORY	0619-0625	Full – Gather $\frac{Frba}{SO}$	minod-count	28,434,728 (25.71s)	29,205,642 (285.68s)
ORY	0619-0625	Full – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	sum	28,419,436 (22.36s)	28,821,599 (296.25s)
ORY	0619-0625	Full – Gather $\frac{Frba}{SO}$	sum	28,063,114 (11.3s)	29,205,442 (296.94s)
ORY	0619-0625	Full – Card – $i2e\frac{Frba}{Bivs+SO}$	minod	TO	TO
ORY	0619-0625	Full – Card – $i2e\frac{Frba}{Bivs+SO}$	minod-count	TO	TO
ORY	0619-0625	Full – Card – $i2e\frac{Frba}{Bivs+SO}$	sum	TO	TO
ORY	0619-0625	None – Card – $i2e\frac{Frba}{Bivs+SO}$	minod	TO	TO
ORY	0619-0625	None – Card – $i2e\frac{Frba}{Bivs+SO}$	minod-count	TO	TO
ORY	0619-0625	None – Card – $i2e\frac{Frba}{Bivs+SO}$	sum	TO	TO
ORY	0619-0625	None – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	sum	TO	TO
ORY	0626-0702	Part – Gather $\frac{Frba}{SO}$	minod-count	32,836,794 (42.6s)	33,450,226 (291.84s)
ORY	0626-0702	Part – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod-count	32,763,085 (45.15s)	33,100,300 (298.42s)
ORY	0626-0702	Part – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod	32,748,894 (49.73s)	33,091,100 (297.52s)
ORY	0626-0702	Part – Bin $\frac{Frba}{Bivs+SO}$	minod	32,741,250 (51.61s)	33,048,700 (287.73s)
ORY	0626-0702	Part – Bin $\frac{Frba}{Bivs+SO}$	sum	32,702,165 (28.88s)	33,052,700 (289.14s)
ORY	0626-0702	Part – Bin $\frac{Frba}{Bivs+SO}$	minod-count	32,698,565 (37.49s)	33,052,700 (289.95s)
ORY	0626-0702	Part – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	sum	32,661,765 (32.41s)	33,065,900 (299.6s)
ORY	0626-0702	Part – Card – $i2e\frac{Frba}{Bivs+SO}$	minod-count	32,651,965 (61.9s)	32,860,591 (298.71s)
ORY	0626-0702	Part – Card – $i2e\frac{Frba}{Bivs+SO}$	sum	32,651,565 (58.25s)	32,833,091 (294.33s)
ORY	0626-0702	Part – Card – $i2e\frac{Frba}{Bivs+SO}$	minod	32,651,465 (69.36s)	32,846,091 (282.78s)
ORY	0626-0702	None – Gather – $i2e\frac{Frba}{Bivs+SO}$	minod	32,647,165 (71.69s)	32,685,865 (266.39s)
ORY	0626-0702	None – Gather – $i2e\frac{Frba}{Bivs+SO}$	minod-count	32,647,165 (91.72s)	32,685,865 (269.84s)
ORY	0626-0702	None – Gather – $i2e\frac{Frba}{Bivs+SO}$	sum	32,647,165 (93.8s)	32,657,565 (291.86s)
ORY	0626-0702	None – Bin – $i2e\frac{Frba}{Bivs+SO}$	sum	32,647,165 (140.09s)	32,656,065 (294.09s)
ORY	0626-0702	None – Bin – $i2e\frac{Frba}{Bivs+SO}$	minod-count	32,647,165 (150.93s)	32,671,665 (296.24s)
ORY	0626-0702	None – Bin – $i2e\frac{Frba}{Bivs+SO}$	minod	32,647,165 (152.56s)	32,671,665 (299.12s)
ORY	0626-0702	None – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod	32,647,165 (160.99s)	32,647,265 (238.15s)
ORY	0626-0702	None – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod-count	32,647,165 (193.75s)	32,647,265 (278.93s)
ORY	0626-0702	Part – Gather $\frac{Frba}{SO}$	minod	32,633,991 (37.64s)	33,449,331 (287.21s)
ORY	0626-0702	Part – Gather $\frac{Frba}{SO}$	sum	32,103,070 (13.58s)	33,450,226 (292.27s)
ORY	0626-0702	Full – Gather $\frac{Frba}{SO}$	minod-count	30,384,797 (34.46s)	30,896,186 (293.08s)
ORY	0626-0702	Full – Bin – $i2e\frac{Frba}{Bivs+SO}$	minod-count	30,182,236 (48.69s)	30,525,845 (272.61s)
ORY	0626-0702	Full – Gather $\frac{Frba}{SO}$	minod	30,161,682 (31.0s)	30,896,186 (299.05s)
ORY	0626-0702	Full – ADEWeak – $i2e\frac{Frba}{Bivs+SO}$	minod	30,120,457 (36.85s)	30,568,135 (299.77s)
ORY	0626-0702	Full – Bin – $i2e\frac{Frba}{Bivs+SO}$	minod	30,057,457 (43.31s)	30,525,845 (290.66s)

Continued on next page

Term	Period	Solver	Model	First B (Time (s))	Last B (Time (s))
ORY	0626-0702	Full – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	30,053,461 (32.92s)	30,523,145 (299.27s)
ORY	0626-0702	Full – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	30,048,261 (38.44s)	30,534,341 (296.71s)
ORY	0626-0702	Full – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	30,027,561 (28.2s)	30,523,045 (299.78s)
ORY	0626-0702	Full – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	29,730,470 (13.38s)	30,895,486 (296.88s)
ORY	0626-0702	Full – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	TO	TO
ORY	0626-0702	Full – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	TO	TO
ORY	0626-0702	Full – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	TO	TO
ORY	0626-0702	None – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	TO	TO
ORY	0626-0702	None – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	TO	TO
ORY	0626-0702	None – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	TO	TO
ORY	0626-0702	None – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	TO	TO
ORY	0703-0709	Part – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	32,679,591 (58.39s)	32,939,296 (299.9s)
ORY	0703-0709	Part – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	32,674,125 (41.67s)	33,481,322 (298.45s)
ORY	0703-0709	Part – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	32,663,697 (44.72s)	32,967,696 (298.38s)
ORY	0703-0709	Part – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	32,658,397 (38.91s)	33,052,096 (297.65s)
ORY	0703-0709	Part – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	32,648,897 (29.49s)	32,967,596 (297.58s)
ORY	0703-0709	Part – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	32,635,502 (53.21s)	33,019,496 (298.88s)
ORY	0703-0709	None – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	32,560,697 (70.77s)	32,652,797 (286.03s)
ORY	0703-0709	None – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	32,560,697 (87.08s)	32,590,897 (295.06s)
ORY	0703-0709	None – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	32,560,697 (96.08s)	32,638,597 (282.04s)
ORY	0703-0709	None – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	32,560,697 (111.88s)	32,629,997 (290.62s)
ORY	0703-0709	None – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	32,560,697 (114.06s)	32,629,997 (292.74s)
ORY	0703-0709	None – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	32,560,697 (142.22s)	32,561,697 (293.63s)
ORY	0703-0709	None – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	32,560,697 (193.18s)	32,560,797 (274.56s)
ORY	0703-0709	None – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	32,560,697 (199.15s)	32,560,797 (282.01s)
ORY	0703-0709	Part – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	32,555,397 (69.27s)	32,860,296 (295.61s)
ORY	0703-0709	Part – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	32,540,697 (64.72s)	32,858,196 (298.48s)
ORY	0703-0709	Part – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	32,538,997 (52.46s)	32,826,196 (293.83s)
ORY	0703-0709	Part – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	32,514,897 (30.23s)	32,955,196 (298.55s)
ORY	0703-0709	Part – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	32,397,314 (30.92s)	33,488,122 (290.05s)
ORY	0703-0709	Part – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	32,008,203 (16.03s)	33,464,822 (299.6s)
ORY	0703-0709	Full – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	30,044,177 (32.67s)	30,794,757 (296.17s)
ORY	0703-0709	Full – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	30,007,646 (52.55s)	30,380,652 (299.85s)
ORY	0703-0709	Full – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	30,004,975 (29.73s)	30,793,057 (297.55s)
ORY	0703-0709	Full – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	29,989,643 (40.24s)	30,457,148 (295.19s)
ORY	0703-0709	Full – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	29,975,042 (35.42s)	30,459,648 (291.55s)
ORY	0703-0709	Full – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	29,974,651 (42.21s)	30,390,552 (299.15s)
ORY	0703-0709	Full – Bin – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	29,870,052 (35.0s)	30,352,252 (287.89s)
ORY	0703-0709	Full – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	29,803,152 (19.77s)	30,368,652 (299.52s)
ORY	0703-0709	Full – Gather – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	29,465,253 (12.49s)	30,785,557 (296.14s)
ORY	0703-0709	Full – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	TO	TO
ORY	0703-0709	Full – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	TO	TO
ORY	0703-0709	Full – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	TO	TO
ORY	0703-0709	None – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod	TO	TO
ORY	0703-0709	None – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	minod-count	TO	TO
ORY	0703-0709	None – Card – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	TO	TO
ORY	0703-0709	None – ADEWeak – $i2e_{\text{Frba}}^{\text{Frba}}$	sum	TO	TO

Table 3.9: Results for overlapping-nbmax family.

3.4.8 Comparison with the ADP algorithm

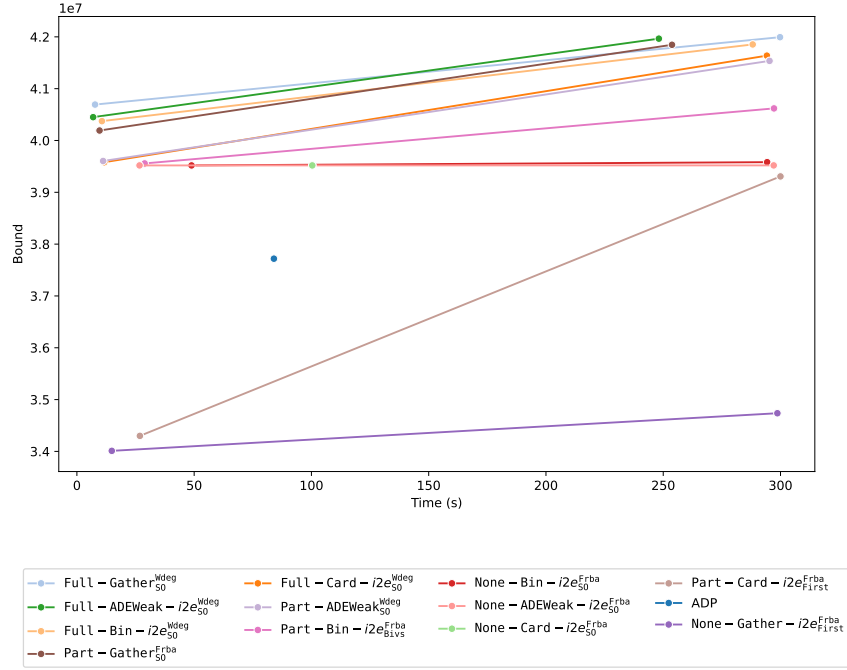
In this section, we compare our top-performing methods (i.e., the methods that have the best score in Tables 3.4, 3.6 and 3.8) against the ADP algorithm. ADP algorithm is a linear method that sequences flights based on descending task count and chronologically by day. It initially employs the resource with the maximum weight. If that resource is unavailable and cannot be overlapped, the algorithm then resorts to the resource with the subsequent lower weight, and so on. This algorithm does not apply swap or backtrack operations.

This algorithm is applied to each scheduled day, applying a similar approach to our partial decomposition. We call this algorithm ADP.

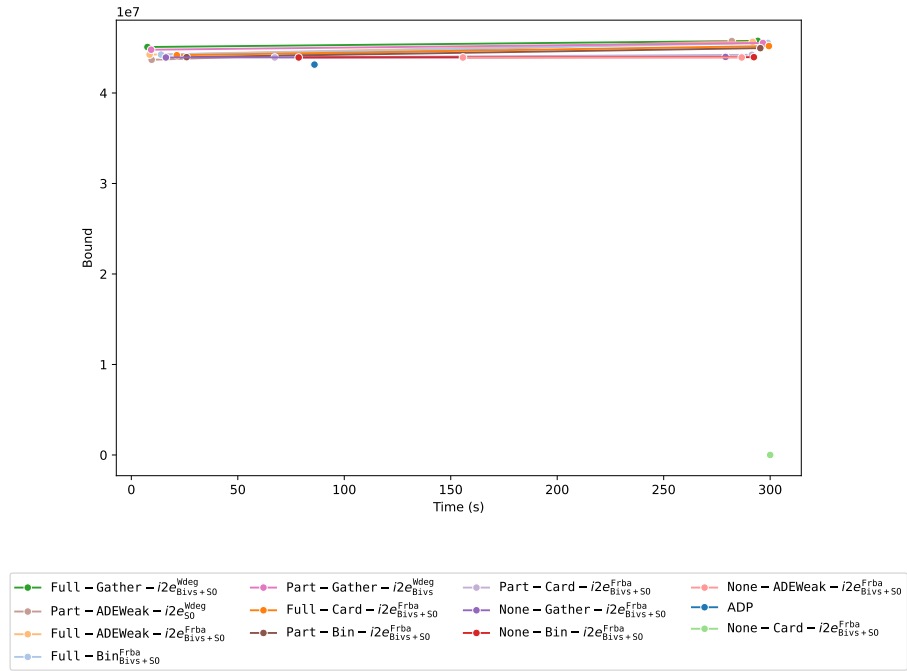
Figure 3.5 comprises three sub-figures, each corresponding to one of the families explored in this chapter for the ORY instance spanning May 8th to May 14th, 2023. Each graph captures the performance of solvers across different decompositions. Individual solvers are depicted by curves, which illustrate the progression of their bounds from the initial to the final values. If a solver requires more than 5 minutes to enhance its first bound, only a single point is depicted. Additionally, the bounds found by the ADP algorithm are also shown.

Figure 3.5a shows all configurations from Table 3.4. Here, one can discern the difficulty encountered by two specific methods, namely $\text{Part-Card-}i2e_{\text{First}}^{\text{Frba}}$ and $\text{None-Gather-}i2e_{\text{First}}^{\text{Frba}}$, in achieving bounds that are greater than the ADP algorithm. Figures 3.5b and 3.5c clearly show that the solver approach outperforms the ADP approach. However, it is worth noting that Figure 3.5c omits methods without decomposition due to their poor performance—a score of 0. This can be explained by the size of the instance to be solved.

Figures 3.6 are similar but for an instance of CDG from August 21st to 27th 2023. For this latter, Figure 3.6a shows that the full decomposition approaches are the only ones that find a better bound than the ADP algorithm. However, the full decomposition approach that uses the **Card** algorithm (original approach from ACE for AllDiffExcept) is not able to obtain a better bound than ADP. Finally, Figure 3.6b shows that when the ADP approach outperforms the solvers approach, the results are relatively close.

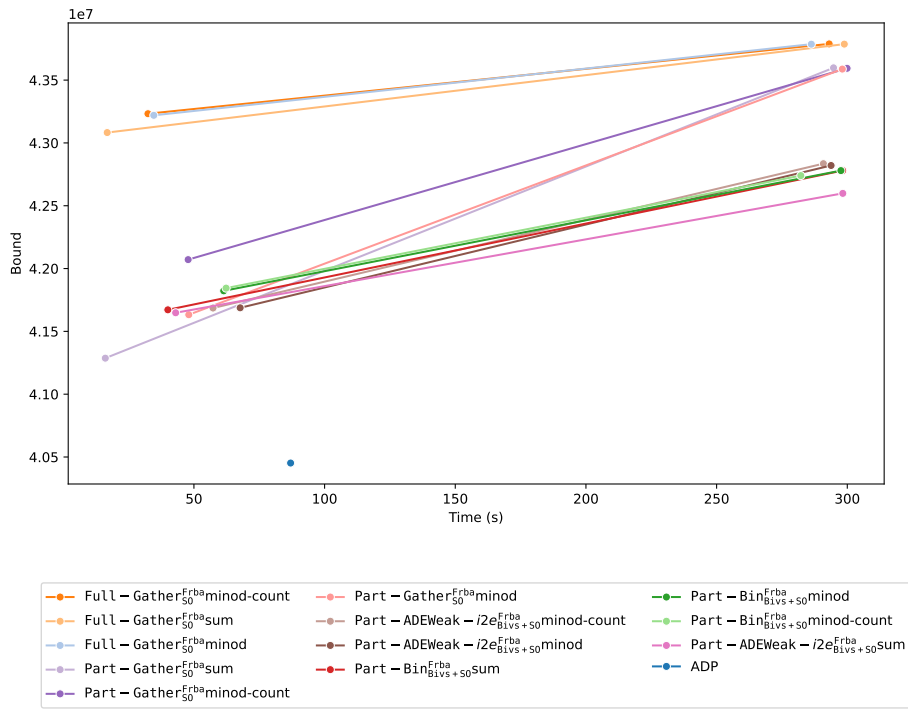


(a) no – overlapping family.



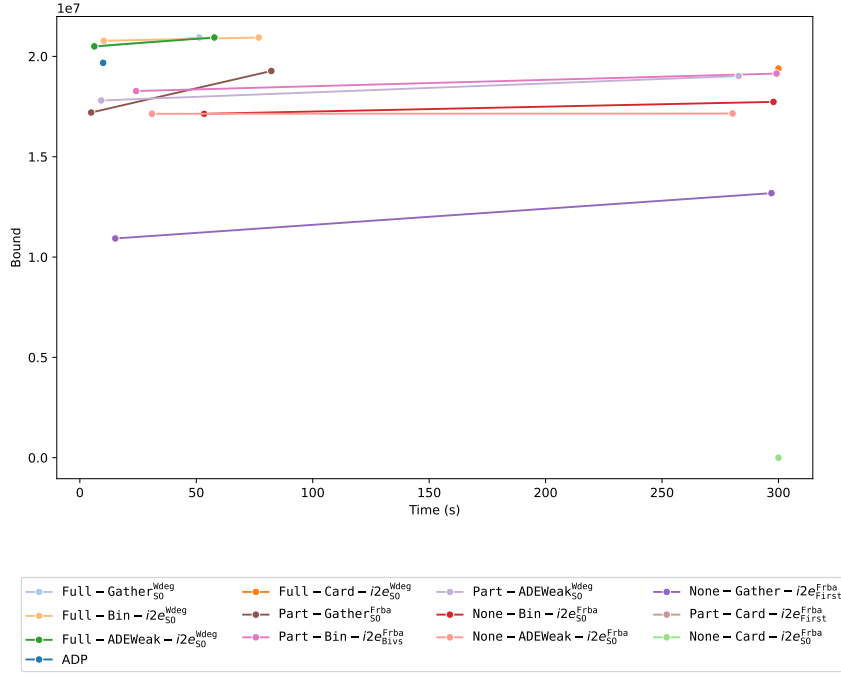
(b) overlapping family.

Figure 3.5: Comparison of our best approaches with ADP algorithm on each family, for instance ORY, from May 8 to 14th 2023 (1/2).

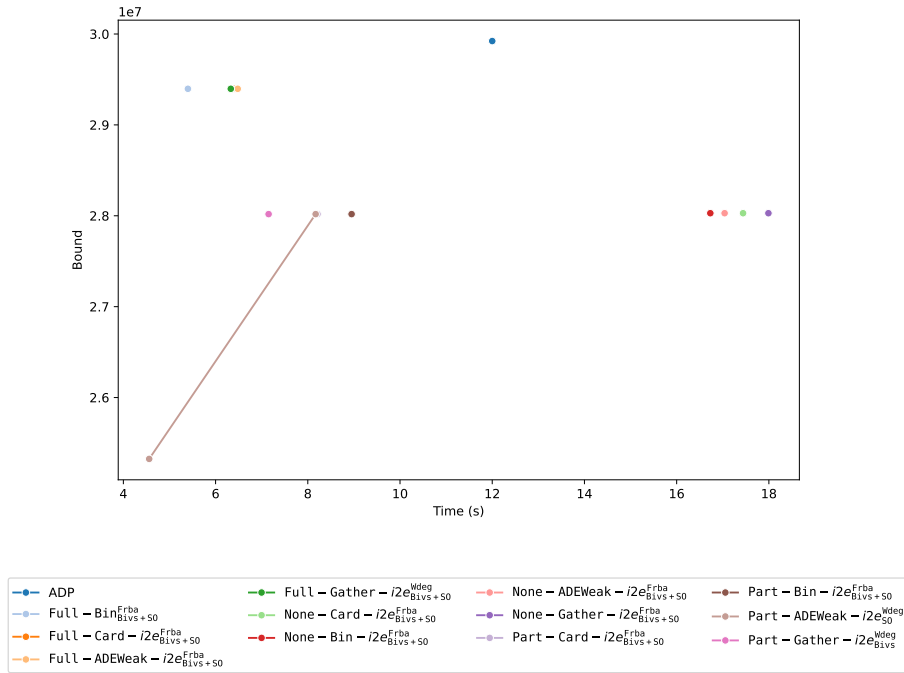


(c) overlapping – nbmax family.

Figure 3.5: Comparison of our best approaches with ADP algorithm on each family, for instance ORY, from May 8 to 14th 2023 (2/2).



(a) no – overlapping family.



(b) overlapping family.

Figure 3.6: Comparison of our best approaches with ADP algorithm on each family, for instance, CDG from August 21st to 27th.

3.5 Conclusion

In this chapter, various models addressing the check-in desk allocation problem have been introduced. We put forward three distinct formulations of the problem: one without overlapping, another with overlapping, and the third, which considers overlapping rules that have a limited number of tasks. We presented an initial model for the third type, followed by two refined versions that utilize `sum` and `count` constraints. Additionally, to efficiently manage the multitude of `AllDifferentExcept` constraints, we integrated them into a single constraint and introduced a corresponding propagator. Subsequently, we conducted an exhaustive analysis of different configurations for the ACE solver, pinpointing the most optimal setups to benchmark against the ADP algorithm.

A recurring theme throughout the chapter was the consistent top-tier performance of the `Gather` method for the `AllDiffExcept` constraint. Regardless of the instance family, solvers employing this method were frequently found at the top of tables.

However, it was about more than just the efficacy of a single method. The chapter illuminated the intricacies and nuances of various solver configurations. While one configuration may excel in a specific scenario, it might falter in another. This variability underlines the importance of understanding problem instances deeply and choosing solver configurations that align with their specific challenges.

Moreover, our comparison with the ADP algorithm served a dual purpose. First, it acted as a reference to validate the performance of our approaches. Second, it highlighted that while simpler algorithms can provide solutions, there is significant value in employing generic solvers regarding performance, the solution's quality, and the system's evolution.

The modeling variants introduced for the *overlapping with a limited number of tasks* family further demonstrated the importance of problem representation. How a problem is modeled can profoundly impact solver performance, making investing time and thought into modeling decisions crucial.

This work is currently being deployed in preview and will be in production by January 2024.

Chapter 4

Modeling of the Airport stand allocation problem

Contents

4.1 Introduction	85
4.2 Modeling of the Stand Allocation Problem	86
4.2.1 Classical variant	88
4.2.2 AllDifferent variant	90
4.2.3 not-break variant	91
4.3 Introduction	92
4.4 Modeling of the Stand Allocation Problem	92
4.4.1 Classical variant	94
4.4.2 AllDifferent variant	96
4.4.3 not-break variant	97
4.5 Experiments	98
4.5.1 Instances	98
4.5.2 classical vs alldiff modeling	98
4.5.3 not-break modeling	100
4.6 Conclusion	103
4.7 Experiments	103
4.7.1 Instances	103
4.7.2 classical vs alldiff modeling	104
4.7.3 not-break modeling	104
4.8 Conclusion	108

4.1 Introduction

Airports serve as vital hubs in the aviation sector, bridging passengers with their intended flights and destinations. With the surge in air travel, refining airport procedures becomes indispensable. A significant challenge lies in allocating aircraft stands to specific airlines and their respective flights, given myriad constraints and goals, including the satisfaction of airlines.

This chapter presents different **modeling approaches** for the stand allocation problem. Like the previous chapter, we start by formally describing the model developed for the *Stand Allocation Problem* (SAP) in a higher “mathematical” form.

Subsequently, it juxtaposes diverse approaches and solver configurations against the existing methods employed by ADP. The analysis of all experimental findings, facilitated by Metrics, is accessible online.

4.2 Modeling of the Stand Allocation Problem

Table 4.5 recall the notation for the stand allocation problem presented for this problem in Section 1.4.2 (page. 5).

Category	Notation	Description
Stand	p PK	Stand. The set of all aircraft stands.
Rotations	ϕ Φ Φ_n a_i and d_i	A registration. The set of all rotations. The set of all rotations with n tasks. The registration’s start and end times.
Tasks	ϕ_i $\mathcal{T}$ $\mathcal{T}_\phi$ PK_i or PK_ϕ $\mathcal{O}_i$ or $\mathcal{O}_{\phi,i}$	i th task (operation) of a rotation ϕ . Set of all operations. Set of tasks of the registration ϕ The set of compatible aircraft stands for the task i or rotation ϕ . The set of operations overlapping with either task i or task i of ϕ .
Constraints	$\mathcal{Q}$ $\mathcal{D}$ $\mathcal{MP}$	The set of shading constraints. The set of reductions constraints. The affinity matrix.

Table 4.1: Notations for the stand allocation problem.

Firstly, we need to introduce the variables of our model. Actually, in addition to a stand-alone variable used to count the number of rotations that are not broken, we need two (2-dimensional) arrays of variables to represent assigned stand and associated rewards:

- p is a matrix of $|\Phi| \times 3$ variables having the set of values $\{0, \dots, |PK| - 1\}$ as domain; $p[\phi, j]$ represents the index (code) of the stand assigned to the j th task of the rotation ϕ .
- r is a matrix of $|\Phi| \times 3$ variables having the set of values $\{0, \dots, 100\}$ as domain; $r[\phi, j]$ represents the satisfaction of the company for the j th task of the rotation ϕ .

Note that, for simplicity, we constantly introduce three variables in p and r for each rotation, even if only one task (or two tasks) is involved. We introduce equality constraints to deal with such cases, forcing some variables to take the same value. In practice, one could avoid introducing such redundant variables.

We now introduce constraints for this problem. An illustration is provided to facilitate understanding of the model’s various constraints.

Example 24

Let us consider an example with a set of 5 rotations $\{\phi_1, \dots, \phi_5\}$, and a set of 4 stands $\{p_1, \dots, p_4\}$. where p_3 and p_4 are assumed to be remote stand; so, we have $PK_{rt} = \{p_4, p_5\}$. Information concerning rotations is given in Table 4.6. Figure 4.3 displays information given in Table 4.6 using an interval chart.

Rot.	airline	ntasks	kind	a_ϕ	d_ϕ	Pkg	Capacity
ϕ_1	a_1	1	k_1	8h	10h	p_1	$\{k_1, k_2\}$
ϕ_2	a_2	2	k_2	8h	12h	p_2	$\{k_2, k_3\}$
ϕ_3	a_2	2	k_2	12h	16h	p_3	$\{k_1, k_2, k_3\}$
ϕ_4	a_3	3	k_3	9h	15h	p_4	$\{k_1, k_2, k_3, k_4\}$
ϕ_5	a_3	3	k_4	12h	18h	p_5	$\{k_1, k_2, k_3, k_4\}$

Table 4.2: Data about Rotations

Table 4.3: Capacity

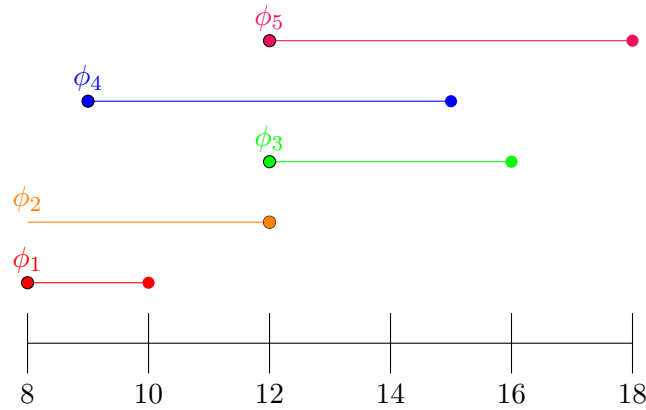


Figure 4.1: Time intervals from Table 4.6

4.2.1 Classical variant

Modelization 8 (Classical variant)

$$p[\phi, 1] = p[\phi, 2] = p[\phi, 3], \forall \phi \in \Phi_1 \quad (C_{13})$$

$$p[\phi, 2] = p[\phi, 3], \forall \phi \in \Phi_2 \quad (C_{14})$$

$$p[\phi, 2] \in PK_{remote}, \forall \phi \in \Phi_3 \quad (C_{15})$$

$$\langle p[\phi, j] \rangle \in PK_\phi, \forall \phi \in \Phi, \forall j \in 1..3, \quad (C_{16})$$

$$\langle p[\phi_1, i], p[\phi_2, j] \rangle \notin \{(\mathbf{p}_1, \mathbf{p}_2,), \forall (\mathbf{p}_1, \mathbf{p}_2, t_{\phi_1, i}, t_{\phi_2, j}) \in \mathcal{Q}\} \quad (C_{17})$$

$$\begin{aligned} &\langle p[\phi, i], p[\phi', j] \rangle \notin \{(\mathbf{p}_1, \mathbf{p}_2,) \mid \mathbf{p}_2 \in \phi\}, \\ &\forall \phi \in \Phi, \forall i \in 1..ntasks(\phi), \\ &\forall \langle k, p, k', \phi \rangle \in \mathcal{D} \mid k = \mathbf{kind}(\phi), \\ &\forall (\phi', j) \in \mathcal{O}_{\phi, i} \mid k' \neq \mathbf{kind}(\phi') \end{aligned} \quad (C_{18})$$

$$\langle p[\phi, j], r[\phi, j] \rangle \in \{(\mathbf{p}_1, r_{\phi, j}^{\mathbf{p}_1}) \mid \mathbf{p}_1 \in PK_\phi\}, \forall \phi \in \Phi, \forall j \in 1..ntasks(\phi) \quad (C_{19})$$

First, we need to introduce some stand constraints concerning the rotation tasks. When splitting a rotation into two or three tasks cannot be done (because rotation time is not large enough), we need to ensure that the plane remains on the same stand (because, for simplicity, as already mentioned, we always have three variables being introduced for each rotation). When three tasks are required, the second stand (for the second task) must be a remote one.

Constraint C_{20} forces all three variables of ϕ to be assigned to the same parking (because the rotation involves only one task). Constraint C_{21} forces the second and third variables of ϕ to be assigned to the same parking (because the rotation involves only two tasks). Finally, Constraint C_{22} forces the middle parking to be remote.

Example 25

For our example :

- $\Phi_1 = \{\phi_1\}$
- $\Phi_2 = \{\phi_2, \phi_3\}$
- $\Phi_3 = \{\phi_4, \phi_5\}$

So we must post the following constraints:

- $p[\phi_1, 1] = p[\phi_1, 2] = p[\phi_1, 3]$
- $p[\phi_2, 2] = p[\phi_2, 3]$
- $p[\phi_3, 2] = p[\phi_3, 3]$
- $p[\phi_4, 2] \in PK_{remote}$
- $p[\phi_5, 2] \in PK_{remote}$

Capacity rules were described in Section 1.4.2 (page. 5). To enforce them, we can post unary constraints (see Constraint C_{23}).

Example 26

Table 4.7 provides each parking capacity for our example. Based on this Table, we must post the following unary constraints:

- $p[\phi_1, j] \in \{p_1, p_3, p_4, p_5\}, \forall j \in 1..3$
- $p[\phi_2, j] \in \{p_1, p_3, p_4, p_5\}, \forall j \in 1..3$
- $p[\phi_3, j] \in \{p_1, p_3, p_4, p_5\}, \forall j \in 1..3$
- $p[\phi_4, j] \in \{p_2, p_3, p_4, p_5\}, \forall j \in 1..3$
- $p[\phi_5, j] \in \{p_2, p_4, p_5\}, \forall j \in 1..3$

Let us recall that when a parking p_1 is shaded by a parking p_2 then these two values cannot be assigned together to any pair of overlapping tasks. This leads to binary negative table constraints. Although not explicitly shown below, assigning the same value twice for any pair of overlapping tasks is also forbidden (see Constraint C_{24}).

The last subset is the constraint **reduction** and is defined with binary negative tables (see Constraint C_{25}).

Example 27

For our example, a **reduction** constraint exists between the stands p_1 and p_2 . Suppose that $\mathcal{D} = \{(k_1, p_1, k_3, \{p_2\}), (k_2, p_1, k_4, \{p_2\})\}$. We also have $\mathcal{O}_{\phi_1,1} = \{(\phi_2, 1), (\phi_4, 1)\}$ and $PK_{\phi_1} = \{p_1, p_3, p_4\}$.

Task 1 from rotation ϕ_2 overlap task 1 of the rotation ϕ_2 . Recall that $\text{kind}(\phi_2) = k_2$. So, for the reduction ϕ_1 , we add the constraint that forbidden the pair of values (p_1, p_2) for the pair of variable $\langle p[\phi_1, 1], p[\phi_2, 1] \rangle$. In other words, it is forbidden to use p_2 with the rotation ϕ_2 because ϕ_2 does not have the allowed kind by p_2 after placing ϕ_1 on p_1 (its capacity is reduced).

$$\langle p[\phi_1, 1], p[\phi_2, 1] \rangle \notin \{(p_1, p_2)\}$$

Although ϕ_1 also overlaps with ϕ_4 , there are no restrictions to consider with ϕ_1 as ϕ_4 has a capacity of type k_3 (see Table 4.6) which is allowed in relation to the reduction.

According to the airlines preferences from affinity matrix $\mathcal{MP}$, we can post binary table constraints to “compute” rewards when filtering such constraints (see Constraint C_{26}).

Airline / Parking	p_1	p_2	p_3	p_4	p_5
a_1	75	75	100	50	50
a_2	60	0	100	50	50
a_3	0	100	80	50	50

Table 4.4: Affinity matrix

Example 28

For our example, let us assume that rewards are given by the matrix 4.8.

From the data in this table, we post the following constraint for the first rotation (the same principle is adopted for the other rotations):

$$\langle p[\phi_1, j], r[\phi_1, j] \rangle \in \{(p_1, 75), (p_2, 75), (p_3, 100), (p_4, 50), (p_5, 50)\}, \forall j \in 1..\text{ntasks}(\phi_1)$$

Considering the reward variables we can express the objective function as follows:

$$\text{maximize} \quad \sum_{\substack{\phi \in \Phi \\ j \in 1..\text{ntasks}(\phi)}} w_{\phi,j} \times r[\phi, j]$$

4.2.2 AllDifferent variant

In this variant, we propose to modify the shading constraint by posting **Alldifferent** constraints in addition to the table constraints already proposed. Note that for this constraint, it is tempting to want to use an **AllDifferent** constraint as already proposed in the state-of-the-art

[Sim07, DS91]. In these approaches, an **AllDifferent** constraint is added between all overlapping pairs of tasks, but this cannot work here, as it would be less restrictive than the **shadow** constraint. Indeed, an **AllDifferent** constraint would force two operations $p[\phi, i]$ and $p[\phi', j]$ to be different. However, by setting p_1 and p_2 as values for these two variables, the constraint would be respected but would violate the shadow constraint if p_1 shaded p_2 .

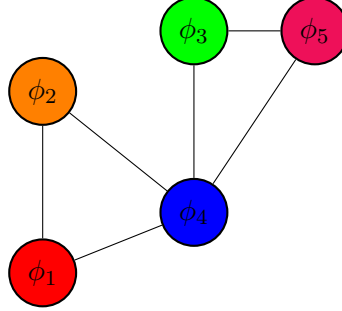


Figure 4.2: Interval graph based on data from Table 4.6.

Nevertheless, it is possible to use **AllDifferent** constraints with an interval graph. Each vertex of the graph represents a task (the task's time interval) and is connected by an edge to another vertex if and only if there is an overlap between the two intervals. Figure 4.4 represents an interval graph $\mathcal{G}$ for our example, each vertex represents an interval and an edge between intervals whose intervals intersect. From this graph, we need to post an **AllDifferent** constraint for each *maximum clique* in the graph.

Definition 43 (Maximum clique)

A maximum clique of $\mathcal{G}$ has the greatest number of vertices, which is maximal for the cardinal. We note $\mathcal{C}_{\mathcal{G}}$ the set of all maximum cliques.

For the set of maximum cliques, we can post the following **AllDifferent** constraints:

$$\text{allDifferent}(\{p[\phi, i], \forall (\phi, i) \in c\}), \forall c \in \mathcal{C}_{\mathcal{G}} \quad (4.1)$$

For our example and based on our interval graph (Figure 4.4), the maximum cliques are: $\{\phi_1, \phi_2, \phi_4\}$ and $\{\phi_3, \phi_4, \phi_5\}$.

4.2.3 not-break variant

This variant of the problem considers that moving an aircraft from one parking lot to another has a specific cost. For this reason, the number of stands used for one rotation (with 2 or 3 tasks) must be minimized. To do that, ADP uses a post-treatment algorithm to join flights with a break (i.e., break in 2 or 3 tasks). Because ACE does not support the multi-objectives, we propose integrating this feature directly into the objective function. # Modeling of the Airport stand allocation problem

4.3 Introduction

Airports serve as vital hubs in the aviation sector, bridging passengers with their intended flights and destinations. With the surge in air travel, refining airport procedures becomes indispensable. A significant challenge lies in allocating aircraft stands to specific airlines and their respective flights, given myriad constraints and goals, including the satisfaction of airlines.

This chapter presents different **modeling approaches** for the stand allocation problem. Like the previous chapter, we start by formally describing the model developed for the *Stand Allocation Problem* (SAP) in a higher “mathematical” form.

Subsequently, it juxtaposes diverse approaches and solver configurations against the existing methods employed by ADP. The analysis of all experimental findings, facilitated by Metrics, is accessible online.

4.4 Modeling of the Stand Allocation Problem

Table 4.5 recall the notation for the stand allocation problem presented for this problem in Section 1.4.2 (page. 5).

Category	Notation	Description
Stand	p PK	Stand. The set of all aircraft stands.
Rotations	ϕ Φ Φ_n a_i and d_i	A registration. The set of all rotations. The set of all rotations with n tasks. The registration’s start and end times.
Tasks	ϕ_i $\mathcal{T}$ $\mathcal{T}_\phi$ PK_i or PK_ϕ $\mathcal{O}_i$ or $\mathcal{O}_{\phi,i}$	i th task (operation) of a rotation ϕ . Set of all operations. Set of tasks of the registration ϕ The set of compatible aircraft stands for the task i or rotation ϕ . The set of operations overlapping with either task i or task i of ϕ .
Constraints	$\mathcal{Q}$ $\mathcal{D}$ $\mathcal{MP}$	The set of shading constraints. The set of reductions constraints. The affinity matrix.

Table 4.5: Notations for the stand allocation problem.

Firstly, we need to introduce the variables of our model. Actually, in addition to a stand-alone variable used to count the number of rotations that are not broken, we need two (2-dimensional) arrays of variables to represent assigned stand and associated rewards:

- p is a matrix of $|\Phi| \times 3$ variables having the set of values $\{0, \dots, |PK| - 1\}$ as domain; $p[\phi, j]$ represents the index (code) of the stand assigned to the j th task of the rotation ϕ .
- r is a matrix of $|\Phi| \times 3$ variables having the set of values $\{0, \dots, 100\}$ as domain; $r[\phi, j]$ represents the satisfaction of the company for the j th task of the rotation ϕ .

Note that, for simplicity, we always introduce three variables in p and r for each rotation, even if only one task (or two tasks) is involved. We introduce equality constraints to deal

with such cases, forcing some variables to take the same value. In practice, one could avoid introducing such redundant variables.

We now introduce constraints for this problem. An illustration is now given to facilitate the understanding of the various constraints of the model.

Example 29

Let us consider an example with a set of 5 rotations $\{\phi_1, \dots, \phi_5\}$, and a set of 4 stands $\{p_1, \dots, p_4\}$. where p_3 and p_4 are assumed to be remote stand; so, we have $PK_{rt} = \{p_4, p_5\}$. Information concerning rotations is given in Table 4.6. Figure 4.3 displays information given in Table 4.6 using an interval chart.

Rot.	airline	ntasks	kind	a_ϕ	d_ϕ	Pkg	Capacity
ϕ_1	a_1	1	k_1	8h	10h	p_1	$\{k_1, k_2\}$
ϕ_2	a_2	2	k_2	8h	12h	p_2	$\{k_2, k_3\}$
ϕ_3	a_2	2	k_2	12h	16h	p_3	$\{k_1, k_2, k_3\}$
ϕ_4	a_3	3	k_3	9h	15h	p_4	$\{k_1, k_2, k_3, k_4\}$
ϕ_5	a_3	3	k_4	12h	18h	p_5	$\{k_1, k_2, k_3, k_4\}$

Table 4.6: Data about Rotations

Table 4.7: Capacity

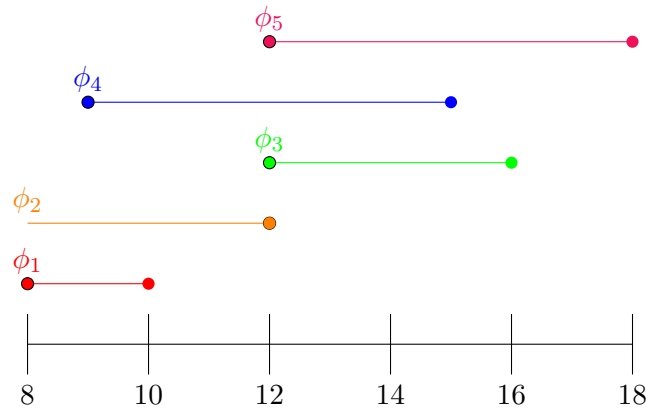


Figure 4.3: Time intervals from Table 4.6

4.4.1 Classical variant

Modelization 9 (Classical variant)

$$p[\phi, 1] = p[\phi, 2] = p[\phi, 3], \forall \phi \in \Phi_1 \quad (C_{20})$$

$$p[\phi, 2] = p[\phi, 3], \forall \phi \in \Phi_2 \quad (C_{21})$$

$$p[\phi, 2] \in PK_{remote}, \forall \phi \in \Phi_3 \quad (C_{22})$$

$$\langle p[\phi, j] \rangle \in PK_\phi, \forall \phi \in \Phi, \forall j \in 1..3, \quad (C_{23})$$

$$\langle p[\phi_1, i], p[\phi_2, j] \rangle \notin \{(\mathbf{p}_1, \mathbf{p}_2,), \forall (\mathbf{p}_1, \mathbf{p}_2, t_{\phi_1, i}, t_{\phi_2, j}) \in \mathcal{Q}\} \quad (C_{24})$$

$$\begin{aligned} &\langle p[\phi, i], p[\phi', j] \rangle \notin \{(\mathbf{p}_1, \mathbf{p}_2,) \mid \mathbf{p}_2 \in \phi\}, \\ &\forall \phi \in \Phi, \forall i \in 1..ntasks(\phi), \\ &\forall \langle k, p, k', \phi \rangle \in \mathcal{D} \mid k = \mathbf{kind}(\phi), \\ &\forall (\phi', j) \in \mathcal{O}_{\phi, i} \mid k' \neq \mathbf{kind}(\phi') \end{aligned} \quad (C_{25})$$

$$\langle p[\phi, j], r[\phi, j] \rangle \in \{(\mathbf{p}_1, r_{\phi, j}^{\mathbf{p}_1}) \mid \mathbf{p}_1 \in PK_\phi\}, \forall \phi \in \Phi, \forall j \in 1..ntasks(\phi) \quad (C_{26})$$

First, we need to introduce some stand constraints concerning the rotation tasks. When splitting a rotation into two or three tasks cannot be done (because rotation time is not large enough), we need to ensure that the plane remains on the same stand (because, for simplicity, as already mentioned, we always have three variables being introduced for each rotation). When three tasks are required, the second stand (for the second task) must be a remote one.

Constraint C_{20} forces all three variables of ϕ to be assigned to the same parking (because the rotation involves only one task). Constraint C_{21} forces the second and third variables of ϕ to be assigned to the same parking (because the rotation involves only two tasks). Finally, Constraint C_{22} forces the middle parking to be remote.

Example 30

For our example :

- $\Phi_1 = \{\phi_1\}$
- $\Phi_2 = \{\phi_2, \phi_3\}$
- $\Phi_3 = \{\phi_4, \phi_5\}$

So we must post the following constraints:

- $p[\phi_1, 1] = p[\phi_1, 2] = p[\phi_1, 3]$
- $p[\phi_2, 2] = p[\phi_2, 3]$
- $p[\phi_3, 2] = p[\phi_3, 3]$
- $p[\phi_4, 2] \in PK_{remote}$
- $p[\phi_5, 2] \in PK_{remote}$

Capacity rules were described in Section 1.4.2 (page. 5). To enforce them, we can post unary constraints (see Constraint C_{23}).

Example 31

Table 4.7 provides each parking capacity for our example. Based on this Table, we must post the following unary constraints:

- $p[\phi_1, j] \in \{p_1, p_3, p_4, p_5\}, \forall j \in 1..3$
- $p[\phi_2, j] \in \{p_1, p_3, p_4, p_5\}, \forall j \in 1..3$
- $p[\phi_3, j] \in \{p_1, p_3, p_4, p_5\}, \forall j \in 1..3$
- $p[\phi_4, j] \in \{p_2, p_3, p_4, p_5\}, \forall j \in 1..3$
- $p[\phi_5, j] \in \{p_2, p_4, p_5\}, \forall j \in 1..3$

Let us recall that when a parking p_1 is shaded by a parking p_2 then these two values cannot be assigned together to any pair of overlapping tasks. This leads to binary negative table constraints. Although not explicitly shown below, assigning the same value twice for any pair of overlapping tasks is also forbidden (see Constraint C_{24}).

The last subset is the constraint **reduction** and is defined with binary negative tables (see Constraint C_{25}).

Example 32

For our example, a **reduction** constraint exists between the stands $\mathbf{p}_1$ and $\mathbf{p}_2$. Suppose that $\mathcal{D} = \{(k_1, \mathbf{p}_1, k_3, \{\mathbf{p}_2\}), (k_2, \mathbf{p}_1, k_4, \{\mathbf{p}_2\})\}$. We also have $\mathcal{O}_{\phi_1,1} = \{(\phi_2, 1), (\phi_4, 1)\}$ and $PK_{\phi_1} = \{\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4\}$.

Task 1 from rotation ϕ_2 overlap task 1 of the rotation ϕ_2 . Recall that $\mathbf{kind}(\phi_2) = k_2$. So, for the reduction ϕ_1 , we add the constraint that forbidden the pair of values $(\mathbf{p}_1, \mathbf{p}_2)$ for the pair of variable $\langle p[\phi_1, 1], p[\phi_2, 1] \rangle$. In other words, it is forbidden to use $\mathbf{p}_2$ with the rotation ϕ_2 because ϕ_2 does not have the allowed kind by $\mathbf{p}_2$ after placing ϕ_1 on $\mathbf{p}_1$ (its capacity is reduced).

$$\langle p[\phi_1, 1], p[\phi_2, 1] \rangle \notin \{(\mathbf{p}_1, \mathbf{p}_2)\}$$

Although ϕ_1 also overlaps with ϕ_4 , there are no restrictions to consider with ϕ_1 as ϕ_4 has a capacity of type k_3 (see Table 4.6) which is allowed in relation to the reduction.

According to the airlines preferences from affinity matrix $\mathcal{MP}$, we can post binary table constraints to “compute” rewards when filtering such constraints (see Constraint C_{26}).

Airline / Parking	$\mathbf{p}_1$	$\mathbf{p}_2$	$\mathbf{p}_3$	$\mathbf{p}_4$	$\mathbf{p}_5$
a_1	75	75	100	50	50
a_2	60	0	100	50	50
a_3	0	100	80	50	50

Table 4.8: Affinity matrix

Example 33

For our example, let us assume that rewards are given by the matrix 4.8.

From the data in this table, we post the following constraint for the first rotation (the same principle is adopted for the other rotations):

$$\langle p[\phi_1, j], r[\phi_1, j] \rangle \in \{(\mathbf{p}_1, 75), (\mathbf{p}_2, 75), (\mathbf{p}_3, 100), (\mathbf{p}_4, 50), (\mathbf{p}_5, 50)\}, \forall j \in 1..\mathbf{ntasks}(\phi_1)$$

Considering the reward variables we can express the objective function as follows:

$$\text{maximize} \quad \sum_{\substack{\phi \in \Phi \\ j \in 1..\mathbf{ntasks}(\phi)}} w_{\phi,j} \times r[\phi, j]$$

4.4.2 AllDifferent variant

In this variant, we propose to modify the shading constraint by posting **Alldifferent** constraints in addition to the table constraints already proposed. Note that for this constraint, it is tempting to want to use an **AllDifferent** constraint as already proposed in the state-of-the-art

[Sim07, DS91]. In these approaches, an **AllDifferent** constraint is added between all overlapping pairs of tasks, but this cannot work here, as it would be less restrictive than the **shadow** constraint. Indeed, an **AllDifferent** constraint would force two operations $p[\phi, i]$ and $p[\phi', j]$ to be different. However, by setting p_1 and p_2 as values for these two variables, the constraint would be respected but would violate the shadow constraint if p_1 shaded p_2 .

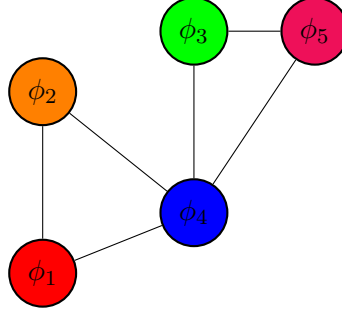


Figure 4.4: Interval graph based on data from Table 4.6.

Nevertheless, it is possible to use **AllDifferent** constraints with an interval graph. Each vertex of the graph represents a task (the task's time interval) and is connected by an edge to another vertex if and only if there is an overlap between the two intervals. Figure 4.4 represents an interval graph $\mathcal{G}$ for our example, each vertex represents an interval and an edge between intervals whose intervals intersect. From this graph, we need to post an **AllDifferent** constraint for each *maximum clique* in the graph.

Definition 44 (Maximum clique)

A maximum clique of $\mathcal{G}$ has the greatest number of vertices, which is maximal for the cardinal. We note $\mathcal{C}_{\mathcal{G}}$ the set of all maximum cliques.

For the set of maximum cliques, we can post the following **AllDifferent** constraints:

$$\text{allDifferent}(\{p[\phi, i], \forall (\phi, i) \in c\}), \forall c \in \mathcal{C}_{\mathcal{G}} \quad (4.2)$$

For our example and based on our interval graph (Figure 4.4), the maximum cliques are: $\{\phi_1, \phi_2, \phi_4\}$ and $\{\phi_3, \phi_4, \phi_5\}$.

4.4.3 not-break variant

This variant of the problem considers that moving an aircraft from one parking lot to another has a specific cost. For this reason, the number of stands used for one rotation (with 2 or 3 tasks) must be minimized. To do that, ADP uses a post-treatment algorithm to join flights with a break (i.e., break in 2 or 3 tasks). Because ACE does not support the multi-objectives, we propose integrating this feature directly into the objective function. We add to our model nb a stand-alone variable with domain $\{0, \dots, |\Phi|\}$ counting the number of not broken rotations.

We must add the following constraint for counting the number of rotations without a break:

$$nb = \sum_{\phi \in \Phi_2} s_{\phi, 1=\phi, 2} + \sum_{\phi \in \Phi_3} s_{\phi, 1=\phi, 2=\phi, 3} \quad (4.3)$$

where $s_{\phi,1=\phi,2}$ is a Boolean variable, which is true if the same stand has been assigned for the first and third tasks in the ϕ rotation, and where $s_{\phi,1=\phi,2=\phi,3}$ is true if all tasks in the ϕ rotation have been assigned to the same stand. The new objective function is defined as follows:

$$\text{maximize} \quad \sum_{\substack{\phi \in \Phi \\ j \in 1..n\text{tasks}(\phi)}} w_{\phi,j} \times r[\phi, j] + \text{nb} \quad (4.4)$$

4.5 Experiments

4.5.1 Instances

Airport	Terminals	Date	#Rotations
CDG	T2B T2D	2023-07-17 2023-07-23	755
CDG	T2B T2D	2023-09-11 2023-09-17	757
CDG	T2B T2D	2023-09-18 2023-09-24	765

Table 4.9: General information about the parking planning.

Table 4.11 presents some factual information about the different planning used for these experiments. The first two columns indicate the area of the planning (i.e., **Airport** and **Terminals** concerning the planning). The third column gives the date of the planning. Finally, the last column displays the number of rotations. For each planning, we have created two families of problems:

- **classical**: This family does contain the constraints as modeling 9.
- **alldiff**: In this family, we add the **allDifferent** constraint based on the maximum cliques (see Section 4.4.2).
- **notbreak**: We change the objective function for integrating the minimization of aircraft movements (see Section 4.4.3).

We use the same set of solvers, noted Ψ_4 , for these experiments based on some variations of ACEX.

$$\left\{ \text{ACEX}_{varh}^{valh} \mid \begin{array}{l} varh \in \{\text{Frba/} \text{dom}, \text{Wdeg}\} \\ valh \in \{\text{first}, \text{Bivs}\} \end{array} \right\} \quad (\Psi_3)$$

4.5.2 classical vs alldiff modeling

We start by comparing the **classical** and the **alldiff** modeling. We create a first set of instances $\mathcal{I}_6$ composed of instances from Table 4.11.

$$\left\{ \mathcal{I}_{parking}^f \mid f \in \{\text{classical}, \text{alldiff}\} \right\} \quad (\mathcal{I}_5)$$

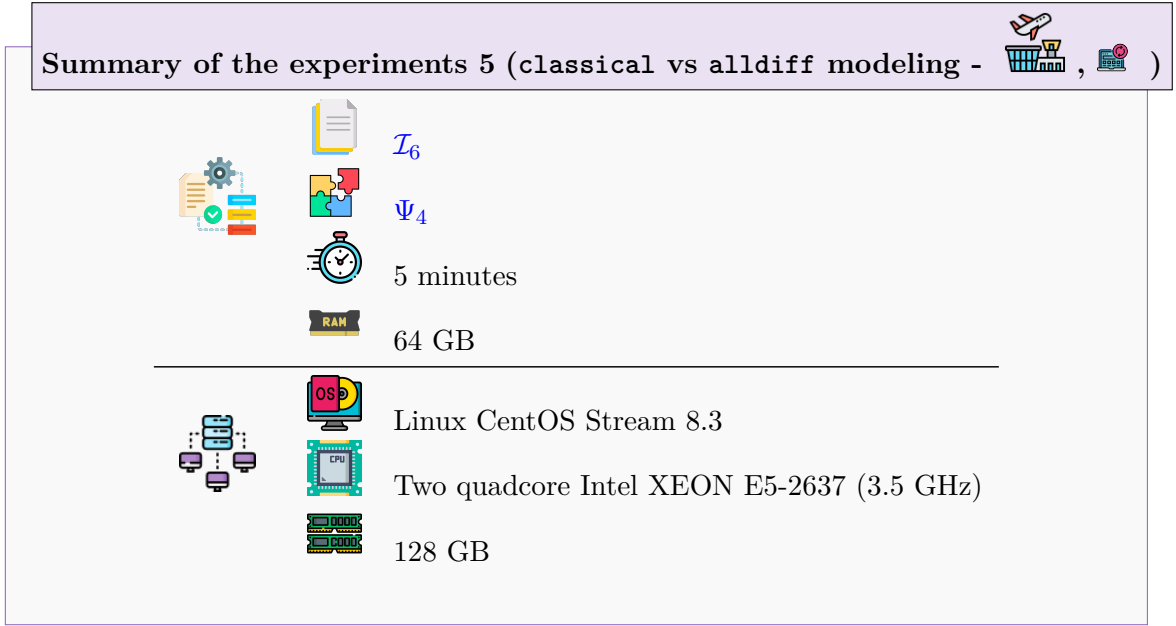


Figure 4.11, 4.12 and 4.16 illustrate the progression of the bounds identified by the solver for each of the tested instances. The x-axis represents time, and the y-axis denotes the bound. The blue dashed line corresponds to the bounds established by the previous system of ADP. For every instance, we consistently achieved a superior bound compared to ADP. However, it is worth noting that the optimal bound might be identified later in the process. We can also observe that the `alldiff` approach (noted `alldiff+extension` on the figure) is very close to the classical approach.

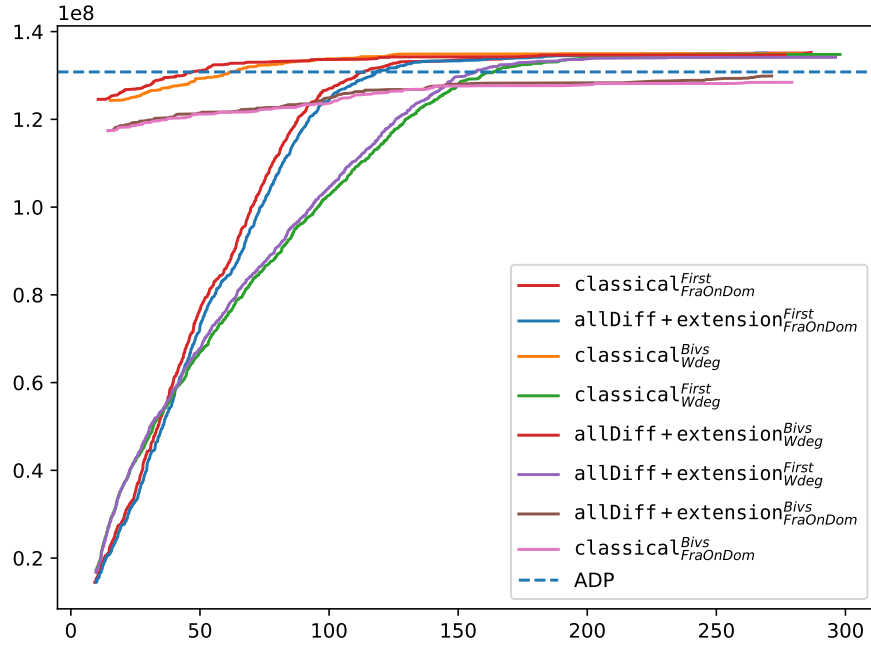


Figure 4.5: CDG T2BD - 2023-07-17 2023-07-23

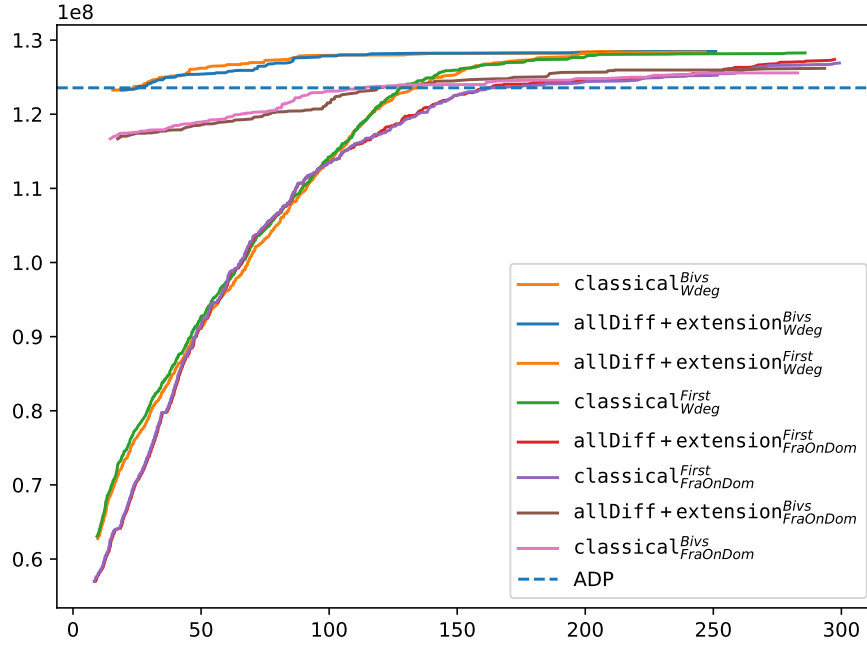


Figure 4.6: CDG T2BD - 2023-09-11 2023-09-17

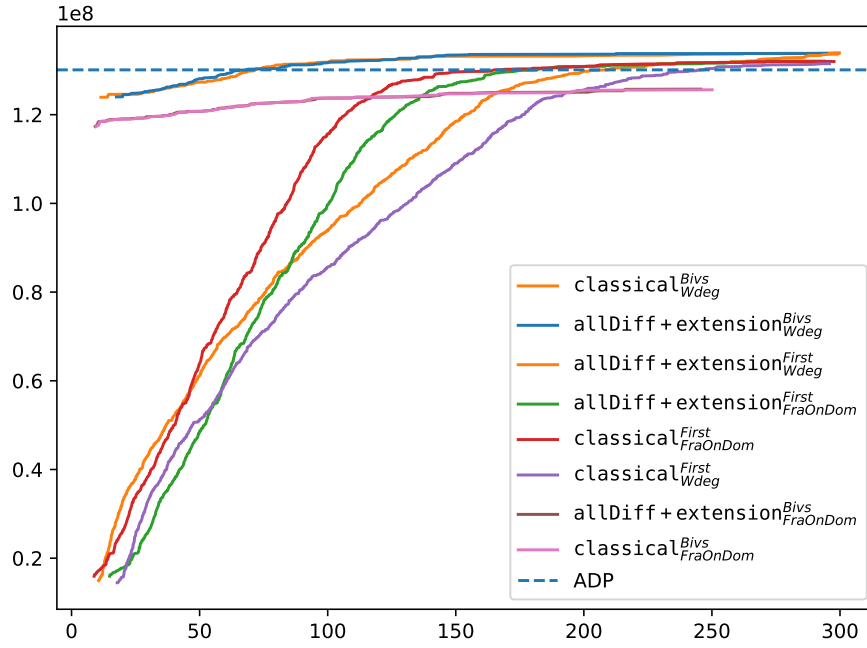


Figure 4.7: CDG T2BD - 2023-09-18 2023-09-24

4.5.3 not-break modeling

Remember, ADP employs a post-processing algorithm to restore rotations without compromising the gains. Instead of this post-treatment, we have embedded this mechanism directly into our

objective function. This section aims to contrast the boundaries acquired after applying both the ADP algorithm and our methodology. Given that the value tied to the count of intact flights is very small, the bounds can be compared to the prior section's method. Consistent with the previous section's findings, every configuration we tested yielded a bound superior to that determined by ADP. Table 4.12 displays the count of rotations that have been reconstructed. In all instances, our configuration enhances the number of reconstructions.

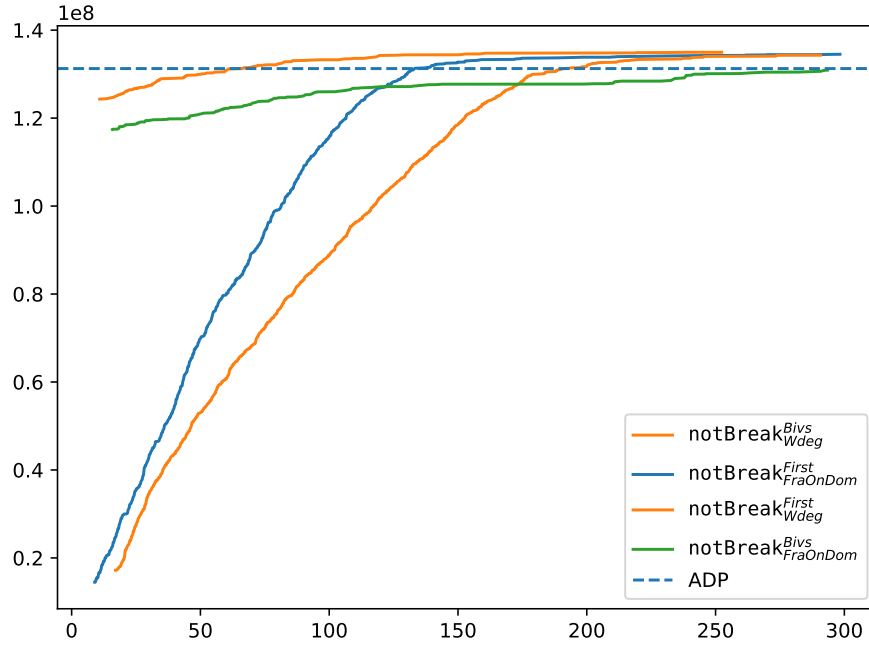


Figure 4.8: CDG T2BD - 2023-07-17 2023-07-23

input	ADP	$\text{notBreak}_{Wdeg}^{First}$	$\text{notBreak}_{Wdeg}^{Bivs}$	$\text{notBreak}_{FraOnDom}^{Bivs}$	$\text{notBreak}_{FraOnDom}^{First}$
CDG S23 WE37 T2BD	3	16	16	16	11
CDG S23 WE38 T2BD	3	8	8	11	7
CDG S23 WE29 T2BD	5	7	6	7	7

Table 4.10: Number of flights reconstitute by each solver and ADP algorithm.

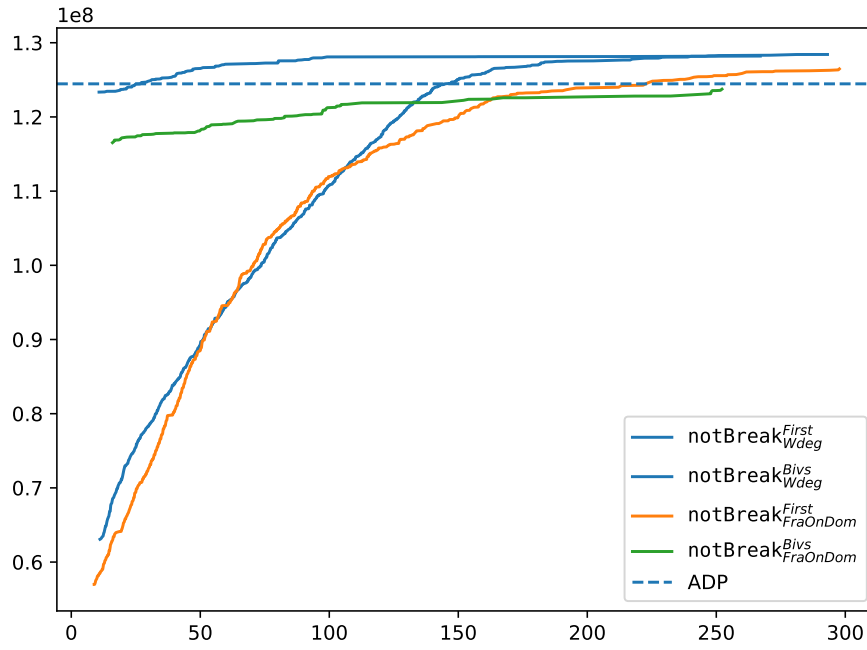


Figure 4.9: CDG T2BD - 2023-09-11 2023-09-17

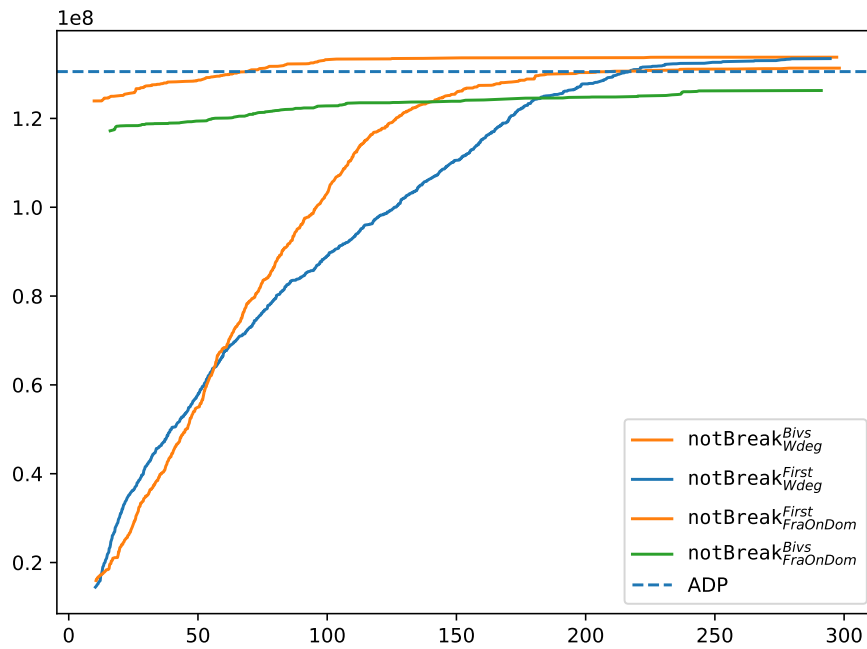


Figure 4.10: CDG T2BD - 2023-09-18 2023-09-24

4.6 Conclusion

In this chapter, various models addressing the stand allocation problem have been introduced. This problem is a fundamental aspect of airport operations that has implications for both efficiency and airline satisfaction. We put forward three distinct formulations of the problem: the **classical** variant which mainly uses extension constraints, **allDifferent** variant, which add **allDifferent** constraints for each maximum clique of the interval graph formed from overlapping tasks, and finally, the **notBreak** variant, which seeks not only to maximize airline satisfaction but also to reduce the number of aircraft movements.

Subsequently, we analyzed different configurations for the **ACE** solver and compared the results with the actual ADP method. The value heuristic order **Bivs** shows excellent performance and always obtains a better bound than that found by ADP.

It is crucial to note that this chapter represents a proof of concept work. While it provides valuable foundational understanding and initial methodologies, there is a pressing need for further optimization. We add to our model *nb* a stand-alone variable with domain $\{0, \dots, |\Phi|\}$ counting the number of not broken rotations.

We must add the following constraint for counting the number of rotations without a break:

$$\text{nb} = \sum_{\phi \in \Phi_2} s_{\phi,1=\phi,2} + \sum_{\phi \in \Phi_3} s_{\phi,1=\phi,2=\phi,3} \quad (4.5)$$

where $s_{\phi,1=\phi,2}$ is a Boolean variable which is true if the same stand has been assigned for the first and third tasks in the ϕ rotation, and where $s_{\phi,1=\phi,2=\phi,3}$ is true if all tasks in the ϕ rotation have been assigned to the same stand. The new objective function is defined as follows:

$$\text{maximize} \quad \sum_{\substack{\phi \in \Phi \\ j \in 1..n\text{tasks}(\phi)}} w_{\phi,j} \times r[\phi, j] + \text{nb} \quad (4.6)$$

4.7 Experiments

4.7.1 Instances

Airport	Terminals	Date	#Rotations
CDG	T2B T2D	2023-07-17 2023-07-23	755
CDG	T2B T2D	2023-09-11 2023-09-17	757
CDG	T2B T2D	2023-09-18 2023-09-24	765

Table 4.11: General information about the parking planning.

Table 4.11 presents some factual information about the different planning used for these experiments. The first two columns indicate the area of the planning (i.e., **Airport** and **Terminals** concerning the planning). The third column gives the date of the planning. Finally, the last column displays the number of rotations. For each planning, we have created two families of problems:

- **classical**: This family does contain the constraints as modeling 9.
- **alldiff**: In this family, we add the **allDifferent** constraint based on the maximum cliques (see Section 4.4.2).

- **notbreak**: We change the objective function for integrating the minimization of aircraft movements (see Section 4.4.3).

We use the same set of solvers, noted Ψ_4 , for these experiments based on some variations of ACEX.

$$\left\{ \text{ACEX}_{varh}^{valh} \mid \begin{array}{l} varh \in \{\text{Frba/dom, Wdeg}\} \\ valh \in \{\text{first, Bivs}\} \end{array} \right\} \quad (\Psi_4)$$

4.7.2 classical vs alldiff modeling

We start by comparing the **classical** and the **alldiff** modeling. We create a first set of instances $\mathcal{I}_6$ composed of instances from Table 4.11.

$$\left\{ \mathcal{I}_{parking}^f \mid f \in \{\text{classical, alldiff}\} \right\} \quad (\mathcal{I}_6)$$

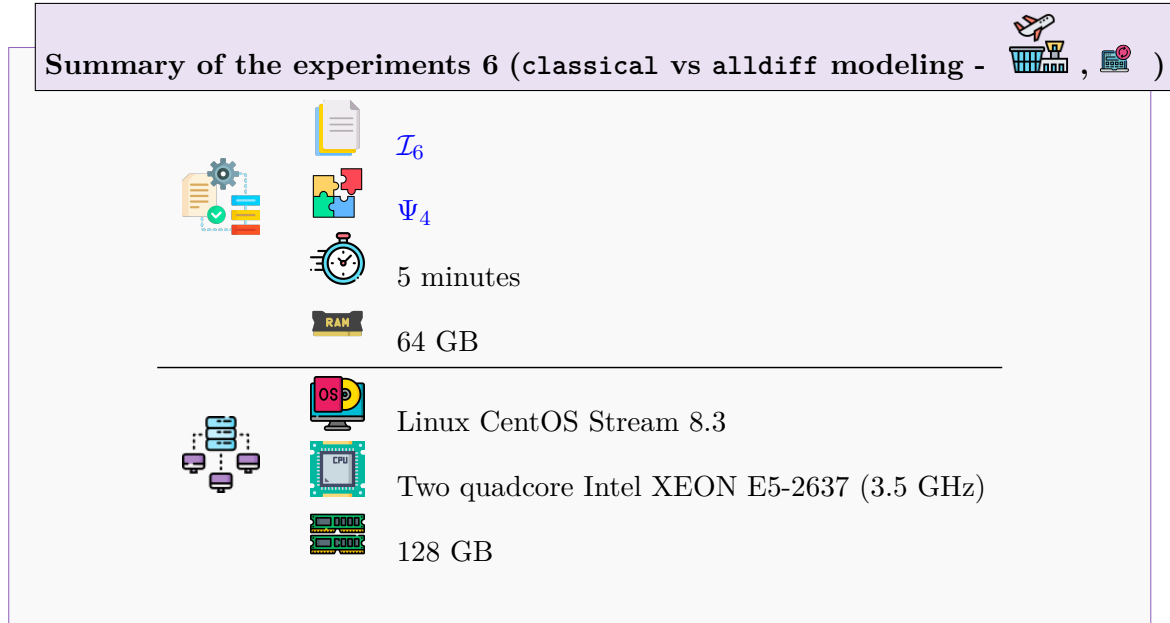


Figure 4.11, 4.12 and 4.16 illustrate the progression of the bounds identified by the solver for each of the tested instances. The x-axis represents time, and the y-axis denotes the bound. The blue dashed line corresponds to the bounds established by the previous system of ADP. For every instance, we consistently achieved a superior bound compared to ADP. However, it is worth noting that the optimal bound might be identified later in the process. We can also observe that the **alldiff** approach (noted **alldiff+extension** on the figure) is very close to the classical approach.

4.7.3 not-break modeling

Remember, ADP employs a post-processing algorithm to restore rotations without compromising the gains. Instead of this post-treatment, we have embedded this mechanism directly into our objective function. This section aims to contrast the boundaries acquired after applying both the ADP algorithm and our methodology. Given that the value tied to the count of unbroken

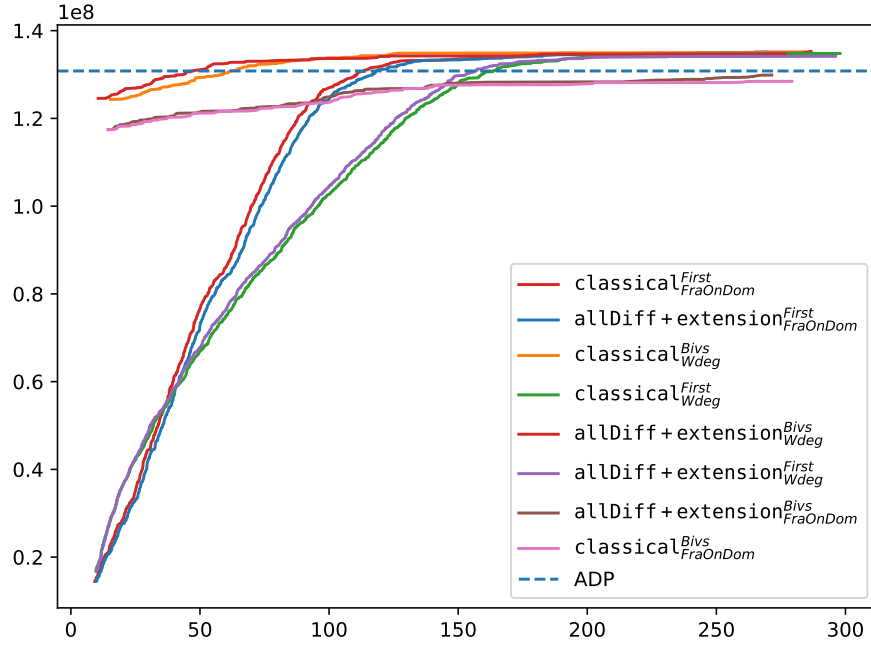


Figure 4.11: CDG T2BD - 2023-07-17 2023-07-23

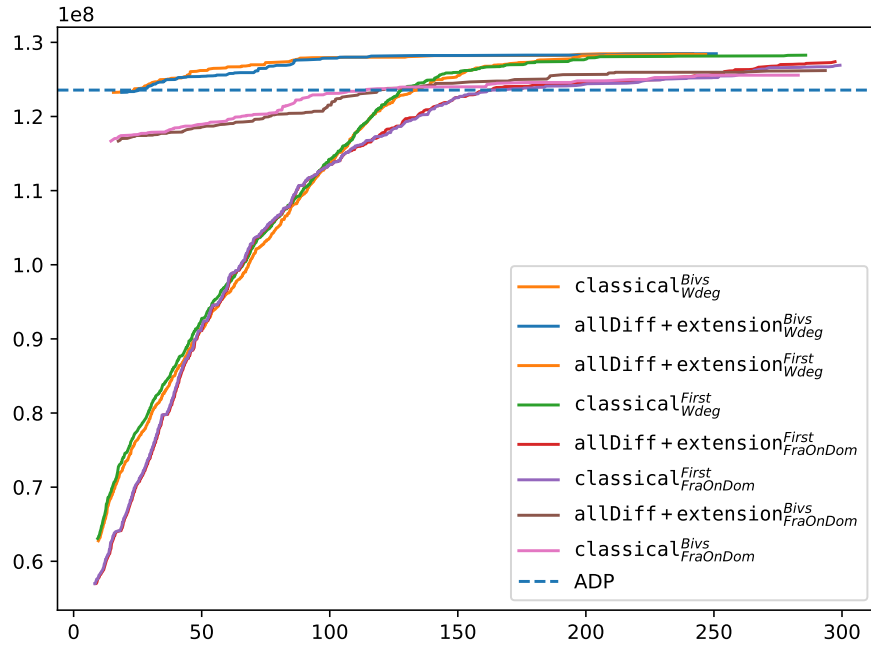


Figure 4.12: CDG T2BD - 2023-09-11 2023-09-17

flights is very small, the bounds can be compared to the prior section's method. Consistent with the previous section's findings, every configuration we tested yielded a bound superior to that determined by ADP. Table 4.12 displays the count of rotations that have been reconstructed. In all instances, our configuration enhances the number of reconstructions.

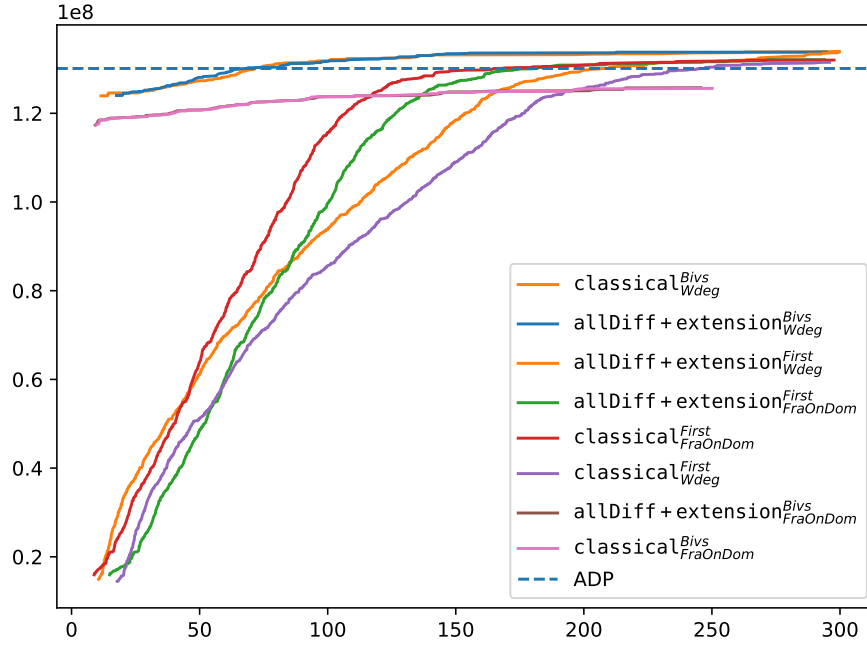


Figure 4.13: CDG T2BD - 2023-09-18 2023-09-24

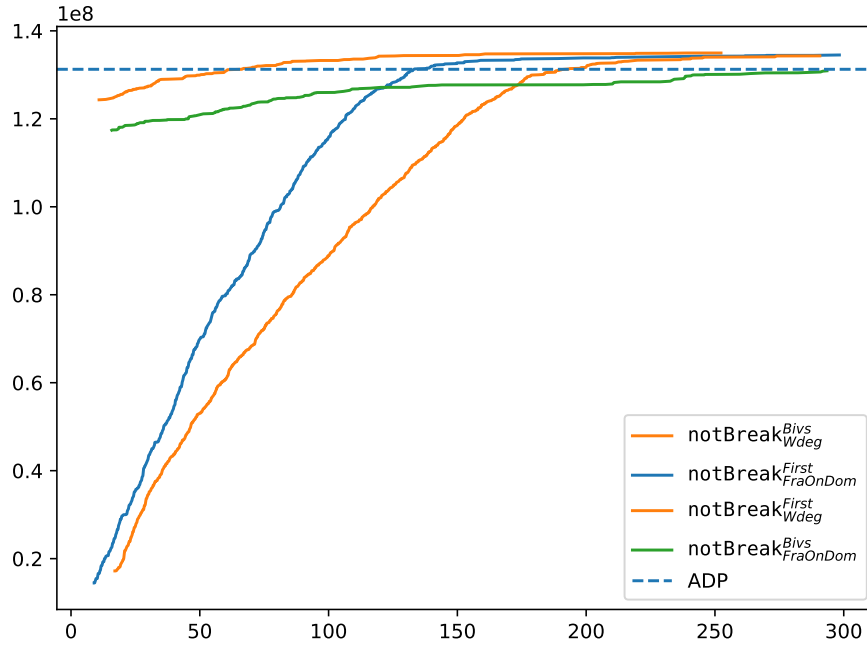


Figure 4.14: CDG T2BD - 2023-07-17 2023-07-23

input	ADP	notBreak ^{First} _{Wdeg}	notBreak ^{Bivs} _{Wdeg}	notBreak ^{Bivs} _{FraOnDom}	notBreak ^{First} _{FraOnDom}
CDG S23 WE37 T2BD	3	16	16	16	11
CDG S23 WE38 T2BD	3	8	8	11	7
CDG S23 WE29 T2BD	5	7	6	7	7

Table 4.12: Number of flights reconstitute by each solver and ADP algorithm.

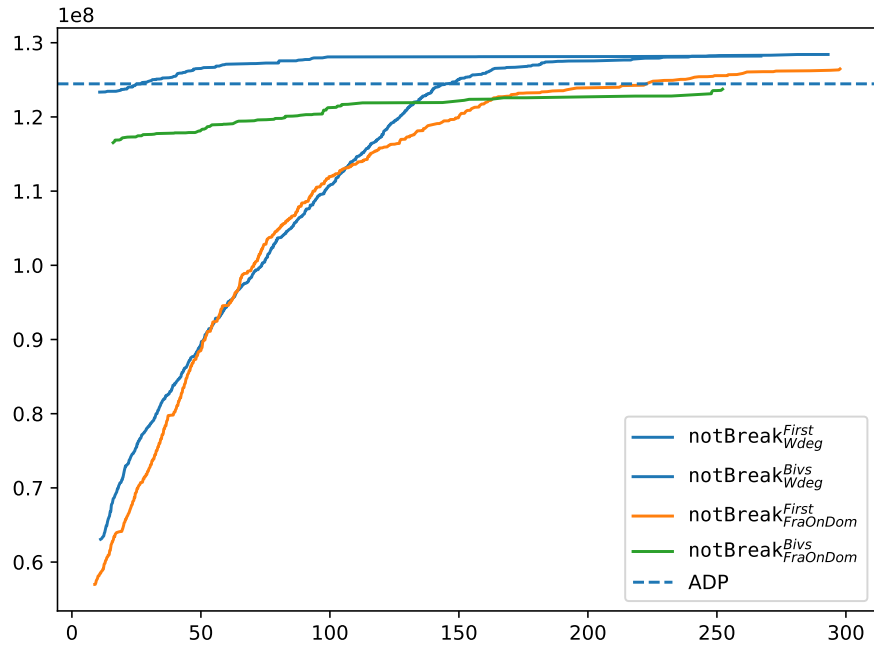


Figure 4.15: CDG T2BD - 2023-09-11 2023-09-17

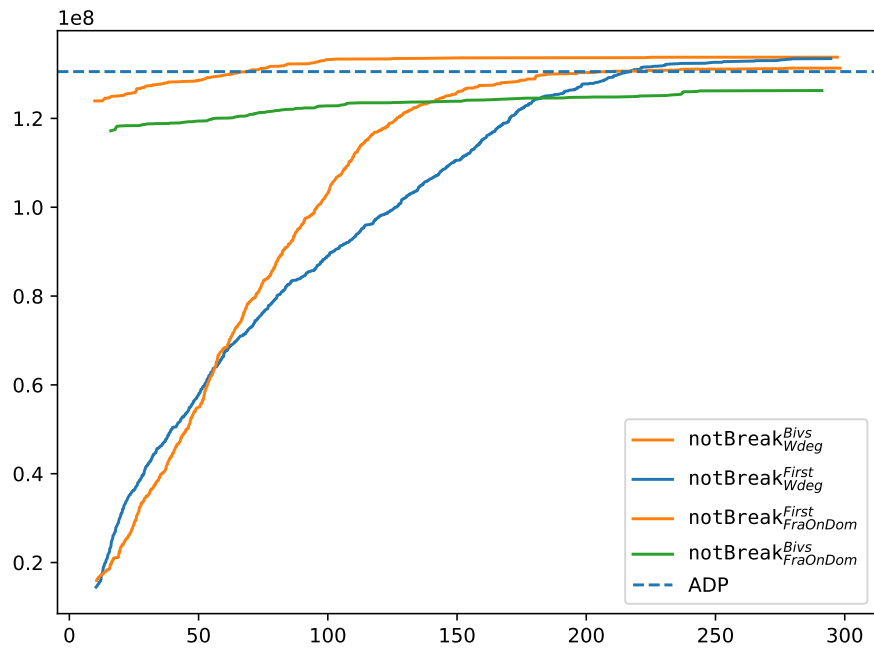


Figure 4.16: CDG T2BD - 2023-09-18 2023-09-24

4.8 Conclusion

In this chapter, various models addressing the stand allocation problem have been introduced. This problem is a fundamental aspect of airport operations that has implications for both efficiency and airline satisfaction. We put forward three distinct formulations of the problem: the **classical** variant which mainly uses extension constraints, **allDifferent** variant which adds **allDifferent** constraints for each maximum clique of the interval graph formed from overlapping tasks, and finally, the **notBreak** variant, which seeks not only to maximize airline satisfaction but also to reduce the number of aircraft movements.

Subsequently, we analyzed different configurations for the **ACE** solver and compared the results with the actual ADP method. The value heuristic order **Bivs** shows excellent performance and always obtains a better bound than that found by ADP.

It is crucial to note that this chapter represents a proof of concept work. While it provides valuable foundational understanding and initial methodologies, there is a pressing need for further optimization.

Chapter 5

Contributions to Resolution Methods of Constraint Programming Problems

Contents

5.1 Aggressive Bound Descent	110
5.1.1 Introduction	110
5.1.2 ABD Policy	111
5.1.3 Simple ABD Implementation	111
5.1.4 Experiments and Results	113
5.1.5 Conclusion	118
5.2 Pseudo-Boolean Encodings	123
5.2.1 Introduction	123
5.2.2 Preliminaries	123
5.2.3 Purely PB Encodings	124
5.2.4 Experiments and Results	127
5.2.5 Conclusion	132
5.3 Parallel solving	132
5.3.1 Related works	133
5.3.2 Architecture of our Framework	134
5.3.3 EPS approaches	136
5.3.4 Experiments and Results	140
5.3.5 Conclusion	141
5.4 Conclusion	144

This chapter unfolds three novel resolution methods, each aiming to bolster the efficiency and adaptability of solving Constraint Programming problems.

The first section introduces a method called *Aggressive Bound Descent* for reducing the number of bounds found by the solver and accelerating the search. The second section explores the use of *Pseudo-Boolean* solvers as the underlying solver. Finally, the last section proposes a new strategy for parallel solving.

5.1 Aggressive Bound Descent

5.1.1 Introduction

When solving a COP instance, the **bound descent** is defined as the sequence $\mathcal{B} = \langle B_1, B_2, \dots \rangle$ of successive bounds identified by the search algorithm. At an extreme, this sequence contains only one value, the optimal bound. At another extreme, it contains a large sequence of values, each one being close to the previous one: the bound descent is said to be **slow**. This is the case when the mean value of the derived sequence of bound gains (or gaps) $G = \langle B_1 - B_2, B_2 - B_3, \dots \rangle$ is small (close to 1).

Indeed, a slow bound descent indicates that there is some room for improvement in the way the backtrack search is conducted. Indeed, enumerating a lot of close solutions before reaching optimality involves solving many derived satisfaction problem instances, always on different, although related, backgrounds (objective constraint limits), and this may be penalizing. This is why we propose an **aggressive**¹⁴ policy of bound descent, ABD (policy) in short [FLMW21, FLMW22]. Instead of setting the strict objective constraint limit to B when a new solution of cost B is found, we propose setting it to a lower value, B' .

A first and simple ABD policy could be to use a static difference between B and B' : $B' = B - \Delta$ where Δ is a fixed positive integer value. However, this **static** policy suffers from a lack of adaptability, and besides, setting the correct value for Δ may be problem-dependent and not very easy to achieve. This is why we propose some **dynamic** ABD policies inspired by studies concerning the sequences used by restart policies.

We first introduce a few general sequences of strictly positive integers to define dynamic ABD policies, i.e., functions $\mathbf{abd} : \mathbb{N}^+ \rightarrow \mathbb{N}^+$. Although detailed later, the parameter $i \geq 1$ of these sequences corresponds to the number of successive successful limit updates, i.e., successive aggressive updates of the objective constraint limit while keeping satisfiability.

Specifically, four integer sequences are used in our study:

$$\mathbf{exp}(i) = 2^{i-1} \quad (5.1)$$

$$\mathbf{rexp}(i) = \begin{cases} 2^{k-1}, & \text{if } i = \frac{k(k+1)}{2} \\ 2^{i - \frac{k(k+1)}{2} - 1}, & \text{if } \frac{k(k+1)}{2} < i < \frac{(k+1)(k+2)}{2} \end{cases} \quad (5.2)$$

$$\mathbf{luby}(i) = \begin{cases} 2^{k-1}, & \text{if } i = 2^k - 1 \\ \mathbf{luby}(i - 2^{k-1} + 1), & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases} \quad (5.3)$$

$$\mathbf{prev}(i) = \begin{cases} 1, & \text{if } i = 1 \\ G_{i-1} \times 2, & \text{else} \end{cases} \quad (5.4)$$

where, in Equation 5.4, G_i is the i th value of the sequence of bound gains, as defined earlier.

Equation 5.1 corresponds to the classical exponential function **exp** (using base 2). Derived from this simple exponential progression, **rexp** in Equation 5.2 corresponds to a regularly reinitialized **exp** sequence. The first values of this sequence are: 1, 1, 2, 1, 2, 4, ... When only considering the highest numbers produced by the first term (condition) of the equation, we obtain a slightly slower progression than the previous one: $O(2^{\sqrt{i}})$. Another sequence commonly used in restart policies is the Luby sequence [LSZ93], given by Equation 5.3. The first values of the Luby sequence are: 1, 1, 2, 1, 1, 2, 4, ... When considering again the highest numbers produced

¹⁴This work was carried out with another CRIL PhD student: Hugues Watez.

by the first term, we can observe that the progression is in $O(i)$. Finally, the last sequence, given by Equation 5.4, is based on the sequence of bound gains and also follows an exponential progression.

Each sequence in $\Psi = \{\text{exp}, \text{rexp}, \text{luby}, \text{prev}\}$ allows us to define an eponymous ABD policy as follows.

5.1.2 ABD Policy

Let $\mathcal{T}$ be the current search tree built by the search algorithm (i.e., during the current run). Let $\mathcal{B}$ be the bound descent produced since the beginning of the current run, and let $\text{abd} \in \Psi$.

The current run can meet three distinct situations:

1. the current run is stopped because the cutoff value is reached,
2. the current run is stopped because the search algorithm indicates that no better solution exists,
3. a new solution S is found.

First, we discuss the most interesting case: the third one. The ABD policy states that when a new solution S of cost B is found, B is appended to $\mathcal{B}$, and the limit of the objective constraint is set to $B + 1 - \text{abd}(i)$, where $i = |\mathcal{B}|$. In other words, the objective constraint becomes: $\text{obj} < B + 1 - \text{abd}(i)$; note that 1 is added to B because abd functions only return values greater than or equal to 1. Now, we give a general precise description (handling in particular the two first situations above) of how an ABD policy can be implemented within a backtrack search.

5.1.3 Simple ABD Implementation

The function `solve`, Algorithm 2, aims at solving the specified CNO $\mathcal{P}$ while using the specified aggressive bound descent policy abd .

Algorithm 2: `solve( $P, \text{abd}$ )`

Output: $\underline{B}_P.. \overline{B}_P$, `runStatus`

```

1  $\underline{B}_P.. \overline{B}_P \leftarrow -\infty.. +\infty$ ;
2 do
3    $P, \text{runStatus} \leftarrow \text{run}(P, \text{abd})$  ;
4 while runStatus = CONTINUE;
5 return  $(\underline{B}_P.. \overline{B}_P, \text{runStatus})$ ;

```

First of all, the lower and upper bounds, denoted by $\underline{B}_P$ and $\overline{B}_P$, of the objective function of P are respectively initialized to $-\infty$ and $+\infty$ (or any relevant values that can be pre-computed). These bounds will be updated during the search (but for the sake of simplicity, this will not be explicitly shown in the pseudo-code). At line 2, the sequence of runs (restarts) is launched. Each time a new run is terminated, it returns the constraint network (possibly updated with some constraints or nogoods that have been learned; this will be discussed in more detail later) and status information. The status takes one of the following values: *CONTINUE* if the solver is allowed to continue with a new run; *COMPLETE* if the last run has exhaustively explored the solution space; *INCOMPLETE* if the solver has reached the timeout limit without entirely exploring the search space. Finally, the function returns the best-found bounds (in case the optimality has been proved, we have $\underline{B}_P = \overline{B}_P$) and the final status of the search.

The function **run**, Algorithm 3, performs a search run, following the restarts and **abd** policies. Before going further, we need to introduce the notion of **safe/unsafe** run solving: when the solver is asked to decrease its objective limit aggressively, we may enter a part of the search space that is UNSAT. If unsatisfiability is proved during the current run, this may be due to our aggressive approach, and consequently, we have to address this issue. This is discussed below.

Algorithm 3: $\text{run}(P, \text{abd})$

Output: (P, status)

```

1  $i \leftarrow 1$ ;
2  $\Sigma_i \leftarrow \emptyset$ ;
3 do
4    $\Delta \leftarrow \text{abd}(i)$ ;
5    $i \leftarrow i + 1$ ;
6    $\Sigma_i, \text{status} \leftarrow \text{search\_next\_sol}(P, \Sigma_{i-1}, \Delta)$ ;
7 while  $\text{status} = \text{SAT}$ ;
8  $\text{safe} \leftarrow \Delta = 1$ ;
9 if  $\text{status} = \text{TIMEOUT}$  then
10  return  $(P, \text{INCOMPLETE})$ 
11 if  $\text{status} = \text{UNSAT} \ \& \ \text{safe}$  then
12  return  $(P, \text{COMPLETE})$ ;
13 if  $\text{status} = \text{UNSAT} \ \& \ \neg \text{safe}$  then
14  return  $(P \oplus \text{nld}(\Sigma_{i-1}), \text{CONTINUE})$ ;
15 if  $\text{status} = \text{CUTOFF\_REACHED} \ \& \ \neg \text{safe}$  then
16  return  $(P \oplus \text{nld}(\Sigma_{i-1}), \text{CONTINUE})$ ;
17 if  $\text{status} = \text{CUTOFF\_REACHED} \ \& \ \text{safe}$  then
18  return  $(P \oplus \text{nld}(\Sigma_i), \text{CONTINUE})$ ;

```

The function starts by initializing a counter i to 1. It corresponds to the number of times we have tried to find a new solution during the current run. At Line 2, Σ_i is the sequence of decisions taken along the rightmost branch of the current run, just before starting the next attempt to find a new solution; this way, we can keep searching from the very same place (in practice, we resume search after it was stopped). Initially, we start from no decisions taken at all (and so, Σ_1 is the empty set). From a practical point of view, as we shall see, only the two last sequences Σ_i and Σ_{i-1} will be helpful to (to deal with the safe and unsafe solving cases). Then, the algorithm iteratively performs runs as long as new solutions can be found. At each loop turn, the next bound gap Δ is computed by soliciting the **abd** policy, i is incremented, and the new upper bound is set (lines 4 and 5). To perform a part of the search, the function **search_next_sol** is called while considering the specified sequence of decisions to start from and the specified bound gap. The gap Δ is used by **search_next_sol** to compute a temporarily upper bound B' which replaces the current upper bound $\overline{B}_P$: we have $B' = \overline{B}_P - \Delta + 1$, forcing then the objective constraint to be $f < B'$ during this call to **search_next_sol**. If a new solution of cost B (necessarily, $B < B'$) is found by **search_next_sol**, the call is stopped, and the objective constraint is updated to become $f < B$ safely. Otherwise, the call is stopped (because the cutoff or timeout limits are reached), and the objective constraint is updated to $f < \overline{B}_P$ (getting back the previous safe upper bound). To summarize, this function implicitly updates the optimization bounds before returning the new sequence of decisions (the exact place where the search has

stopped) and a (local) status. The local status is either SAT, in which case the run can be continued with a new loop iteration, or a value among UNSAT, CUTOFF_REACHED, and TIMEOUT.

The current run necessarily executes some statements starting from line 8. At this line, a Boolean is set, informing us whether the current run was safe or not regarding the last computed Δ value. When the global timeout limit is reached, the constraint network and the status INCOMPLETE are returned (lines 9 and 10). When the local status notifies UNSAT, we have two cases to consider. If the current search was performed in safe mode, COMPLETE can be returned because the search space is guaranteed to have been thoroughly explored. Otherwise, CONTINUE is returned with the constraint network P , possibly integrating some new constraints (nogoods). The notation $P \oplus nld(\Sigma_{i-1})$ indicates that all nogoods that can be extracted from the last but one sequence of decisions (see [LSTV09]) are added to P ; this is valid because this sequence was the one corresponding to the last found solution. When the local status notifies CUTOFF_REACHED, we can also continue while considering the adjunction of some restart nogoods, from either Σ_i or Σ_{i-1} .

We conclude this section with two remarks. Firstly, the algorithm is introduced within the context of a light nogood recording scheme (only nogoods that can be extracted from the rightmost branch, when the search is temporarily stopped, are considered). However, it is possible to adapt it to other learning schemes by keeping track of the exact moment when a nogood (clause) is inferred; this is purely technical. Secondly, there is a specific case concerning unsatisfiability: if ever we encounter a situation where $B' \leq \underline{B}_P$ when trying to set a new temporary upper bound B' during the current run, the sequence is reinitialized by forcing back $i = 1$ and B' is recomputed.

5.1.4 Experiments and Results

Our approach has been evaluated on a wide range of optimization problems coming from the XCSP distribution. To evaluate the performance of our approach, we implemented the ABD policy in the solver ACE and ran it through ACEURANCETOURIX. We executed different variants of ACEURANCETOURIX, denoted ACEX_s^r in the rest of this section, where s is the name of the ABD policy and r the ratio. We define the set of solvers for this campaign by Ψ_5 .

$$\left\{ \text{ACEX}_s^r \mid \begin{array}{l} r \in \{1.2, 1.4, 1.6, 2\} \\ s \in \{\text{exp}, \text{prev}\} \end{array} \right\} \quad (\Psi_5)$$

5.1.4.1 Scoring

When evaluating campaigns, our preference leans towards utilizing absolute methods. These methods assess a solver comprehensively without comparing it with other solvers. Conversely, evaluating based on commonly solved instances among a group of solvers introduces interdependencies. This is termed a relative evaluation. While relative evaluations hold value, they are less desirable since altering the solver set means reevaluating every solver within that set. A leading solver in one set may lose its position if a new solver is introduced, and the top performer is not necessarily the latest addition.

In optimization contexts, evaluations often lean towards these relative methods. Diversifying the relative methods is suggested to derive more meaningful insights from such evaluations. Absolute evaluation methods exist for optimization campaigns, such as tallying the instances where a solver confirms optimality or noting the overall resolution time.

While optimality is vital in evaluating a COP campaign, it does not provide a comprehensive view. Some relative methods gauge how often a solver dominates a campaign, marking instances where it outperforms others. Another approach normalizes each solver's bounds between 0 and 1 at specific intervals, aiming to discern which solvers remain closest to the optimal bound.

We will consider that we only have minimization problems for simplicity's purpose. We propose additions based on the different scoring variables defined in Section 3.4.4 that we recall.

Given a set $\mathcal{I}$ of instances and a set Ψ of solvers,

$$b_{s,t}^i$$

corresponds to the best bound obtained by solver $s \in \Psi$ on instance $i \in \mathcal{I}$ in t second(s), where $t \in [0, \dots, T]$ and T is the timeout. We also have the boolean variable:

$$c_{s,t}^i \tag{5.5}$$

where the value is **true** when the solver s has completed a complete exploration of the search space of the instance i with a time less than t , **false** otherwise.

We can now define two values that correspond to the lowest(best) and to the largest (worst) bounds obtained by a set of solvers Ψ on a given instance i at time t where $t \in [0, \dots, T]$.

$$\text{minb}_t^i = \min_{s \in \Psi} b_{s,t}^i$$

$$\text{maxb}_t^i = \max_{s \in \Psi} b_{s,t}^i$$

We also define the notation for the minimum time taken by a set of solvers Ψ to find the first solution of an instance i .

$$\text{mint}_t^i = \min_{s \in \Psi} t_s^i$$

$$\text{maxt}_t^i = \max_{s \in \Psi} t_s^i$$

These previous equations allow us to normalize the bound $b_{s,t}^i$ based on the **min** – **max** operation.

$$n_{s,t}^i = \begin{cases} 0, & \text{if } b_{s,t}^i = \infty \\ 1, & \text{if } b_{s,t}^i = \text{max}_t^i = \text{min}_t^i \\ \frac{\text{max}_t^i - b_{s,t}^i}{\text{max}_t^i - \text{min}_t^i}, & \text{otherwise} \end{cases} \tag{5.6}$$

For a solver without a solution, its reward is set to 0. Conversely, if both the lowest and highest bounds are identical—indicating the solver has identified the sole recognized solution at a given time t —its reward is 1. The **min** – **max** operator is utilized in other scenarios.

Armed with these defined variables, we can lay out the evaluation operators for a campaign encompassing solvers Ψ applied to a collection of instances $\mathcal{I}$. For a given solver s within Ψ and at a specific time t , the following are the diverse absolute and relative evaluations applicable to the instance set $\mathcal{I}$:

$$\mathcal{E}_{s,t}^{\text{opti}} = \frac{1}{|\mathcal{I}|} \times \sum_{i \in \mathcal{I}} \mathbb{1}c_{s,t}^i \quad (5.7)$$

$$\mathcal{E}_{s,t}^{\text{domi}} = \frac{1}{|\mathcal{I}|} \times \sum i \in \mathcal{I} \lfloor n_{s,t}^i \rfloor \quad (5.8)$$

$$\mathcal{E}_{s,t}^{\text{quali}} = \frac{1}{|\mathcal{I}|} \times \sum_{i \in \mathcal{I}} n_{s,t}^i \quad (5.9)$$

$$(5.10)$$

Where $\mathbb{1}_\alpha$ yields 1 if α is true and 0 if not. The evaluation $\mathcal{E}_{s,t}^{\text{opti}}$ offers an absolute method, representing the frequency the solver s completes its search by timestamp t for the instances within set $\mathcal{I}$. This optimality rate is gauged relative to the total instances. Conversely, $\mathcal{E}_{s,t}^{\text{domi}}$ quantifies the instances where solver s determines the best bound (i.e., when $n_{s,t}^i = 1$) for an instance i at t , all against the backdrop of set $\mathcal{I}$. This metric is then adjusted in relation to the instance count, hence depicting the relative dominance of solver s over set $\mathcal{I}$ at t . Lastly, the $\mathcal{E}_{s,t}^{\text{quali}}$ metric computes the mean caliber of bounds pinpointed across $\mathcal{I}$ by solver s at t .

All the proposed operators are available in the Python library **Metrics**.

5.1.4.2 ABD for XCSP instances

Our experiments utilized benchmarks from the *COP* and *MiniCOP* tracks of the XCSP22 and XCSP23 competitions. Specifically, set $\mathcal{I}_7$ comprises 18 problem families with 296 instances, while set $\mathcal{I}_8$ includes 26 problem families with 405 instances.

$$\mathcal{I}_{\text{XCSP22}}^{\text{COP}} \quad (\mathcal{I}_7)$$

$$\mathcal{I}_{\text{XCSP23}}^{\text{COP}} \quad (\mathcal{I}_8)$$

Summary of the experiments 7 (ABD on XCSP instances - ,)

$\mathcal{I}_7$ and $\mathcal{I}_8$

Ψ_5

40 minutes

64 GB

Linux CentOS Stream 8.3

Two quadcore Intel XEON E5-2637 (3.5 GHz)

128 GB

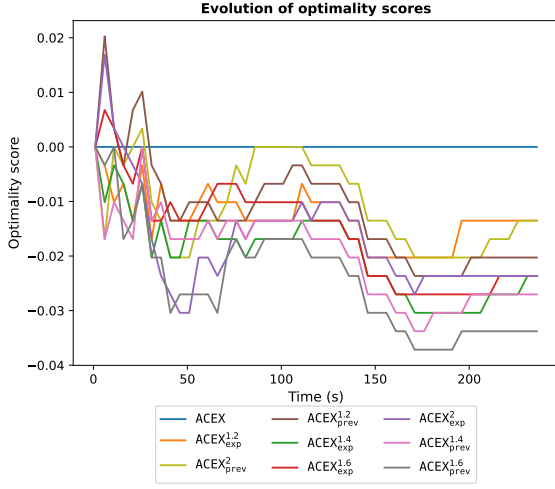
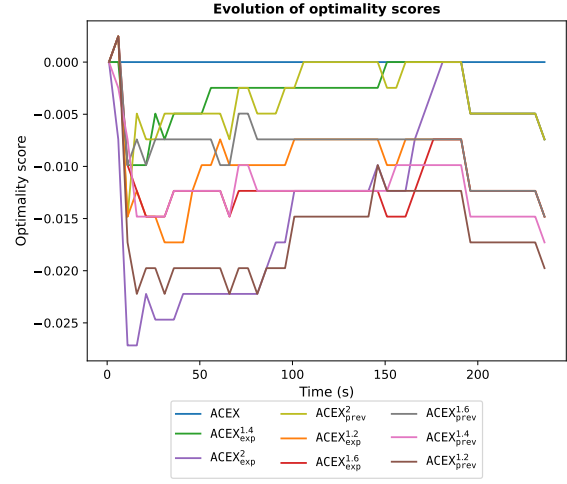
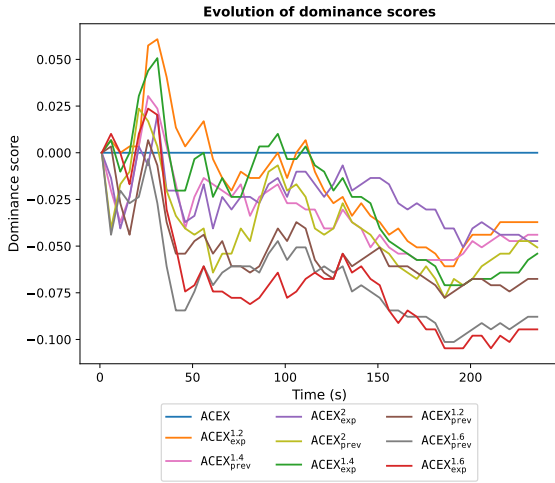
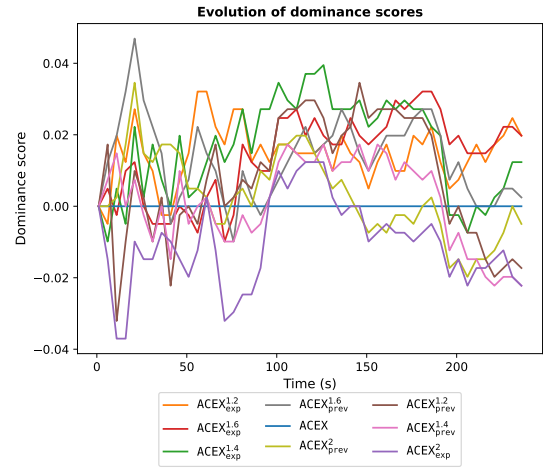
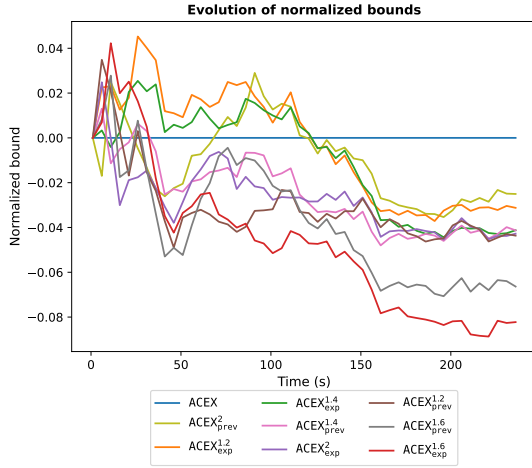

 (a) Difference of *optimality* proportion with default solver ACEX for $\mathcal{I}_7$.

 (b) Difference of *optimality* proportion with default solver ACEX for $\mathcal{I}_8$.

 (c) Difference of *dominance* proportion with default solver ACEX for $\mathcal{I}_7$.

 (d) Difference of *dominance* proportion with default solver ACEX for $\mathcal{I}_8$.

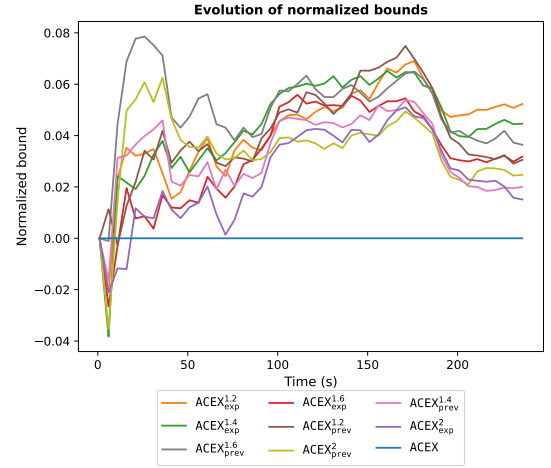
 Figure 5.1: Comparison between the default solver and ABD policy over $\mathcal{I}_7$ and $\mathcal{I}_8$ instances.

Figure 5.1 shows the different proposed operators for comparing the default solver ACEX with different ABD policies. On the left column, there are the results for $\mathcal{I}_7$, while the right column presents the results for $\mathcal{I}_8$.

Figures 5.1a and 5.1b show the optimality score. The optimality score of the ABD policy remains largely consistent. This can be explained by the fact that the ABD policy is aggressive, with bounds that progress slowly. When approaching the optimal boundary, the search requires a finer progression to avoid overshooting the boundary and unnecessarily searching on the UNSAT side of the search space. Figures 5.1a and 5.1b show a fluctuation of just over half a percent, which is positive. Figure 5.1c shows the dominance score for $\mathcal{I}_7$. We can observe a slight improvement in score over the default solver in the first 50 seconds after the solvers with ABD



(e) Difference of *average bound quality* with default solver ACEX for $\mathcal{I}_7$.



(f) Difference of *average bound quality* with default solver ACEX for $\mathcal{I}_8$.

Figure 5.1: Comparaision between the default solver and ABD policy over $\mathcal{I}_7$ and $\mathcal{I}_8$ instances. (cont.)

policies bring some disadvantage but which also remains quite small. However, on instances 5.1d ABD policies seem to be more interesting and provide a gain in terms of total resolution time (see Figure 5.1d). Figures 5.1e and 5.1f confirms the global behaviour.

5.1.4.3 ABD for Airport Problems

We start this section by testing our ABD approach on the check-in desk problem ($\mathcal{I}_1$)

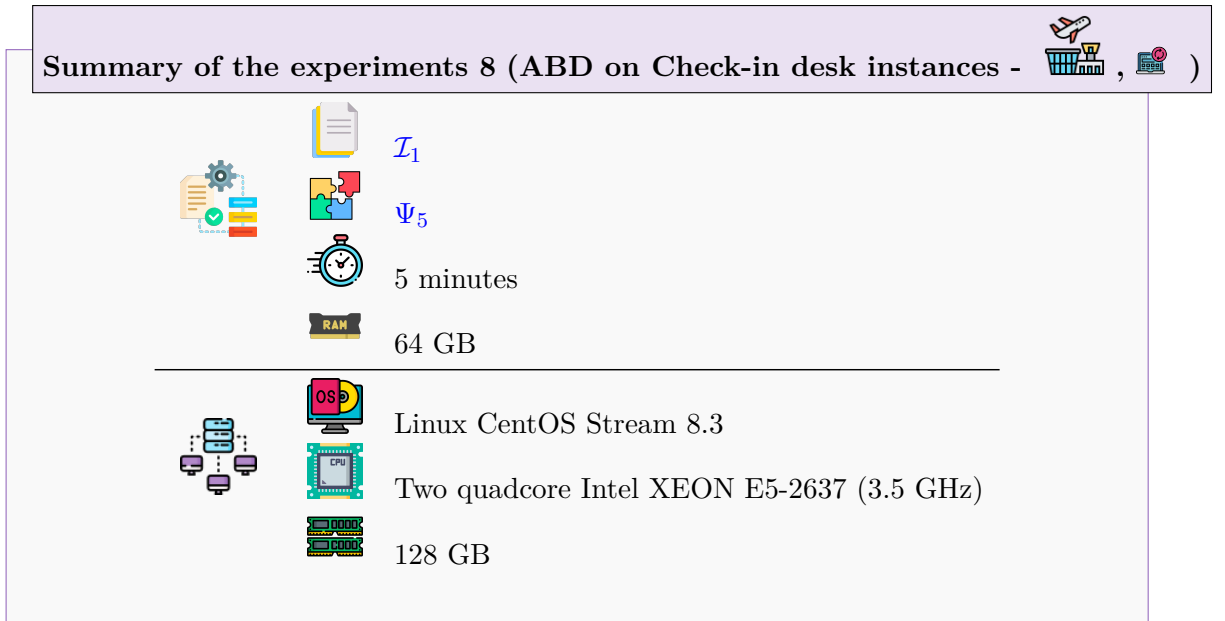


Figure 5.3 provides an overview of the results based on the previous scores. It is important

to highlight that we employed our ABD approach to every instance detailed in $\mathcal{I}_1$, without distinction to decomposition or modeling used. We use a logarithmic scale.

The data presented in Figure 5.2a attests to the stability of the optimality score, indicating that it remains largely unaltered by the ABD. Moreover, Figure 5.2b underscores the dominance of the variant labeled as $\text{Gather}_{First}^{Frba} + \text{prev}^{1.6}$.

Interestingly, the first variant that uses **Bivs** as its value heuristic order ranks in the seventh position. This observation aligns with the insights drawn from Figure 5.2c, which illustrates the average quality of the solutions.

As for the check-in desk, we test the ABD approach on all the stand allocation approaches regardless of this modeling using the set of instances $\mathcal{I}_9$:

$$\left\{ \mathcal{I}_{parking}^f \mid f \in \{\text{classical}, \text{alldiff}, \text{notBreak}\} \right\} \quad (\mathcal{I}_9)$$

Summary of the experiments 9 (ABD on Stand Allocation instances -










$\mathcal{I}_9$

Ψ_5

5 minutes

32 GB






Linux CentOS Stream 8.3

Two quadcore Intel XEON E5-2637 (3.5 GHz)

128 GB

Figure 5.3 illustrates the results of the stand allocation problems.

Initially, it is worth noting that we chose not to display the graph representing the optimality score. The reason behind this decision is the consistent behavior across all solvers where none could close the problem; thus, the graph would merely depict a flat line at $y = 0$.

Moving on to Figure 5.2b, it captures the dominance score associated with each solver. A striking observation is that no particular solver distinctly outperforms the rest. This phenomenon might be attributed to a couple of factors. Firstly, the configurations and solvers in play bear a significant resemblance. Secondly, the quantity of instances considered is relatively modest, which can influence such outcomes.

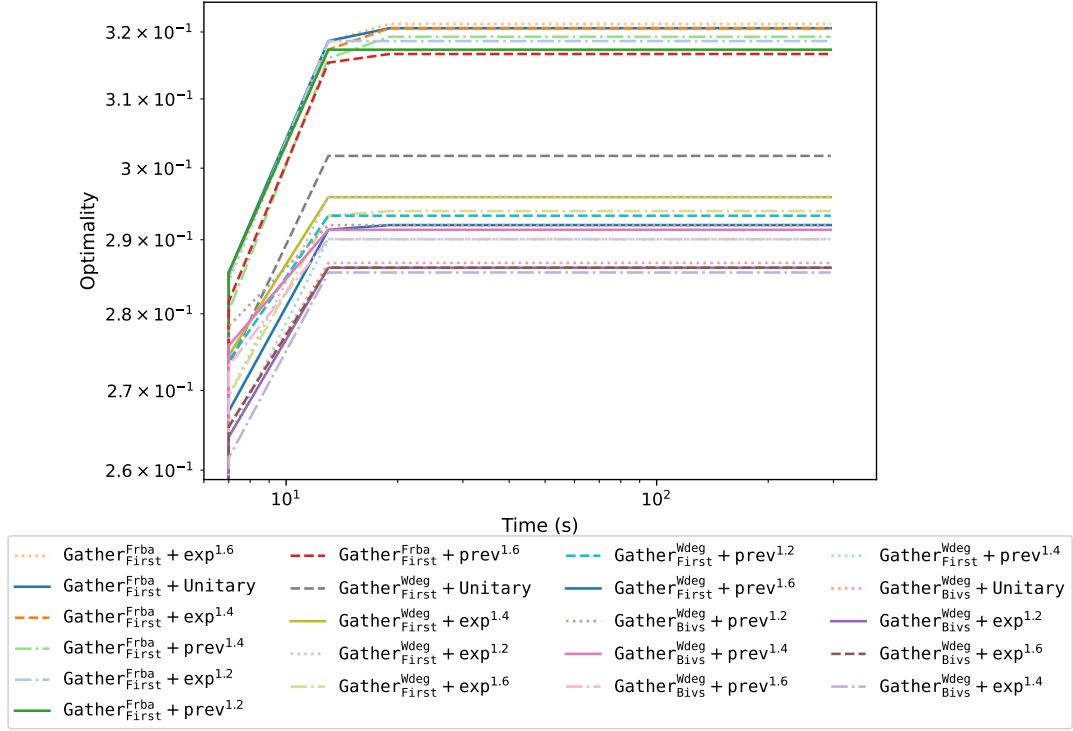
Lastly, Figure 5.2c sheds light on the average quality of the solutions presented. Notably, strategies that employ ABD and the **Bivs** value choice heuristic appear to take the lead. They are closely trailed by a traditional, unitary solver (i.e., without the ABD policy).

5.1.5 Conclusion

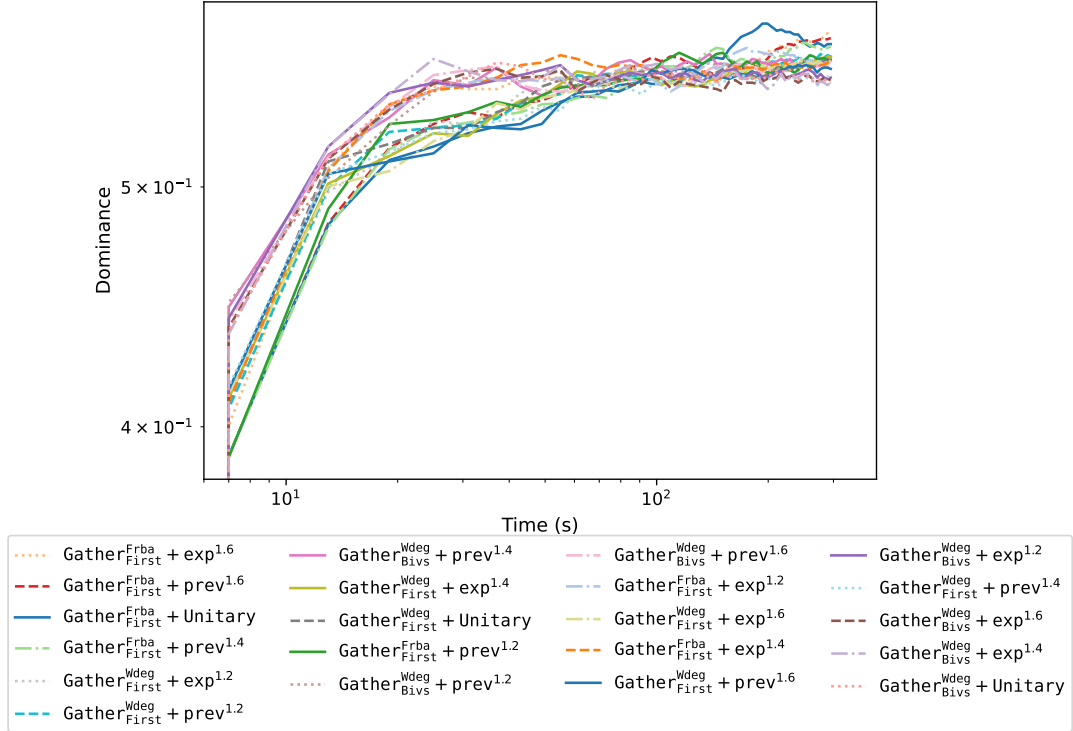
In this section, we introduced the **ABD** technique, a method that aggressively adjusts the bounds of objective constraints. By periodically venturing into potentially unsafe areas of the search

space, this technique can yield notable performance improvements on constraint optimization problems. While the enhancements presented in this section may appear modest, the comprehensive research detailed in [FLMW22] demonstrates a substantial boost in solver efficiency, with improvements exceeding 10%. The previously mentioned research integrated the ABD methodology into the PB **Sat4j** solver. This integration offers a refined **safe** management system, employing selectors to efficiently discard irrelevant information, achieving an impressive performance improvement of up to 20% over the default **Sat4j**.

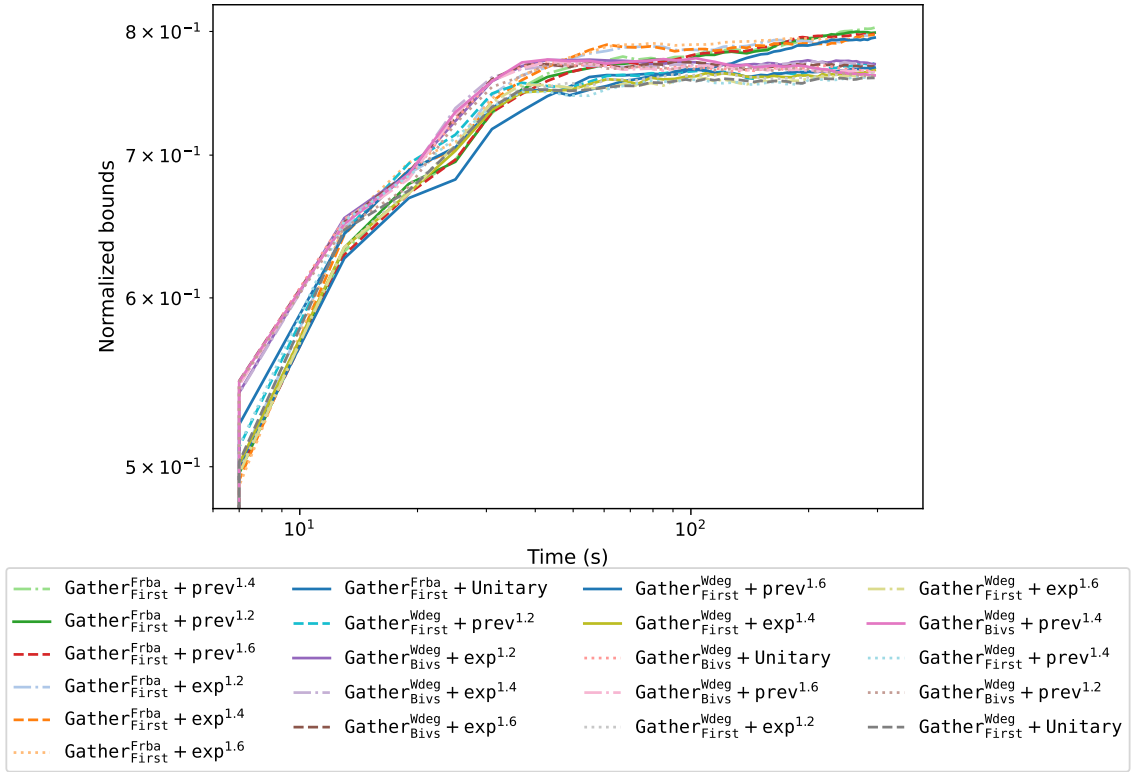
Moving forward, there is potential for further refining ABD. Future endeavors might delve into leveraging historical data on bound improvements or pinpointing optimal sequences of limit deviations based on the inherent structure of specific problems.



(a) Optimality proportion.

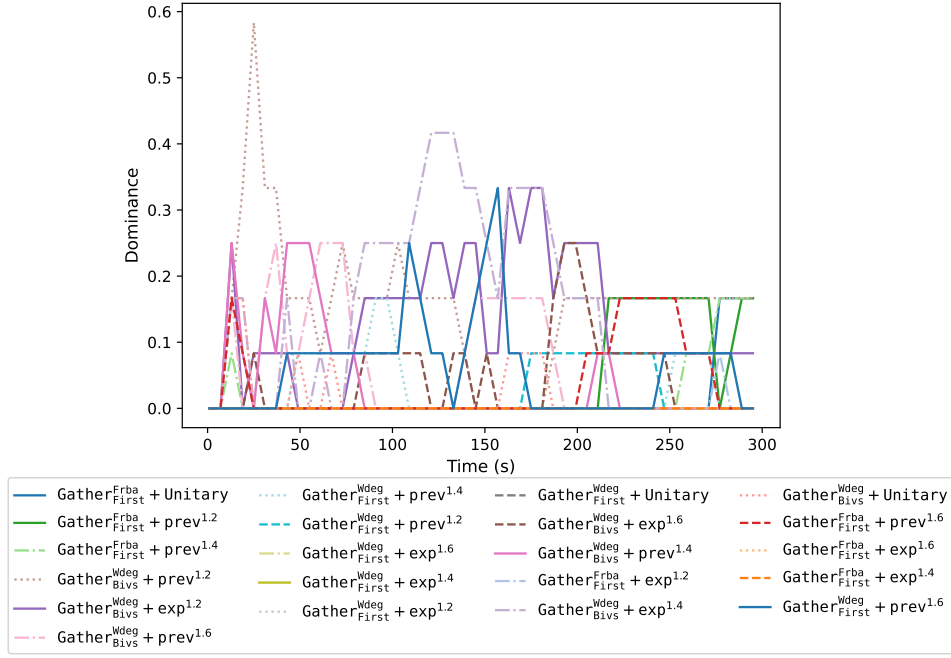


(b) Dominance proportion.

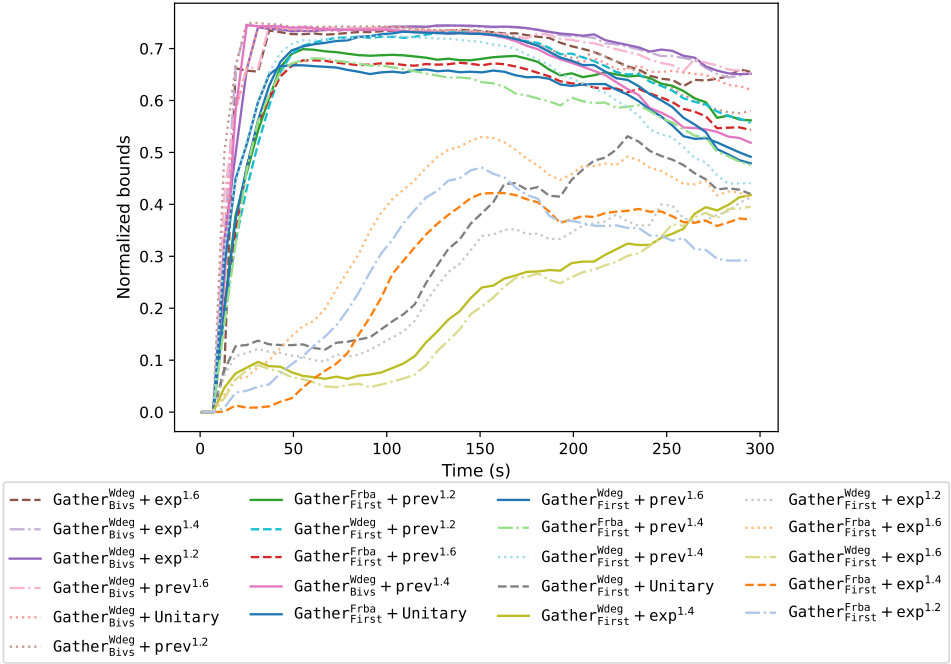


(c) Average solution quality.

Figure 5.2: Comparison between the default solver and ABD policy over $\mathcal{I}_1$ instances.



(a) Dominance proportion.



(b) Average solution quality.

Figure 5.3: Comparaision between the default solver and ABD policy over $\mathcal{I}_9$ instances.

5.2 Pseudo-Boolean Encodings

5.2.1 Introduction

As we have seen in previous chapters, it is possible to solve the CSP (resp. COP) problem with approaches that natively deal with the constraints of the problem, using efficient data structures to represent both the constraints and the structure of the problem, as it is done, for instance, in Choco [PFL16], Nacre [Glo18] and ACE¹⁵. Another possible approach is to leverage the practical efficiency of modern SAT solvers, based on the CDCL architecture [MS99, MMZ⁺01, ES04], to solve CSPs. As these solvers take as input a propositional formula in *Conjunctive Normal Form*, one needs to *encode* the variables (especially their domains) and the constraints of the original problem into clauses to use them. In this context, many encodings have been proposed, as those described in [Wal00, Gav07, DLB08, BKN⁺09, TTKB09], which often provide good empirical results.

However, the main weakness of SAT solvers is the resolution proof system they use during conflict analysis. This proof system has a weak inference power, and some apparently simple problems cannot be solved efficiently. In particular, SAT solvers are known to perform poorly on instances requiring the ability to “count”. This is, for example, the case for the well-known *pigeonhole principle* problem, which requires an exponential number of resolution steps to derive the unsatisfiability of the input [Hak85]. This observation led to the development of a different kind of solvers called *pseudo-Boolean* (PB) solvers [DG02, LBP10, EN18]. These solvers inherit many features of modern SAT solvers and implement a proof system that is stronger than the resolution proof system, and which is known as the *cutting planes* proof system [Gom58, Hoo88, Nor15]. These solvers can natively deal with PB constraints, i.e., linear equations or inequations over Boolean variables. This is an interesting observation, as among existing SAT encodings of CSPs, many of them actually use intermediate PB representations of the constraints before encoding them into clauses. This step is necessary to use a SAT solver, but it requires both to introduce additional variables and to increase the number of constraints to give to the solver (a single PB constraint can represent exponentially many clauses [BSS94]).

In this section, we introduce new encodings for CSPs leveraging both the succinctness of PB constraints and the inference power of PB solvers. The proposed encodings are based on well-known representations of variable domains using Boolean variables, such as the *direct-encoding*, the *log-encoding* or the *order-encoding*, to encode different CSP constraints into PB constraints.

To this end, we first introduce some preliminaries about SAT and PB solving. We then formally describe the proposed encodings for different constraints and empirically evaluate them on different sets of instances from the XCSP library and Paris Airports.

5.2.2 Preliminaries

5.2.2.1 SAT Solvers and CSPs

A variable x is *Boolean* when $\text{dom}(X) = \{0, 1\}$. We call a *literal* ℓ a Boolean variable x or its negation $\bar{x} = 1 - x$. A literal ℓ is *satisfied* when ℓ is assigned to 1, and *falsified* otherwise. A clause is a disjunction of literals, requiring at least one of its literals to be satisfied. A problem is in *Conjunctive Normal Form* (CNF) when it is a conjunction of clauses. The *SATisfiability problem* (SAT) is to determine whether such a conjunction is consistent.

The SAT problem is the first problem proven NP-complete [Coo71]. It is thus possible to use SAT solvers to solve CSPs using different encodings. In particular, to represent the domain

¹⁵<https://github.com/xcsp3team/ace>

of a CSP variable X , one can use the so-called *direct-encoding* (see, e.g., [Wal00]). It defines a Boolean variable x_v for each value $v \in \text{dom}(X)$. In this case, the value assigned to X may be retrieved by identifying the (only) Boolean variable x_v to be satisfied.

This representation is beneficial in the case where the domain of a variable is an enumerated set of values. When it is a range of values, another option is to represent the variable using the *log-encoding* (see, e.g., [Wal00]), which uses the binary representation of the variable X using Boolean variables b_i representing the bits of X , so that $X = \min(\text{dom}(X)) + \sum_{i=0}^{\lceil \log_2(X) \rceil} 2^i b_i$. Observe here the use of $\min(\text{dom}(X))$, which guarantees that the binary decomposition always starts at 0.

Finally, another approach is that based on the *order-encoding* [TTKB09], which defines a Boolean variable $x_{\geq v}$ for each value $v \in \text{dom}(X) \setminus \{\min(\text{dom}(X))\}$. This variable is satisfied if and only if $X \geq v$. In this case, the value assigned to X can be retrieved by identifying two variables $x_{\geq v}$ and $x_{\geq v+1}$ such that the former is satisfied and the latter is falsified, in which case X is assigned to v .

5.2.2.2 Pseudo-Boolean (PB) Constraints

A *pseudo-Boolean* (PB) constraint is a constraint of the form $\sum_{i=1}^n \alpha_i \ell_i \triangle \delta$, where n is a positive integer, the *weights* (or *coefficients*) α_i and the *degree* δ are integers, ℓ_i are literals and $\triangle \in \{<, \leq, =, \geq, >\}$. A PB constraint is said to be *normalized* when all the coefficients and the degree of this constraint are positive, and $\triangle$ is $\geq$. It is well known that any PB constraint may be rewritten as a conjunction of normalized PB constraints, which is particularly useful for the encodings we present later on. A *PB cardinality constraint* is a normalized PB constraint in which all the coefficients are equal to 1, and a *clause* is a PB cardinality constraint with its degree equal to 1. This definition is equivalent to the definition of clauses as disjunctions of literals, and shows that PB solvers generalize SAT solvers.

5.2.3 Purely PB Encodings

In this section, we use the *direct-encoding*, *log-encoding* and *order-encoding* to represent the domains of the variables from a CN and use these representations to encode different common CSP constraints into PB constraints. In the following, we use the notation $(\odot, k)$, where $\odot \in \{<, \leq, =, \neq, \geq, >, \in, \notin\}$ and k is an integer, a variable, a set, or an interval.

5.2.3.1 (De)activating PB constraints

To encode both the domain of the variables and the constraints of a CSP into PB constraints it is often needed to activate (or deactivate) a constraint. To do so, a common practice is to introduce *selectors*, i.e., an auxiliary variable s such that its satisfaction entails that of the considered constraint. In the case of PB constraints, such a selector s could have the following semantics, using $\Rightarrow$ to denote material implication: $s \Rightarrow \sum_{i=1}^n \alpha_i \ell_i \geq \delta$. The particular form of PB constraints allows to concisely represent such an implication with the (single) PB constraint $\delta \bar{s} + \sum_{i=1}^n \alpha_i \ell_i \geq \delta$.

Recall that, in this case, the satisfaction of the constraint does not guarantee the satisfaction of s . If such a guarantee is needed, we must add the reciprocal implication, i.e., $s \Leftarrow \sum_{i=1}^n \alpha_i \ell_i \geq \delta$, which can be represented using a single PB constraint $(\sum_{i=1}^n \alpha_i - \delta + 1) s + \sum_{i=1}^n \alpha_i \bar{\ell}_i \geq \sum_{i=1}^n \alpha_i - \delta + 1$.

From now on, we denote by s a selector for which only one implication is defined, and by $\mathcal{S}$ a selector for which both implications are defined (when needed, indices may be added to these selectors).

To illustrate the use of selectors, let us consider the constraint $\sum_{i=1}^n \alpha_i \ell_i \neq \delta$. This constraint cannot be normalized directly, as $\neq$ is not an allowed operator for a PB constraint. However, we can observe that this constraint is equivalent to the disjunction of the two constraints $(\sum_{i=1}^n \alpha_i \ell_i \leq \delta - 1)$ and $(\sum_{i=1}^n \alpha_i \ell_i \geq \delta + 1)$. Let us define two selectors $s_{\leq}$ and $s_{\geq}$, such that $s_{\leq} \Rightarrow \sum_{i=1}^n \alpha_i \ell_i \leq \delta - 1$ and $s_{\geq} \Rightarrow \sum_{i=1}^n \alpha_i \ell_i \geq \delta + 1$. These two constraints, combined with the disjunction $s_{\leq} \vee s_{\geq}$, allow to represent the constraint above using PB constraints.

5.2.3.2 Variables and Domains

Some constraints must be added to ensure that the encodings of the variables mentioned in the previous section effectively encode the variables of the original problem. In the case of the *direct-encoding*, one simply needs to add the constraint $\sum_{v \in \text{dom}(X)} x_v = 1$ to make sure that the variable X is assigned exactly one value. It is then possible to represent the variable X using the equality $X = \sum_{v \in \text{dom}(X)} v x_v$.

In the case of the *log-encoding*, the value of X is given by the equality $X = \min(\text{dom}(X)) + \sum_{i=0}^{\lceil \log_2(X) \rceil} 2^i b_i$. To make sure that the domain of X is correct, one can use the constraint $\sum_{i=0}^{\lceil \log_2(X) \rceil} 2^i b_i \leq \max(\text{dom}(X)) - \min(\text{dom}(X))$ (recall that we only use this encoding to encode interval domains).

Finally, in the case of the *order-encoding*, the constraints to add are exactly the same clauses as those used in [TTKB09]. Thus, for each value $v \in \text{dom}(X) \setminus \{\min(\text{dom}(X))\}$, the implication $x_{\geq v} \Rightarrow x_{\geq v-1}$ is added to the solver. When the domain of X is not an interval, it is possible to forbid a value v by adding the implication $x_{\geq v} \Rightarrow x_{\geq v+1}$. Additionally, it is possible to represent the value of X using the equality $X = \min(\text{dom}(X)) + \sum_{v \in \text{dom}(X) \setminus \{\min(\text{dom}(X))\}} x_{\geq v}$. As for the *log-encoding*, let us remark the use of $\min(\text{dom}(X))$ to make sure that the encoding starts at 0. Without loss of generality, we can thus always represent a CSP variable X using a weighted sum of literals, to which a constant μ may be added, giving $X = \mu + \sum_{i=1}^n \alpha_i \ell_i$.

Additionally, encoding CSP constraints often requires to determine whether a variable X is assigned to a given value v . In the case of the *direct-encoding*, one only needs to check the value of x_v . To obtain such a value with the *order-encoding*, note first that X is assigned to v if and only if the conjunction $x_{\geq v} \wedge \overline{x_{\geq v+1}}$ is satisfied. To obtain a variable x_v equivalent to that used in the *direct-encoding*, one just needs to use the constraint $x_v \Leftrightarrow x_{\geq v} \wedge \overline{x_{\geq v+1}}$, which can be encoded using PB constraints, as shown in the previous section. Obtaining such a value is a little bit harder with the *log-encoding*, as one needs to check the assignment of *all* the Boolean variables used in the representation of the variable to know the value of X . In this case, we thus propose to use a lazy form of the *direct-encoding*, where X is first encoded using the *log-encoding*, and the *direct-encoding* is used only when x_v is required. The constraint $\sum_{v \in \text{dom}(X)} v x_v = \min(\text{dom}(X)) + \sum_{i=0}^{\lceil \log_2(X) \rceil} 2^i b_i$ may be used to make sure that both encodings encode the same value. It is thus possible to obtain, for each variable X and for each value $v \in \text{dom}(X)$, a unique Boolean variable x_v representing the assignment $X = v$.

5.2.3.3 Constraint cardinality

As mentioned before, one of the main advantages of PB solvers compared to SAT solvers is their ability to count efficiently. To benefit from this advantage, we first propose to encode **cardinality** constraints [Rég96].

A first variant of this constraint is to force bounds on the number of variables among $\{X_1, \dots, X_N\}$ that are assigned a given value v . Let m and M be the minimum and maximum number of variables X_i that can be assigned to v , respectively, and let x_v^i be the Boolean variable representing the assignment $X_i = v$. Clearly, the number of satisfied x_v^i must be between m and M . Thus, this **cardinality** constraint can be represented with $m \leq \sum_{i=1}^N x_v^i \leq M$, which can easily be decomposed into two PB constraints. Let us remark that, depending on the values of m and M , these two constraints may represent an exponential number of clauses without requiring the use of auxiliary variables [BSS94], as SAT solvers would.

Another variant of the **cardinality** constraint is to make sure that a variable C is assigned to the number of variables X_i that are assigned to a given value v . Let $\mu + \sum_{i=1}^n \alpha_i \ell_i$ be the representation of the variable C . This variable must be equal to the number of satisfied x_v^i . So, if we use the same notations as before, this can be encoded using the equality $\sum_{i=1}^N x_v^i = \mu + \sum_{i=1}^n \alpha_i \ell_i$, which is equivalent to the PB constraint $\sum_{i=1}^N x_v^i - \mu - \sum_{i=1}^n \alpha_i \ell_i = 0$.

There exist other variants of the constraints described here, where the values are variables Z instead of constants. To encode them, one simply needs to replace the variables x_v^i by variables x_Z^i , such that $x_Z^i \Leftrightarrow (X_i - Z = 0)$. By representing X_i and Z into weighted sums of literals, this constraint may be normalized following the procedure described in Section 5.2.3.1.

5.2.3.4 Constraint count

Let us now consider the constraint **count**, introduced in CHIP [BC94] and Sicstus [COC97], which ensures that the number of variables in $\{X_1, \dots, X_N\}$ which are assigned a value in $\{v_1, \dots, v_M\}$ respects a numerical condition $(\odot, k)$.

To encode this constraint let $x_{v_j}^i$ be the Boolean variable representing the assignment $X_i = v_j$. Clearly, the number of satisfied $x_{v_j}^i$ must satisfy the condition $(\odot, k)$. Thus, the **count** constraint can be represented with $\sum_{i=1}^N \sum_{j=1}^M x_{v_j}^i \odot k$, which can easily be represented as a set of PB constraints.

5.2.3.5 Constraint nValues

The constraint **nValues** [BHH⁺06] ensures that the number of distinct values taken by variables in $\{X_1, \dots, X_N\}$ respects a numerical condition $(\odot, k)$.

To encode this constraint let $x_{v_j}^i$ be the Boolean variable representing the assignment $X_i = v_j$ where $v_j \in \bigcup_{i=1}^N \text{dom}(X_i)$. We start by creating a clause for each value v_j of the form: $\sum_{i=1}^N x_{v_j}^i \geq 1$. For each of these clauses, we define a selector S_j , which is thus true if and only if the value v_j is assigned to one of the X_i . The **nValues** constraint can then be represented with $\sum_j S_j \odot k$.

5.2.3.6 Constraint sum

A **sum** constraint is a constraint of the form $\sum_{i=1}^N A_i X_i \odot k$. It is clear that such a constraint can easily be represented using PB constraints. Indeed, if each X_i may be written as $X_i = \mu^i + \sum_{j=1}^{n_i} \alpha_j^i x_j^i$, where x_j^i are Boolean variables, then the constraint above is equivalent to $\sum_{i=1}^N A_i \left(\mu^i + \sum_{j=1}^{n_i} \alpha_j^i x_j^i \right) \odot k$, which can be developed as a PB constraint (the case of the operator $\neq$ is treated as in Section 5.2.3.1).

The **sum** constraint is at the core of many other constraints. For example, we can easily encode the **binPacking**, **cumulative**, **knapsack**, and **noOverlap** constraints by using **sum**

constraints, and thus PB constraints based on the encoding presented above.

5.2.3.7 Constraint `allDifferent`

Recall that the global constraint `allDifferent` enforces that a set of variables are all assigned to different values. We illustrate its encoding following the example `allDifferent`($X_1, \dots, X_n$).

Let $\mathcal{D} = \bigcup_{i=1}^N \text{dom}(X_i)$. The semantics of `allDifferent` enforces that each value $v \in \mathcal{D}$ is used at most once among all variables X_i . Let $v \in \mathcal{D}$, and let us note x_v^i the Boolean variable representing $X_i = v$ ($i \in 1..N$). We can represent this constraint on v using the PB constraint $\sum_{i=1}^N x_v^i \leq 1$. This constraint has to be applied on all values $v \in \mathcal{D}$. The operation must thus be repeated on each possible value.

Let us observe that this encoding is similar to that of the *pigeonhole-principle*, which is hard for SAT solvers based on resolution [Hak85]. This encoding allows thus to benefit from the reasoning power of PB solvers, at least on the subset of constraints encoding `allDifferent`.

5.2.3.8 Constraint extension with Supports

A *support* is a constraint that explicitly lists all the possible solutions for a constraint. To encode supports, let us first remark that a PB constraint of the form $\sum_{i=1}^n \ell_i \geq n$ represents the conjunction of the literals ℓ_i , i.e., $\bigwedge_{i=1}^n \ell_i$. Based on this observation, we can encode a (unique) support of the form $(X_i \mid 1 \leq i \leq N) = (v_i \mid 1 \leq i \leq N)$ using the PB constraint $\sum_{i=1}^N x_{v_i}^i \geq N$, where $x_{v_i}^i$ is the Boolean variable representing the assignment $X_i = v_i$ ($i \in 1..N$). When several tuples t are allowed in a support, we need to add a selector s_t to each constraint associated to a tuple, giving the PB constraint $s_t \Rightarrow \sum_{i=1}^N x_{v_i}^i \geq N$. Indeed, the set of allowed tuples is a disjunction, so one of the tuples must be assigned to the associated variables: we thus add the clause $\bigvee s_t$ to encode the full support.

Finally, let us remark that if the symbol $*$ is used in one of the allowed tuples instead of a value v_i (to represent that the variable X_i can take any value), one just needs to ignore the literal corresponding to the variable X_i in the constraint above.

5.2.4 Experiments and Results

5.2.4.1 PB encoding for XCSP instances

This section presents some experimental results of the PB encodings presented in the previous section on different sets of optimization instances. To evaluate the performance of our approach, we implemented the encodings presented in the previous sections, and we executed different variants of `Sat4j` [LBP10], denoted `Sat4j + S` in the rest of this section, where $\mathcal{S}$ is the name of the considered variant. The set of solver is denoted Ψ_6 . Unless otherwise specified, the combination of the *direct-encoding* and *log-encoding* is used to represent the domain of the variables.

$$\{\text{Sat4j} + \mathcal{S} \mid \mathcal{S} \in \{\text{Res}, \text{SoberBoth}, \text{Both}, \text{CuttingPlanes}, \text{RS}, \text{PartialRS}\}\} \quad (\Psi_6)$$

In the context of our experiments, we used the union of $\mathcal{I}_7$ and $\mathcal{I}_8$. This set, denoted $\mathcal{I}_{10}$, comprises 49 families of problems and contains 757 instances.

$$\mathcal{I}_{\text{XCSP22}, \text{XCSP23}}^{\text{COP}} \quad (\mathcal{I}_{10})$$

Summary of the experiments 10 (COP - ,)

$\mathcal{I}_{10}$
 Ψ_6
40 minutes
64 GB

Linux CentOS Stream 8.3
Two quadcore Intel XEON E5-2637 (3.5 GHz)
128 GB

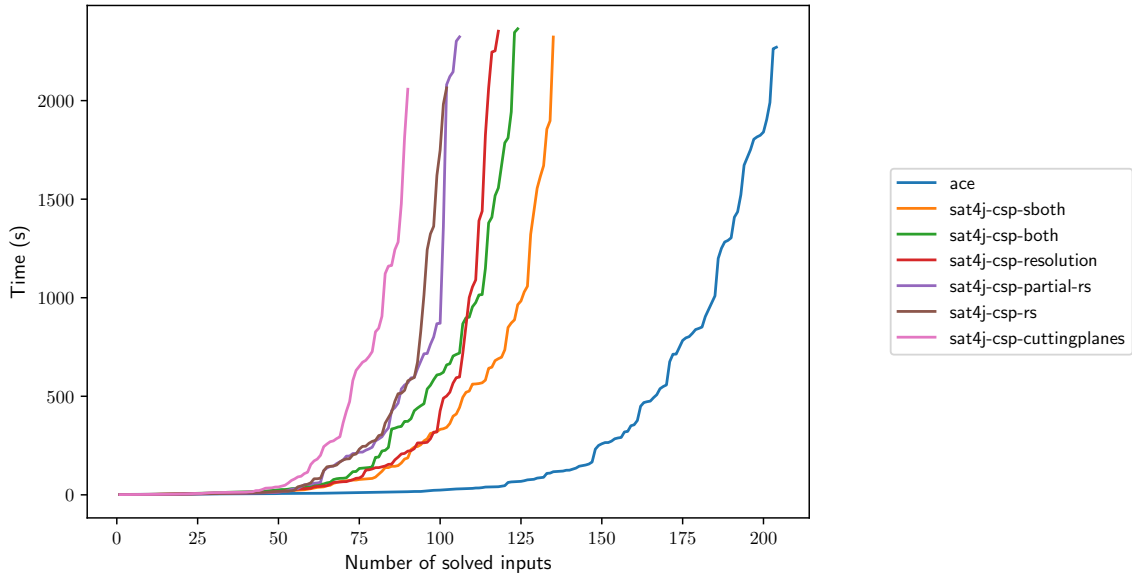


Figure 5.4: *Cactus plot* of the executed solvers.

Figure 5.4 shows an overview of the solvers we ran on this benchmark. This figure is a so-called *cactus-plot*. Each line corresponds to a solver and shows the number of instances solved (in this context, the instance is considered solved if the solver has performed a context search, i.e., proved optimality or unsatisfiability) within a given time limit by this solver.

The faster solver is **ACE**, which solves many more instances than the others. The first PB solver is **Sat4j+SoberBoth**, which combines two solvers **Sat4j+CuttingPlanes** and **Sat4j+Res** during 60 seconds and then only **Sat4j+Res**. This latter PB solver implements a conflict analysis as classical SAT solvers do, by lazily inferring a clause each time a PB constraint is encountered during the analysis. The main advantage of this approach is that it allows to combine the succinctness of PB constraints and the use of efficient data structures of SAT solvers. Yet, the

efficiency of these solvers is not very good. This may be explained by the fact that (too) many constraints have to be encoded using clauses in the considered problems, which does not allow to exploit the full power of the proof system implemented in PB solvers.

5.2.4.2 PB encoding for Airport Problems

In this section, we have tested the pseudo-Boolean encoding on check-in instances. We have run the same set of solvers as in the previous section (Ψ_6).

Summary of the experiments 11 (Sat4j for CDAP - ,)

$\mathcal{I}_2, \mathcal{I}_3$ and $\mathcal{I}_4$
 Ψ_6
5 minutes
32 GB

Linux CentOS Stream 8.3
Two quadcore Intel XEON E5-2637 (3.5 GHz)
128 GB

Term	Period	Decompo	Solver	First B (Time (s))	Last B (Time (s))
CDG T1	0703-0709	Part	Sat4j + both	17,665,000 (13.15s)	20,545,000 (296.53s)
CDG T1	0703-0709	Part	Sat4j + res	15,900,000 (4.9s)	17,950,000 (291.36s)
CDG T1	0703-0709	None	Sat4j + res	14,845,000 (49.99s)	18,110,000 (181.81s)
CDG T1	0703-0709	None	Sat4j + both	14,845,000 (130.63s)	15,955,000 (282.44s)
CDG T1,2,3	0821-0827	Full	Sat4j + res	17,540,500 (6.49s)	18,367,200 (268.77s)
CDG T1,2,3	0821-0827	Full	Sat4j + cp	16,958,900 (15.31s)	19,644,100 (295.93s)
CDG T1,2,3	0821-0827	Part	Sat4j + res	14,928,200 (14.96s)	15,822,700 (18.6s)
CDG T1,2,3	0821-0827	Part	Sat4j + both	14,758,000 (26.08s)	15,822,700 (51.67s)
CDG T1,2,3	0821-0827	None	Sat4j + res	12,357,200 (155.08s)	13,135,500 (235.07s)
CDG T1,2,3	0821-0827	None	Sat4j + both	TO	TO
CDG T1,2,3	0911-0917	Full	Sat4j + cp	TO	TO
CDG T1,2,3	0911-0917	Full	Sat4j + res	TO	TO
CDG T1,2,3	0911-0917	Part	Sat4j + both	TO	TO
CDG T1,2,3	0911-0917	Part	Sat4j + res	TO	TO
CDG T1,2,3	0911-0917	None	Sat4j + both	TO	TO
CDG T1,2,3	0911-0917	None	Sat4j + res	TO	TO
CDG T1,2,3	0918-0924	Full	Sat4j + cp	TO	TO
CDG T1,2,3	0918-0924	Full	Sat4j + res	TO	TO
CDG T1,2,3	0918-0924	Part	Sat4j + both	TO	TO
CDG T1,2,3	0918-0924	Part	Sat4j + res	TO	TO
CDG T1,2,3	0918-0924	None	Sat4j + both	TO	TO
CDG T1,2,3	0918-0924	None	Sat4j + res	TO	TO

Continued on next page

Term	Period	Decompo	Solver	First B (Time (s))	Last B (Time (s))
CDG T2B T2D	0703-0709	Full	Sat4j + res	15,654,200 (6.26s)	18,839,027 (284.8s)
CDG T2B T2D	0703-0709	Full	Sat4j + cp	13,844,236 (11.86s)	16,534,242 (256.21s)
CDG T2B T2D	0703-0709	Part	Sat4j + both	13,799,362 (15.98s)	14,549,245 (33.1s)
CDG T2B T2D	0703-0709	Part	Sat4j + res	13,614,342 (8.6s)	14,659,279 (208.1s)
CDG T2B T2D	0703-0709	None	Sat4j + res	11,274,440 (63.47s)	11,274,440 (63.47s)
CDG T2B T2D	0703-0709	None	Sat4j + both	11,274,440 (158.46s)	11,274,440 (158.46s)
ORY	0508-0514	Full	Sat4j + res	25,948,700 (17.96s)	26,178,500 (25.5s)
ORY	0508-0514	Part	Sat4j + res	24,205,300 (43.28s)	26,240,500 (56.79s)
ORY	0508-0514	Part	Sat4j + both	22,391,000 (70.34s)	26,240,500 (100.6s)
ORY	0508-0514	Full	Sat4j + cp	21,573,000 (48.69s)	24,135,800 (259.37s)
ORY	0508-0514	None	Sat4j + both	TO	TO
ORY	0508-0514	None	Sat4j + res	TO	TO
ORY	0619-0625	Part	Sat4j + res	19,815,150 (34.52s)	19,659,309 (46.74s)
ORY	0619-0625	Part	Sat4j + both	19,815,150 (69.36s)	19,659,309 (98.42s)
ORY	0619-0625	Full	Sat4j + res	17,425,699 (24.8s)	17,522,687 (34.19s)
ORY	0619-0625	Full	Sat4j + cp	TO	TO
ORY	0619-0625	None	Sat4j + both	TO	TO
ORY	0619-0625	None	Sat4j + res	TO	TO
ORY	0626-0702	Part	Sat4j + res	18,080,849 (29.63s)	20,210,892 (41.81s)
ORY	0626-0702	Full	Sat4j + res	18,027,618 (26.94s)	17,295,989 (30.51s)
ORY	0626-0702	Part	Sat4j + both	16,243,025 (54.64s)	20,245,569 (109.4s)
ORY	0626-0702	Full	Sat4j + cp	TO	TO
ORY	0626-0702	None	Sat4j + both	TO	TO
ORY	0626-0702	None	Sat4j + res	TO	TO
ORY	0703-0709	Part	Sat4j + both	20,073,634 (74.7s)	20,770,740 (129.96s)
ORY	0703-0709	Full	Sat4j + res	18,538,057 (26.09s)	19,272,751 (36.73s)
ORY	0703-0709	Part	Sat4j + res	18,104,682 (30.06s)	20,913,738 (46.03s)
ORY	0703-0709	Full	Sat4j + cp	TO	TO
ORY	0703-0709	None	Sat4j + both	TO	TO
ORY	0703-0709	None	Sat4j + res	TO	TO

Table 5.1: Results for No-overlapping family.

Term	Period	Decompo	Solver	First B (Time (s))	Last B (Time (s))
CDG T1	0703-0709	Part	Sat4j + res	22,530,000 (7.26s)	24,310,000 (123.26s)
CDG T1	0703-0709	Part	Sat4j + both	22,155,000 (10.53s)	27,555,000 (132.02s)
CDG T1	0703-0709	None	Sat4j + res	19,800,000 (66.13s)	21,415,000 (124.51s)
CDG T1	0703-0709	None	Sat4j + both	19,800,000 (135.04s)	21,415,000 (263.17s)
CDG T1,2,3	0821-0827	Full	Sat4j + res	24,885,400 (5.76s)	27,301,500 (270.95s)
CDG T1,2,3	0821-0827	Full	Sat4j + both	24,067,400 (7.42s)	28,278,800 (275.9s)
CDG T1,2,3	0821-0827	Part	Sat4j + res	21,766,200 (12.05s)	22,356,700 (19.5s)
CDG T1,2,3	0821-0827	Part	Sat4j + both	20,819,600 (18.72s)	22,714,700 (57.63s)
CDG T1,2,3	0821-0827	None	Sat4j + res	20,440,800 (150.01s)	21,512,100 (274.46s)
CDG T1,2,3	0821-0827	None	Sat4j + both	TO	TO
CDG T1,2,3	0911-0917	Full	Sat4j + res	88,747,600 (251.85s)	88,747,600 (251.85s)
CDG T1,2,3	0911-0917	Full	Sat4j + both	TO	TO
CDG T1,2,3	0911-0917	None	Sat4j + both	TO	TO
CDG T1,2,3	0911-0917	None	Sat4j + res	TO	TO
CDG T1,2,3	0918-0924	Full	Sat4j + res	88,164,200 (213.62s)	88,184,200 (274.59s)
CDG T1,2,3	0918-0924	Part	Sat4j + res	TO	TO
CDG T1,2,3	0918-0924	Full	Sat4j + both	TO	TO
CDG T1,2,3	0918-0924	Part	Sat4j + both	TO	TO
CDG T1,2,3	0918-0924	None	Sat4j + both	TO	TO
CDG T1,2,3	0918-0924	None	Sat4j + res	TO	TO
ORY	0508-0514	Full	Sat4j + res	30,469,800 (18.24s)	31,128,500 (25.06s)
ORY	0508-0514	Full	Sat4j + both	28,762,000 (32.29s)	31,128,500 (55.78s)

Continued on next page

Term	Period	Decompo	Solver	First B (Time (s))	Last B (Time (s))
ORY	0508-0514	Part	Sat4j + res	28,516,800 (47.17s)	28,166,800 (60.95s)
ORY	0508-0514	Part	Sat4j + both	27,524,300 (94.46s)	28,576,600 (118.02s)
ORY	0508-0514	None	Sat4j + both	TO	TO
ORY	0508-0514	None	Sat4j + res	TO	TO
ORY	0619-0625	Part	Sat4j + res	21,762,419 (32.02s)	22,513,890 (41.52s)
ORY	0619-0625	Part	Sat4j + both	20,493,563 (53.51s)	22,544,719 (78.09s)
ORY	0619-0625	Full	Sat4j + res	18,412,520 (24.05s)	19,254,921 (28.53s)
ORY	0619-0625	Full	Sat4j + both	17,708,375 (40.08s)	19,631,810 (250.14s)
ORY	0619-0625	None	Sat4j + both	TO	TO
ORY	0619-0625	None	Sat4j + res	TO	TO
ORY	0626-0702	Part	Sat4j + res	22,063,146 (31.05s)	23,981,390 (48.52s)
ORY	0626-0702	Part	Sat4j + both	21,903,089 (69.17s)	23,900,583 (93.0s)
ORY	0626-0702	Full	Sat4j + both	19,769,761 (49.8s)	20,352,845 (256.61s)
ORY	0626-0702	Full	Sat4j + res	19,691,244 (22.68s)	19,853,765 (30.14s)
ORY	0626-0702	None	Sat4j + both	TO	TO
ORY	0626-0702	None	Sat4j + res	TO	TO
ORY	0703-0709	Part	Sat4j + res	23,081,408 (36.04s)	23,929,208 (43.05s)
ORY	0703-0709	Part	Sat4j + both	21,083,493 (58.44s)	24,345,638 (84.75s)
ORY	0703-0709	Full	Sat4j + both	20,059,776 (48.28s)	21,187,511 (233.13s)
ORY	0703-0709	Full	Sat4j + res	19,621,684 (22.23s)	20,990,780 (30.21s)
ORY	0703-0709	None	Sat4j + both	TO	TO
ORY	0703-0709	None	Sat4j + res	TO	TO

Table 5.2: Results for overlapping family.

Term	Period	Decompo	Solver	Model	First B (Time (s))	Last B (Time (s))
CDG T1	0703-0709	Part	Sat4j + res	minod-count	18,740,000 (10.41s)	19,535,000 (296.1s)
CDG T1	0703-0709	Part	Sat4j + res	minod	18,415,000 (9.36s)	19,535,000 (294.96s)
CDG T1	0703-0709	None	Sat4j + res	minod-count	14,620,000 (122.24s)	14,880,000 (297.16s)
CDG T1	0703-0709	None	Sat4j + res	minod	14,620,000 (175.25s)	14,700,000 (294.23s)
ORY	0508-0514	Full	Sat4j + both	minod-count	27,881,200 (123.37s)	28,091,500 (215.7s)
ORY	0508-0514	Full	Sat4j + res	minod	27,536,600 (60.66s)	27,804,400 (103.52s)
ORY	0508-0514	Part	Sat4j + res	minod-count	26,606,900 (106.46s)	28,609,500 (215.75s)
ORY	0508-0514	Full	Sat4j + res	minod-count	26,345,000 (53.9s)	27,881,200 (80.76s)
ORY	0508-0514	Full	Sat4j + both	minod	26,075,500 (85.55s)	27,804,400 (152.13s)
ORY	0508-0514	Part	Sat4j + res	minod	24,912,400 (106.14s)	27,096,900 (176.53s)
ORY	0508-0514	None	Sat4j + res	minod	TO	TO
ORY	0508-0514	None	Sat4j + res	minod-count	TO	TO
ORY	0619-0625	Part	Sat4j + res	minod-count	20,948,424 (98.29s)	21,532,898 (144.3s)
ORY	0619-0625	Part	Sat4j + res	minod	20,409,552 (88.33s)	21,322,962 (155.64s)
ORY	0619-0625	Full	Sat4j + both	minod-count	18,205,391 (112.1s)	19,262,126 (167.75s)
ORY	0619-0625	Full	Sat4j + res	minod-count	17,258,416 (63.49s)	18,976,432 (92.54s)
ORY	0619-0625	Full	Sat4j + both	minod	16,825,514 (93.25s)	18,789,961 (281.9s)
ORY	0619-0625	Full	Sat4j + res	minod	16,289,644 (56.38s)	18,166,148 (80.11s)
ORY	0619-0625	None	Sat4j + res	minod	TO	TO
ORY	0619-0625	None	Sat4j + res	minod-count	TO	TO
ORY	0626-0702	Part	Sat4j + res	minod	21,877,213 (94.2s)	22,827,742 (162.2s)
ORY	0626-0702	Full	Sat4j + res	minod	19,733,365 (71.73s)	20,293,372 (100.55s)
ORY	0626-0702	Part	Sat4j + res	minod-count	19,665,432 (80.39s)	22,407,406 (179.84s)
ORY	0626-0702	Full	Sat4j + both	minod-count	18,780,875 (126.77s)	20,624,204 (287.38s)
ORY	0626-0702	Full	Sat4j + res	minod-count	17,272,272 (53.83s)	20,386,156 (86.4s)
ORY	0626-0702	Full	Sat4j + both	minod	16,438,134 (93.94s)	20,513,757 (176.6s)
ORY	0626-0702	None	Sat4j + res	minod	TO	TO
ORY	0626-0702	None	Sat4j + res	minod-count	TO	TO
ORY	0703-0709	Part	Sat4j + res	minod	19,519,772 (90.62s)	22,517,931 (146.76s)
ORY	0703-0709	Part	Sat4j + res	minod-count	18,903,838 (78.75s)	22,797,314 (130.23s)

Continued on next page

Term	Period	Decompo	Solver	Model	First B (Time (s))	Last B (Time (s))
ORY	0703-0709	Full	Sat4j + res	minod-count	17,889,794 (63.98s)	19,728,182 (101.84s)
ORY	0703-0709	Full	Sat4j + both	minod	17,616,033 (117.82s)	19,804,539 (276.38s)
ORY	0703-0709	Full	Sat4j + res	minod	17,609,671 (69.68s)	18,820,080 (89.86s)
ORY	0703-0709	Full	Sat4j + both	minod-count	17,310,521 (103.03s)	19,814,326 (270.62s)
ORY	0703-0709	None	Sat4j + res	minod	TO	TO
ORY	0703-0709	None	Sat4j + res	minod-count	TO	TO

Table 5.3: Results for overlapping-nbmax family.

Tables 5.1, 5.2 and 5.3 present the first bound and the last bound for each family of instances. We can remark that the results were less than satisfactory. Nonetheless, an interesting observation can be made for the **overlapping-nbmax** family that integrates **sum** and **count** constraints: the solvers managed to identify a bound in the majority of scenarios. The notable exception was instances without decomposition, which can be primarily attributed to the size of the instance post-encoding.

5.2.5 Conclusion

This section proposed to use different Boolean encodings for the domain of CSP variables to define new encodings based on PB constraints. The main advantage of the proposed encodings is that they allow to exploit the inference power of PB solvers, and especially their ability to count.

The experimental analysis showed that our encodings, combined with the use of PB solvers, allow us to solve efficiently problems containing mainly **sum** and **cardinality** constraints [FW22b]. However, this good performance does not generalize to problems containing other constraints in which native CP solvers remain faster.

From perspectives, we would like to investigate the use of different encodings for the constraints we already have, so has to improve the performance of PB solvers when solving CSPs, especially by favoring the use of pure PB constraints rather than clauses. Another perspective is to exploit the complementarity between the different solving paradigms for CSPs (either native, based on SAT, or based on PB) to leverage the best of all approaches.

5.3 Parallel solving

As seen in previous sections, various solvers have been proposed for solving constraint programming problems. Even though these solvers perform well in practice, some problems remain hard to solve in a reasonable time. This is particularly true for enumeration and optimization problems, which require to explore the whole search space. To solve such problems, a tempting approach is to exploit the architecture of modern CPUs (or even cloud computing) to develop parallel or distributed techniques. For example, in the previous sections, we exploited the structure of the problem to break down the problem of allocating check-in desk into sub-problems that are solved in parallel.

In this context, several approaches have been proposed, such as the *portfolio*, which runs different solvers in parallel on the same input, or *Embarrassingly Parallel Search*, which consists of assigning different parts of the search space to the solvers that are run in parallel [RRM13]. These approaches have been implemented in different solvers, for instance Gecode¹⁶, Toulbar2¹⁷,

¹⁶<https://github.com/Gecode/gecode>

¹⁷<https://github.com/toulbar2/toulbar2>

Choco [PFL16] or OR-Tools [PF22].

However, these approaches may be hard to implement since they imply the use of complex synchronization mechanisms and resource management. To simplify the development of such solvers, different libraries and frameworks have been proposed, such as pFactory [AGL⁺19], which provides simplified tools for dealing with parallel approaches, PaInleSS [LFBSK17] for developing parallel SAT solvers, or Bob++ [GL07, ML13] for constraint programming (built on top of the solvers Gecode and OR-Tools). They provide the elementary components required for the development of parallel and distributed solvers, allowing to focus on the development of new strategies.

In the same spirit, this section introduces a new C++ library designed to be cross-platform and to support a wide variety of solvers that may be written in different languages (namely, C, C++, and Java, thanks to JNI). Our framework provides several interfaces defining different strategies for task assignment, information sharing, and synchronization between solvers, among others. These interfaces may be implemented independently to easily try new parallel approaches, or to implement existing approaches. In this section, we focus on EPS and show that our framework can be used to implement existing decomposition techniques, as well as new ones that can be competitive with respect to the state of the art. Information sharing between solvers is based on the MPI (*Message Passing Interface*) library, which allows to deal with both parallel and distributed architectures transparently. The choice of the information to share is left to independent strategies, allowing to prototype new ideas quickly.

5.3.1 Related works

Among existing parallel approaches, the *portfolio* is probably the “simplest” one in its design. This approach aims to simulate the *Virtual Best Solver* (VBS), i.e., the solver that one would obtain by selecting among a set of solvers the one that performs best on a specific input. As proposed in ppfolio [Rou12], it consists of executing different solvers (or different configurations of the same solver) in parallel on the same input. Typically, the answer given by the portfolio will be the one of the first solver to find a solution or prove the unsatisfiability of the input. Many solvers propose an implementation of this technique, such as, for instance, Gecode¹⁸, Sat4j [LBP10], pLingeling [Bie16], Choco [PFL16], dSyrup [ALST17].

Another approach is the *Embarassingly Parallel Search* (EPS) [RRM13, MRR16], which applies a *cube-and-conquer* technique. The basic idea of this approach is to divide the search space into different subparts, which are assigned to the different available solvers based on a so-called *guiding-path* [ZBH96]. A guiding path is made by choosing a set of variables, and each solver starts by assigning these variables in different ways. The chosen set of variables and their corresponding assignment is generally referred to as a *cube*. Most of the time, there are more cubes than available solvers (the authors of EPS advise to generate around 30 cubes per solver). The cubes are thus put in a queue until a solver becomes available again and eventually takes one of them. Whenever a solver finds a solution, then the formula is satisfiable. A single solver cannot in general prove the unsatisfiability of the input unless it finds a conflict that is not due to the guiding path.

The main problem with space splitting is that some parts of the search space maybe easy, while others may be hard, and the solvers exploring easy parts may become idle. To avoid this, the *work stealing* approach has been proposed, as in Gecode for instance. A running solver splits its current search space, and idle solvers may then take some of the obtained subparts and try

¹⁸<https://github.com/Gecode/gecode>

to solve them. A running solver can also split its search space at some point (e.g. if it detects that it is exploring a hard part) and enqueue it so as not to be interrupted later on. To avoid such issues, another option is to use a hybrid approach, which consists in starting the search by space splitting and then switching to portfolio, either when a hard subpart of the search space is reached [Blo05] or when load balancing becomes an issue [MML10].

To improve the performance of these approaches, one could also *share* information between the running solvers. In the case of SAT solving, learned constraints may, for instance, be shared between the different solvers. This requires, in particular, to select *which* constraints to share, often by considering their size: for instance, in the first version, **pLingeling** [Bie16], only constraints with 1 variable (a.k.a., unit clauses) were shared. Another key aspect of constraint sharing is to decide *when* to share them. [AS14] proposes to perform a lazy constraint exchange while [NI20] delays this exchange and estimates the time needed for this exchange to ensure reproducibility. In a more general and higher level, information exchange can be done on optimization problems by sharing the intermediate solutions and bounds obtained during the search to all of the solvers. This prevents the solvers from exploring parts of the search space that do not contain the optimal solution (see, e.g., [Der16, Section 3.1]).

In the rest of this section, we focus on the implementation of EPS in our framework and new space-splitting strategies we designed, as doing so allows the exploitation of highly efficient state-of-the-art solvers without being too intrusive in their implementation.

5.3.2 Architecture of our Framework

An important building block of our framework is the **Universe** solver interface, that we developed as an independent tool. It provides an object-oriented interface for any kind of solvers (SAT, PB, or CP) allowing to adapt any existing solvers to use it on our framework (its design also allows to exploit implementing solvers in a wide variety of constraint solving-based applications). The interfaces are available in both C++ and Java (using the *Java Native Interface*), allowing to adapt almost all existing state-of-the-art solvers without relying on inter-process communication. A (simplified) class diagram of this library is given in Figure 5.5, showing that the main interface, named **IUniverseSolver** may be used to load an existing problem from its file, and to solve it with or without hypotheses on the variable assignments (named **assumptions** in the diagram). The other interfaces specialize this interface to allow feeding the solver with constraints programmatically.

In our parallel framework **PANORAMYX**, an abstract factory is used to create instances of **IUniverseSolver**, which can then be executed in parallel. Figure 5.7 shows a sequence diagram describing such an execution of our framework, independently of the parallel paradigm that is being executed. One can observe that the *main* solver can invoke methods on a *remote* solver, which is a proxy hiding the network communication (in our case, based on MPI) with an actual *worker* solver, which may be different processes executed either on the same machine or on a distant machine (in the case of distributed solving). All solving tasks are executed *asynchronously*: whenever a solver either finds a solution of its assigned problem or proves its unsatisfiability, it sends the answer to the main solver, which decides what is the next action to perform (either terminate or assign new tasks to the running solvers).

The main solver is the one that actually implements the parallel paradigm to be executed, such as portfolio or EPS. To make easier its implementation, our framework provides a template-method based **AbstractParallelSolver** (which is also an **IUniverseSolver**), whose methods are given in Figure 5.6. These methods allow to react to the various solving events that may occur asynchronously, as the various paradigms do not handle them in the same way (for instance, if

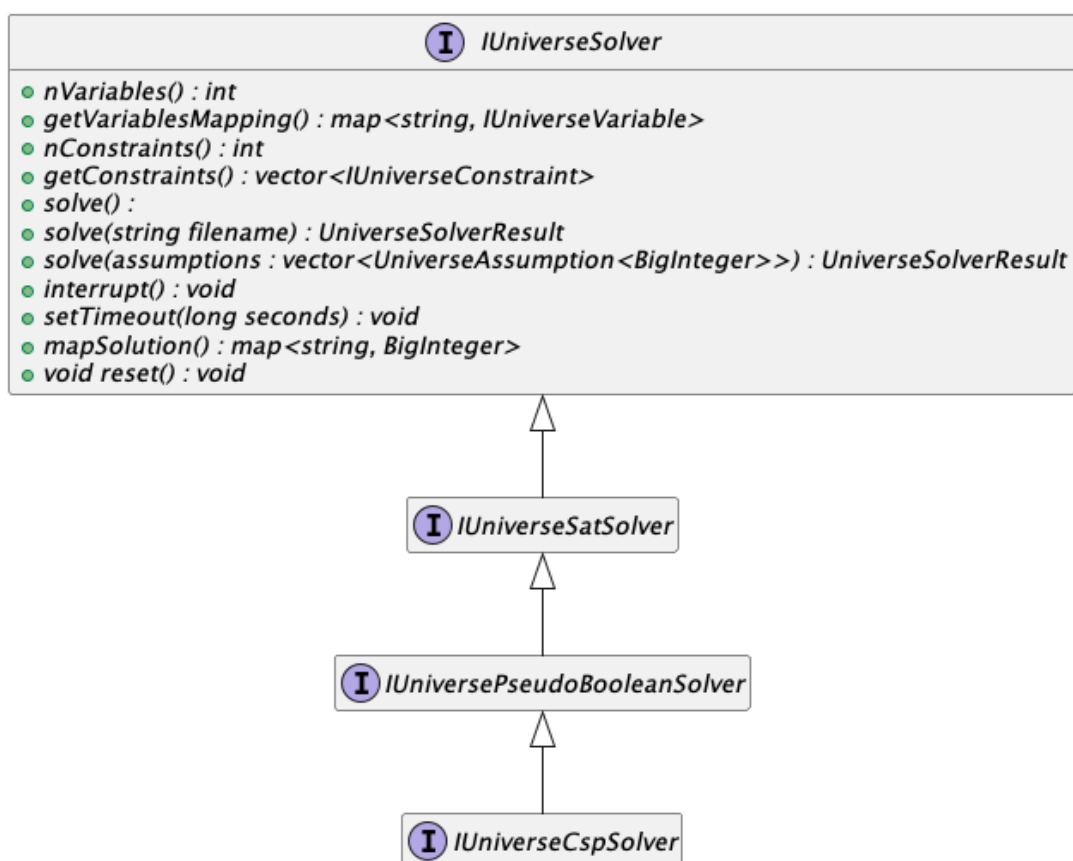


Figure 5.5: Some interfaces from the Universe solver interface (only relevant methods are shown). `BigInteger` is an alias for either `long long` or `mpz_class` from `gmp`, depending on compile options.

a solver proves the unsatisfiability of its problem in a portfolio, then the input problem itself is unsatisfiable, while it is not necessarily the case for EPS).

5.3.3 EPS approaches

This section details the implementation of EPS in our framework and how to implement different space-splitting strategies for this approach.

5.3.3.1 Our Implementation

When the EPS approach is used in our framework, two threads are executed in parallel in the main process. The first one runs Algorithm 4. It is mainly a loop (l. 1) that iterates over all possible cubes generated by the `generateCubes` function (see the following subsection), which assigns the cubes to the available solvers. The `availableSolver` function (l. 2) is blocking: it waits until a solver has no assigned task before returning it. The solving task is run asynchronously (l. 3): the answer returned by the solver is received and handled in the second thread.

Algorithm 4: EPS solving loop.

```

1 foreach cube  $\in$  generateCubes() do
2   solver  $\leftarrow$  availableSolver()
3   async solver.solve(cube)
4 end
```

This thread runs Algorithm 5. This loop receives the solver answers as messages that are received one after the others (l. 3). The `receive` function blocks until a message arrives from a particular solver. If this message is SATISFIABLE (l. 4), then a solution to the problem has been found, and the input problem is satisfiable, thus all solvers may be interrupted (l. 5-6). If the message is UNSATISFIABLE (l. 7), then the problem assigned to the solver (given by its cube) is unsatisfiable, which does not necessarily mean that the input problem is unsatisfiable, so the solver is marked as available again, allowing new problems to be assigned to it (l. 9).” The input problem is unsatisfiable only if all generated cubes lead to an UNSATISFIABLE answer, which

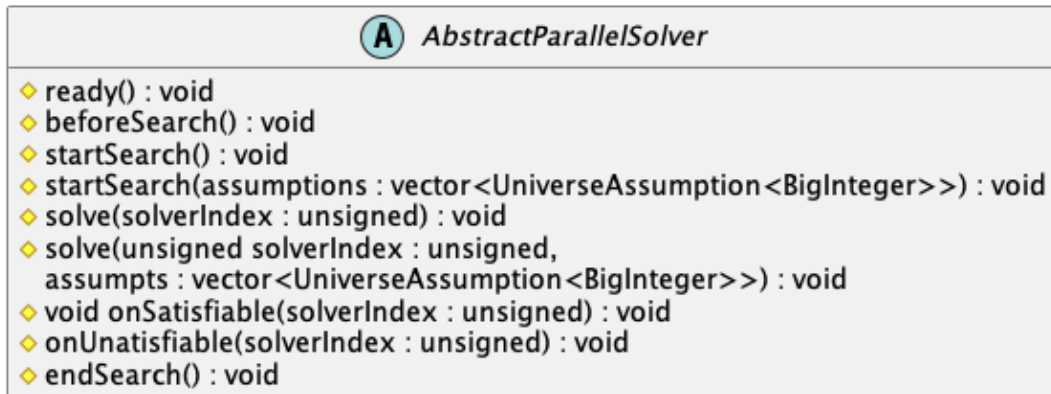


Figure 5.6: The protected methods to implement when developing a new parallel paradigm.

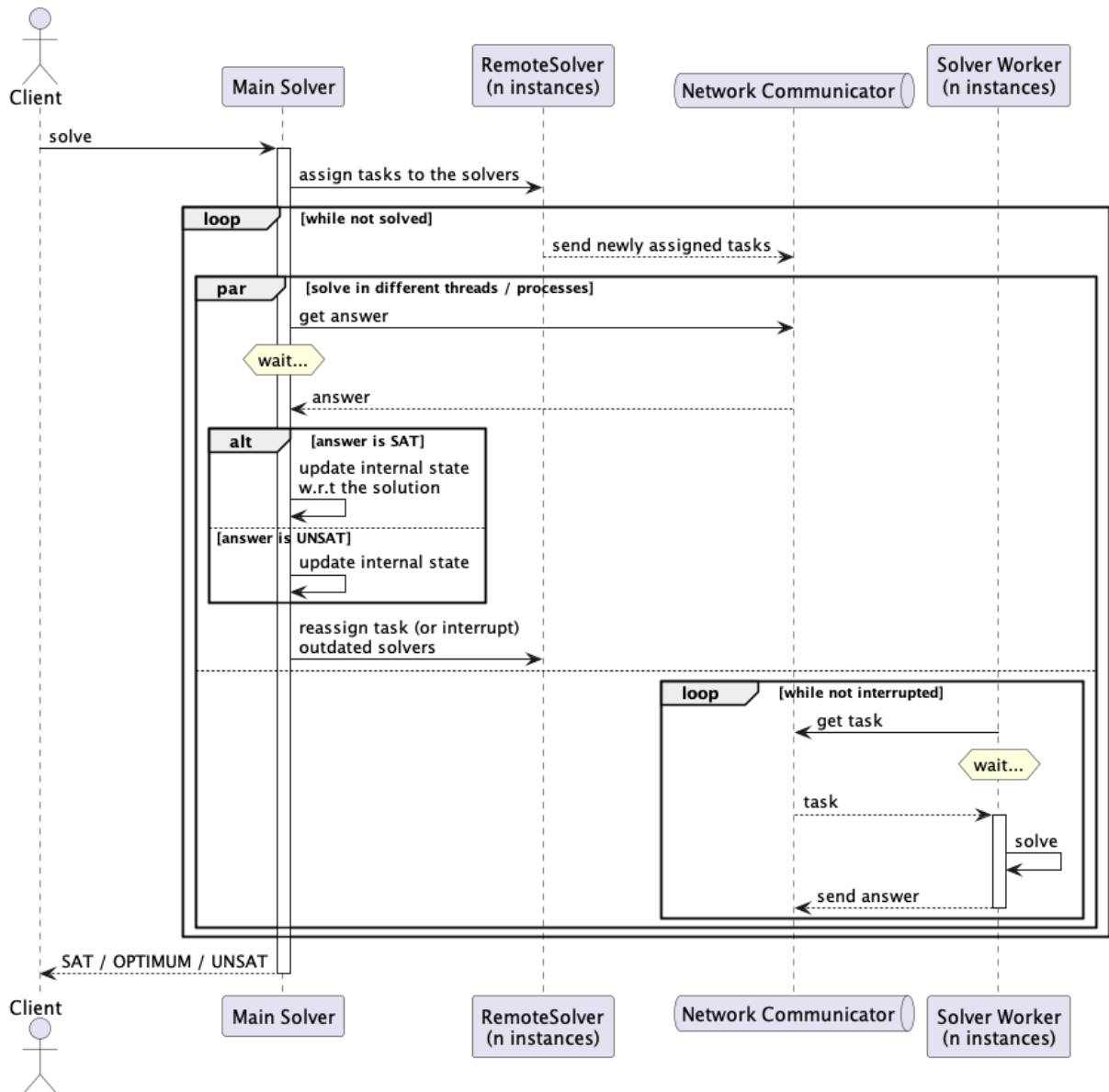


Figure 5.7: Sequence diagram for an execution of our parallel framework.

is why the loop terminates at that point (l. 2). The `nbUnsat` allows thus to determine the final answer to return (l. 12-15).

Algorithm 5: EPS message handle loop.

```
1 nbUnsat ← 0
2 while nbUnsat < nbCubes do
3   message, solver ← receive()
4   if message is SATISFIABLE then
5     interrupt()
6     return SATISFIABLE
7   else if message is UNSATISFIABLE then
8     nbUnsat ← nbUnsat + 1
9     markAsAvailable(solver)
10  end
11 end
12 if nbUnsat = nbCubes then
13   return UNSATISFIABLE
14 end
15 return SATISFIABLE
```

5.3.3.2 Generating Cubes

In order to allow an easy implementation of various space-splitting strategies for the EPS approach provided in our framework, we define an interface for generating cubes. Its class diagram is given in Figure 5.8. The strategy itself is defined as `ICubeGenerator`, and uses an `IConsistencyChecker` to check (or not) the arc-consistency of the cube. More specifically, the check may be performed either when the cube is entirely generated or each time a variable assignment is assigned to it so as to avoid generating useless cubes. The cubes are generated in a `Stream`, which is basically a *lazy* iterator: when possible, the next cube is only computed when asked to, instead of generating all cubes before starting to assign them to the solvers. Doing so allows to be both time and memory efficient during cube generation. We note that it is also possible to adapt vectors and queues as `Stream` instances when lazy generation is not applicable.

A first possible option we implemented to generate cubes is a *lexicographic* approach: variables are added to the cube in lexicographic order, and the values to which they are assigned are tried in ascending order from their domains. To avoid generating too many cubes, we follow the recommendations of the authors of EPS, by generating around 30 cubes per worker. This number may be estimated by iterating over the variable and multiplying the size of their domains until this limit is reached: the number of iterated variables gives the size of the cubes to generate. Despite being a naive approach, the lexicographic approach has the advantage of allowing the lazy generation mentioned above, as determining the next cube from the current one is straightforward.

This is not the case with the *cartesian product iterative refinement* (CPIR) [Der16, Section 3.2.1], which requires the cubes to be generated before their assignment to the solvers. This approach, that we also implemented, is given in Algorithm 6. First, an empty cube is put in a priority queue (l. 1). The priority of this queue is given by the size of the cartesian product of the domain of the variables that are not assigned by the cube (problems with bigger size are

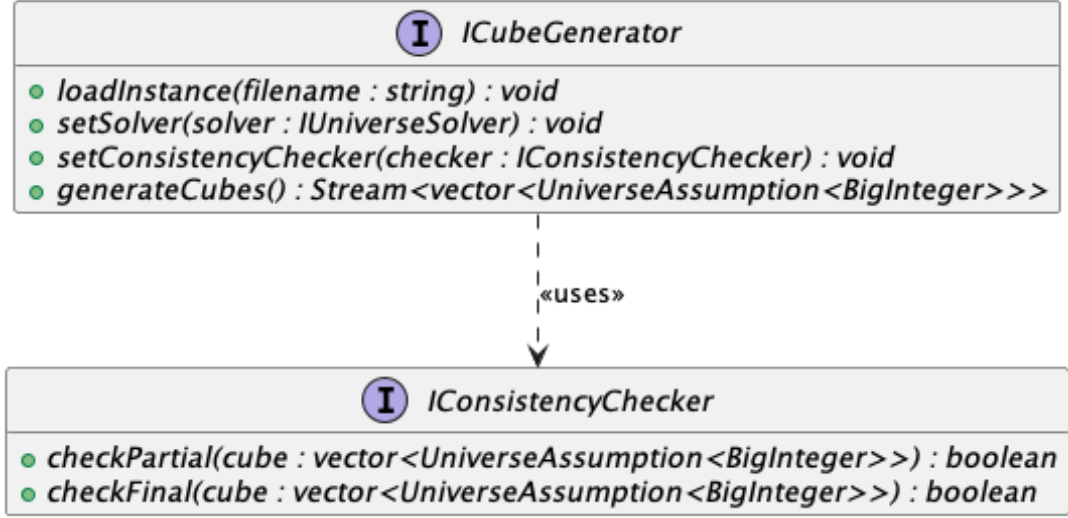


Figure 5.8: The interfaces defining the cube generation strategies.

taken first). Then, while the queue does not reach the expected size (as before, 30 cubes per worker) (l. 2), a cube is taken from the queue (l. 3), and an unassigned variable is selected from the problem to extend this cube (l. 4). The extended cubes are obtained by trying all possible assignments for this variable in different cubes, that are added to the queue (l. 5-6). Each time a cube is extended, constraint propagation may be applied on the problem to both check the arc-consistency of the generated cube and reduce the domain of the unassigned variables.

Algorithm 6: CPIR cube generation.

input : A problem $\mathcal{P}$.
output: A Stream of cubes.

```

1 queue ← {}
2 while |queue| < cubeLimit do
3   cube ← queue.removeFirst()
4   var ← unassignedVariable( $\mathcal{P}$ , cube)
5   foreach val ∈ dom(var) do
6     queue.add(cube ∪ {var = val})
7   end
8 end
9 return Stream(queue)

```

5.3.3.3 A New Cube Generation Strategy

In order to show the versatility of our framework, we also implemented a new cube generation strategy based on the partitioning of the *dual hypergraph* of the problem to solve. This hypergraph has as hypervertices the constraints of the problem, while its hyperedges correspond to the variables of the problem: each hyperedge joins the constraints with the corresponding variable in their scope. For instance, suppose that we consider a problem composed of the variables x, y, z, t, u , and v and the constraints c_1, c_2, c_3 and c_4 such that $\text{scp}(c_1) = \{y, z\}$, $\text{scp}(c_2) = \{z, u\}$,

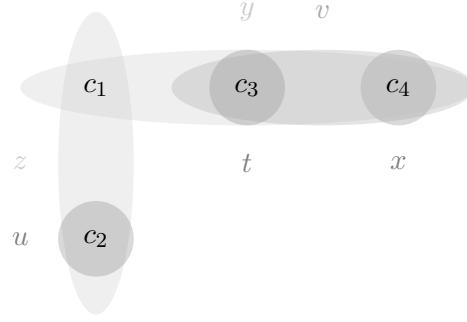


Figure 5.9: Example of a dual hypergraph.

$\text{scp}(c_3) = \{y, t, v\}$, and $\text{scp}(c_4) = \{x, y, v\}$, then the dual hypergraph associated to that problem is the one given in Figure 5.9. Partitioning this hypergraph (for instance, into 2 subhypergraphs) is to identify which hyperedges (i.e., variables) to remove so that the obtained hypergraphs are composed of several (in our example, 2) unconnected components. The set of hyperedges to remove is called the *cutset*. If we apply such a partitioning to the example problem given above, we can for instance see that a possible cutset is $\{y\}$, as removing y gives on the one hand the hypergraph composed of the hypervertices $\{c_1, c_2\}$, and on the other hand, the hypergraph composed of the hypervertices $\{c_3, c_4\}$, which are not connected by any hyperedge. Finding a good partitioning often means to find a *balanced* one, with approximately the same number of constraints in each subhypergraph. This problem is NP-hard, so we propose to use the open-source **KaHyPar** library [SHG⁺22], which provides a good approximation algorithm for finding our cutset of interest. This cutset is used in the cube generation strategy to select the variable on which to create the cubes. Indeed, to remove a hyperedge, we can assign the corresponding variable, as it becomes constant afterwards, which is exactly what a cube does. As there may be many variables in the cutset, though, we restrict the number of variables to put in the cube using the same approach as in the lexicographic strategy described in the previous section.

5.3.4 Experiments and Results

To evaluate our approach, we used all the instances from the CSP track of the XCSP22 competition¹⁹ noted $\mathcal{I}_{11}$:

$$\mathcal{I}_{\text{XCSP22}}^{\text{CSP}} \quad (\mathcal{I}_{11})$$

We have run different state-the-art solvers, namely **Choco** [PFL16] in its parallel mode (which implements a portfolio), **Toulbar2**²⁰, and **RBO**²¹, which implements the approach presented in [JKT16]. For our implementation, we run the EPS approach using the different cube generation strategies (namely, **lexico**, **cpir**, and **kahypar**, referring respectively to the lexicographic cube generation, the cartesian product iterative refinement and to the hypergraph decomposition based on **kahypar**) using two solvers, namely **ACE** [Lec23] and **Sat4j** [LBP10]. We note, however that **Sat4j** does not support the operations required for the CPIR approach to reduce the size of the domains of the variables (as they are encoded as Boolean variables), so we did not run it for the **cpir** strategies. In all cases, we both run the approach by checking the arc-consistency

¹⁹<https://www.cril.univ-artois.fr/XCSP22/>

²⁰<https://github.com/toulbar2/toulbar2>

²¹<https://github.com/Terrioux/BTD-RBO>

either each time it is extended (**partial**) or only when it is complete (**final**). The set of solvers Ψ_7 is defined as follows:

$$\left\{ \mathcal{S}_g^c \left| \begin{array}{l} \mathcal{S} \in \{\text{ACE}, \text{Sat4j}\} \\ g \in \{\text{lexico}, \text{cpir}, \text{kahypar}\} \\ c \in \{\text{partial}, \text{final}\} \end{array} \right. \right\} \cup \{\text{btd}, \text{toulbar2}, \text{choco}\} \quad (\Psi_7)$$

Experiments were run on a cluster of computers equipped with 80-core Intel Xeon Gold 6248 (2.5 Ghz) and 768 GB of RAM. The time limit was set to 1200 seconds and the memory limit to 64 GB. All approaches were given 20 cores (in our case, it means that there were 1 main solver and 19 worker solvers).

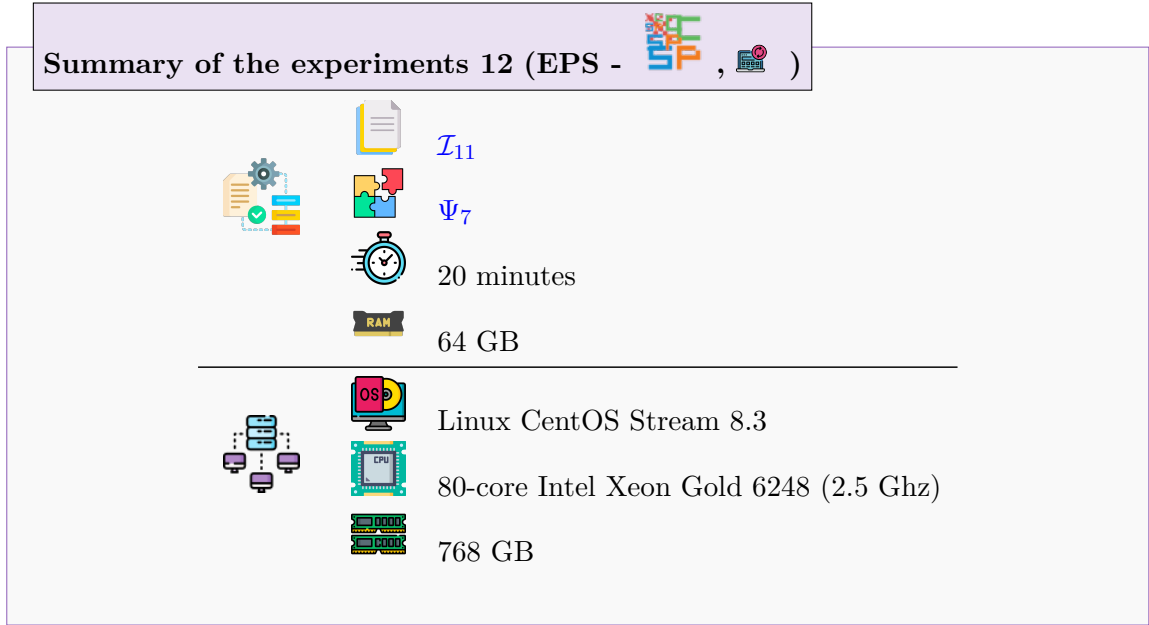


Figure 5.10 gives an overview of the results. We can see that the solver **Choco** is really fast in its portfolio mode, which is not really comparable to our approach (it does not have the cost of generating cubes, and it is necessarily at least as good as the sequential version of the solver it runs internally). An interesting observation is that our new approach for generating cubes (based on hypergraph decomposition with KaHyPar) is faster than the lexicographic and CPIR approaches on both **Sat4j** and **ACE**, and allows the solver to solve more instances. Additionally, we run the sequential version of **Choco** and **ACE** to identify hard instances, which we define as those that are solved in more than 600 seconds or not solved at all by both sequential solvers. The results are given in Table 5.4 and show that the parallel approaches improve the runtime or even solve inputs that were not solved before.

5.3.5 Conclusion

The third section introduced a new framework for developing parallel and distributed constraint solvers. Thanks to a set of object-oriented solver interfaces, it allows to seamlessly integrate various kinds of state-of-the-art SAT, PB, or CP solvers, as well as defining strategies for running them in parallel. In particular, this section focused on the development of EPS-based approaches, and showed that new techniques could easily be developed in our framework, even beating the state-of-the-art. As perspectives, we would like to extend our framework to design other

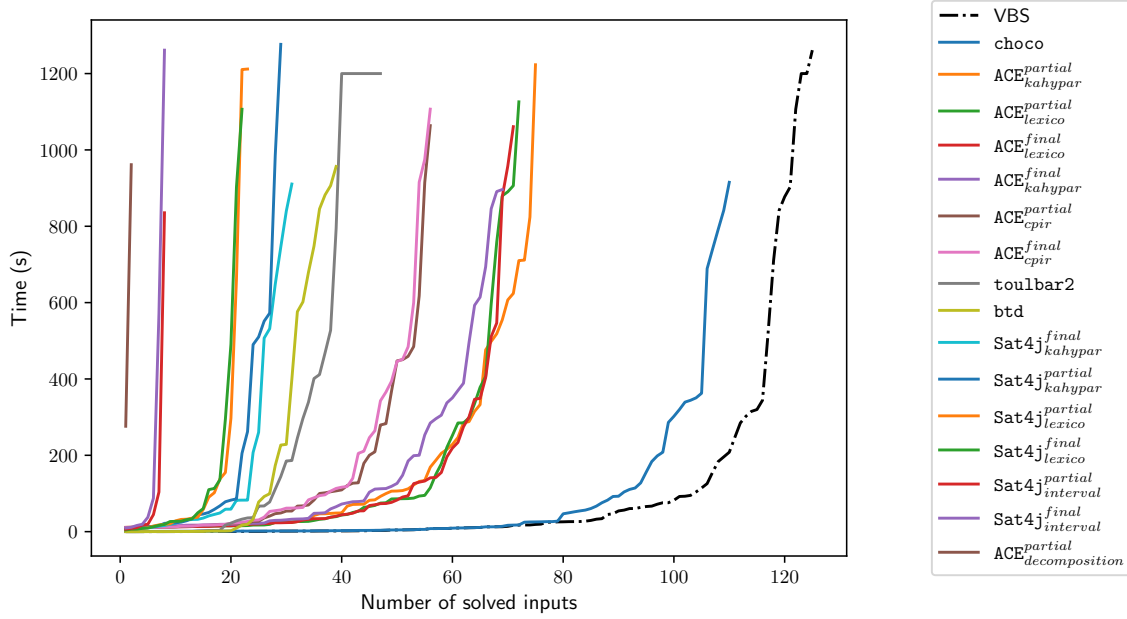


Figure 5.10: *Cactus plot* of the executed solvers. For each solver (given by its line), we can easily read the number of inputs on which it has been run until completion within a certain time limit, given in seconds of the wall-clock time (and not CPU time). In particular, the more a solver is to the right, the faster it is in general.

input	choco-sequential	choco	speed-up choco	ace-sequential	ace-kahypar-partial	speed-up ace
CarSequencing_table_85_02_c22	TIMEOUT	688.990000	1.741680	TIMEOUT	99.763000	12.028508
CarSequencing_table_90_01_c22	TIMEOUT	TIMEOUT	-	TIMEOUT	710.158000	1.689765
DiamondFree_70_c22	TIMEOUT	302.177000	3.971182	TIMEOUT	TIMEOUT	-
Eternity_08_07_c22	TIMEOUT	TIMEOUT	-	TIMEOUT	518.337000	2.315096
Eternity_09_07_c22	TIMEOUT	TIMEOUT	-	717.764000	TIMEOUT	0.598137
Hadamard_29_c22	TIMEOUT	915.134000	1.311283	TIMEOUT	82.159000	14.605825
Hidato_table_13_13_None_c22	TIMEOUT	5.861270	204.733786	653.617000	TIMEOUT	0.544681
Hidato_table_14_14_None_c22	TIMEOUT	9.106500	131.774008	TIMEOUT	TIMEOUT	-
Hidato_table_15_15_None_c22	TIMEOUT	12.814200	93.646111	TIMEOUT	TIMEOUT	-
NumberPartitioning_260_c22	TIMEOUT	60.539900	19.821638	748.510000	TIMEOUT	0.623758
Pb_BeauxArts_K77_c22	TIMEOUT	TIMEOUT	-	TIMEOUT	246.689000	4.864424
QuasiGroup_base_v6_16_c22	TIMEOUT	92.163500	13.020339	TIMEOUT	TIMEOUT	-
RoomMate_sr0300_c22	TIMEOUT	TIMEOUT	-	747.281000	TIMEOUT	0.622734
SportsScheduling_dummy_16_c22	TIMEOUT	TIMEOUT	-	801.936000	315.026000	2.545618

Table 5.4: Detailed results on hard instances for Choco and the best parallel version of ACE, compared to their sequential versions. The *speed-up* shows how much faster (or slower) the parallel version of the solver is compared to its sequential version. Values in bold are the highest on the instance. Instances that are solved by neither of the approaches are omitted.

“general” parallel approaches (in addition to portfolio and EPS), such as ones exploiting the decomposition of a problem into independent subproblems. We would also like our framework to implement more intrusive techniques, such as those based on work stealing or those allowed to share information between the running solvers.

5.4 Conclusion

In this comprehensive exploration spanning three distinct sections, we navigated through various techniques and methodologies designed to enhance the efficiency of constraint solvers. Initially, we delved into the [ABD](#) technique, a method that proactively tweaks objective constraints, demonstrating potential performance enhancements, particularly when integrated with the PB `Sat4j` solver.

The subsequent section proposed a novel approach using Boolean encodings for CSP variable domains, highlighting the capabilities of PB solvers, especially in inference and counting. However, its efficacy was primarily notable for certain types of constraints, leaving room for further refinement and leveraging diverse encoding paradigms.

The concluding section introduces a new framework for parallel and distributed constraint solver development. By integrating a multitude of state-of-the-art solvers, it showcased the flexibility and power of the framework, suggesting its adaptability for various parallel approaches and integration of more sophisticated techniques.

Part III

Predicting flow passengers and flight delays

Chapter 6

Predicting the Number of Passengers with Reduced Mobility on Flights

Contents

6.1 Introduction	147
6.2 Exploratory data analysis	148
6.3 Model definition	153
6.3.1 Basic flight features (BFF)	153
6.3.2 PRM count features	153
6.4 Model configuration	154
6.5 Correlation Analysis	154
6.6 Results	154
6.7 Conclusion	159

6.1 Introduction

In an age of increasing globalization and unprecedented connectivity, air transport has become essential to reducing geographical distances and fostering international interaction.

The aviation sector is pivotal in molding the contemporary world, offering millions the opportunity to discover new places, engage in business, reconnect with family, and immerse in diverse cultures. However, this remarkable mobility that aviation affords us also brings to light a crucial challenge — ensuring that air travel is accessible and inclusive for all individuals, regardless of their physical abilities.

The cornerstone of aviation inclusivity is to provide assistance and accommodations to passengers with reduced mobility (PRMs). These individuals may include seniors, people with physical disabilities, temporary mobility impairments, and others requiring specialized support during their journey through airports and aboard aircraft. As the demand for air travel continues to recover towards pre-COVID-19 trends, the aviation industry faces the task of meeting the needs of PRMs and ensuring that these passengers enjoy the same level of safety, comfort, and convenience as their able-bodied counterparts.

In Paris Airports, people with reduced mobility are managed by service providers. Paris Airports must therefore, determine the number of PRMs and communicate this to the service

providers, who will allocate the correct number of people to manage them. A prediction error can result in under- or over-staffing, which, in turn, can cause over-invoicing by service providers.

Therefore, our task will be to study the data available to us to find a technical solution to the need to predict the number of PRMs on flights departing from Paris Charles de Gaulle or Orly airports. The aim is to find the total number of PRMs for the day at each airport. To do this, we will aggregate the predictions made on each flight individually.

Like any application and any problem, there may be several machine learning techniques (here, regression techniques) that can be used. It is, therefore, necessary to carry out a comparative study to evaluate them and decide which one to choose, taking into account certain considerations (such as the technologies to be used, scalability, operational constraints, security, robustness, explainability, etc.). In our case, Paris Airports which is the end-user of our proposal has already positioned itself on Microsoft technologies (in particular through the `ML.net` framework) and wants as an ensemble of regression trees (called in Microsoft jargon `FastTrees`) and already deployed by Paris-CDG. This is mainly justified by the proven performance of this technique at Paris-CDG, its speed of prediction and the possibility of training and updating the model very quickly. So, as we will see in the results section, we have focused our tests on technologies easily deployable in ADP infrastructures (e.g., `FastTree` from the library `ML.net`).

The rest of the chapter is organized as follows: in section 6.2, we explore the dataset from Paris Airports. Section 6.3 presents our model's attributes and configuration. Section 6.5 investigates the correlations between the various attributes (features) and the target variable. Before conclusion, we continue with some results of our proposal model (Section 6.6).

6.2 Exploratory data analysis

Our study utilized data from CDG and ORLY between August 8, 2022, and September 15, 2023. At Charles de Gaulle Airport (CDG), there were 215,478 arrival flights and 219,607 departing flights during this period. Regarding passenger counts at CDG, the departing flights carried a total of 36,514,206 passengers, while the arrival flights saw 34,877,654 passengers. This translates to a daily average of 177,591 passengers, of which about 2,685 were passengers with reduced mobility.

Concerning Orly Airport, the airport recorded 112,272 arrival flights and 114,593 departing flights. On any given day, Orly managed an average of 89,269 passengers. Notably, among these daily figures, approximately 840 passengers were individuals with disabilities.

Figure 6.1 the evolution of total passenger count at CDG over the period studied. Similarly, Figure 6.2 shows the evolution of disabled passengers. Figure 6.3 highlights the fact that there are far more PRMs during the IATA summer season than the winter season²². Finally, Figure 6.4 shows the number of flights with and without PRM. Figures 6.5 to 6.8 present similar information but for Orly airport.

²²IATA Summer begins on the last Sunday of March and ends on the last Saturday of October. IATA Winter begins on the last Sunday of October and ends on the last Saturday of March.

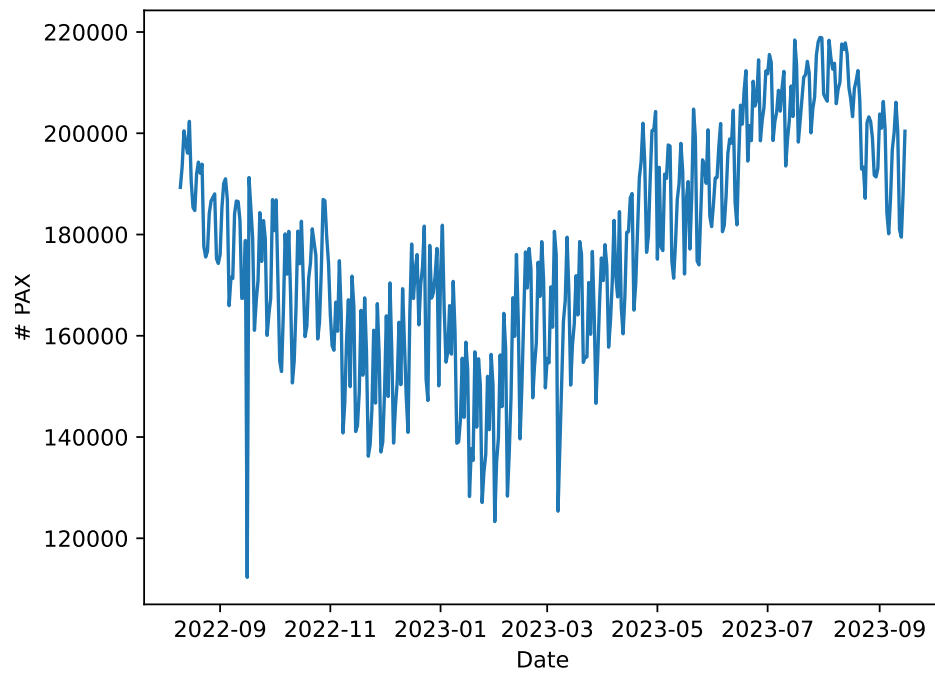


Figure 6.1: Evolution of the number of passengers during the period at CDG.

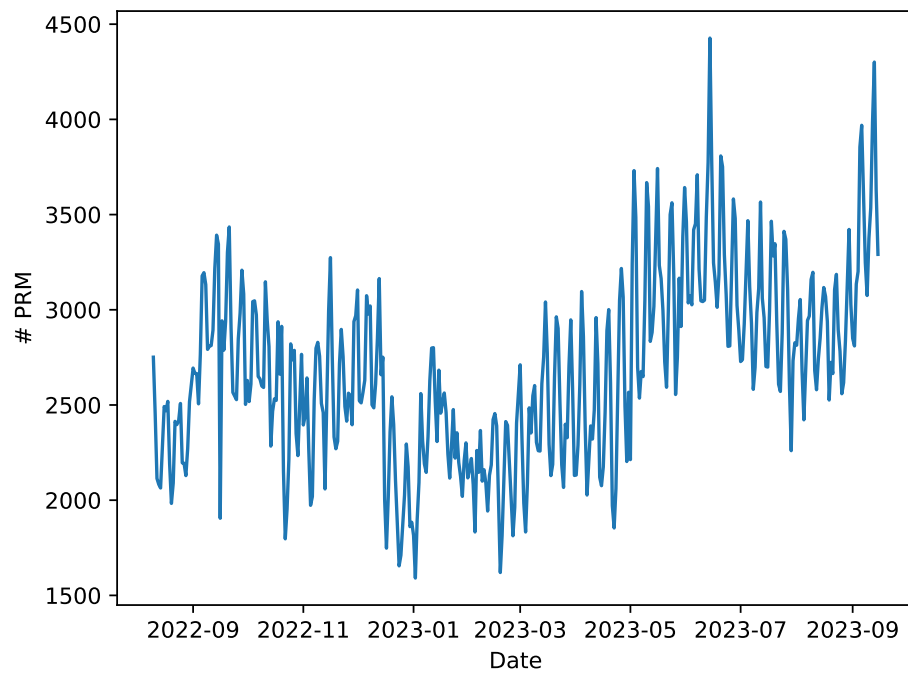


Figure 6.2: Evolution of the PRM number during the period at CDG.

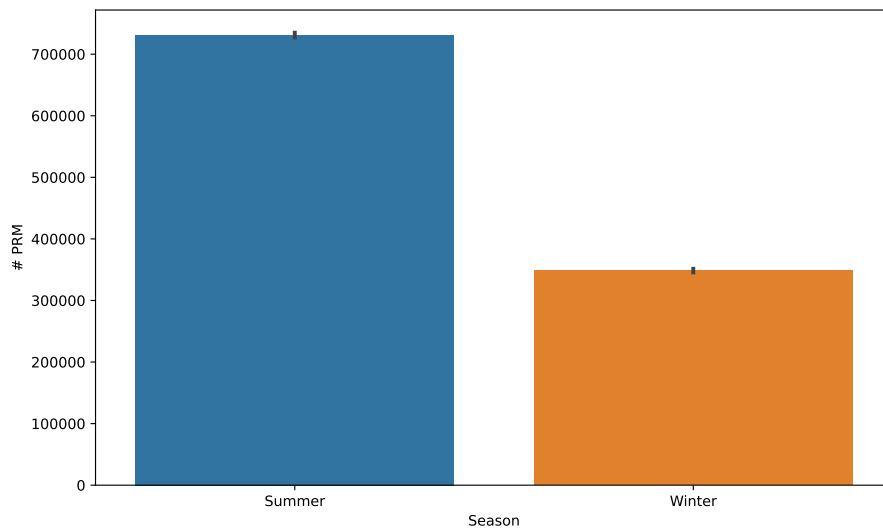


Figure 6.3: Number of PRM by IATA season at CDG.

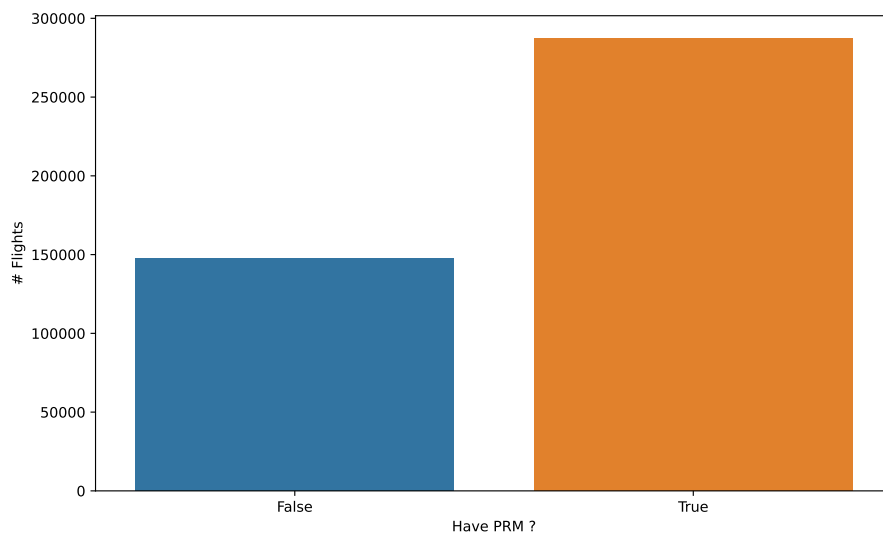


Figure 6.4: Number of flights with and without PRM at CDG.

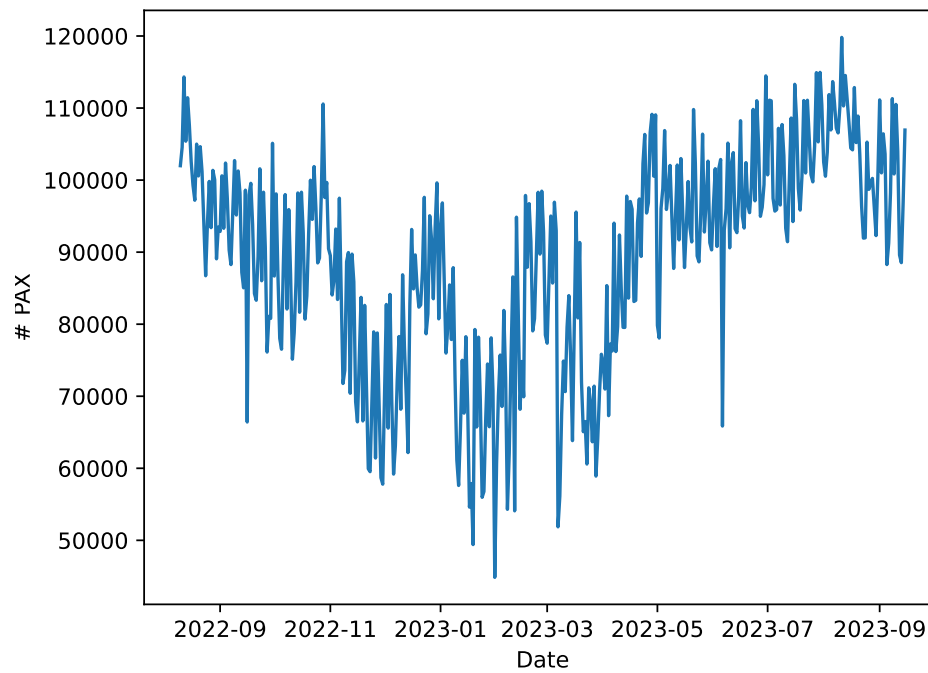


Figure 6.5: Evolution of the number of passengers during the period at Orly Airport.

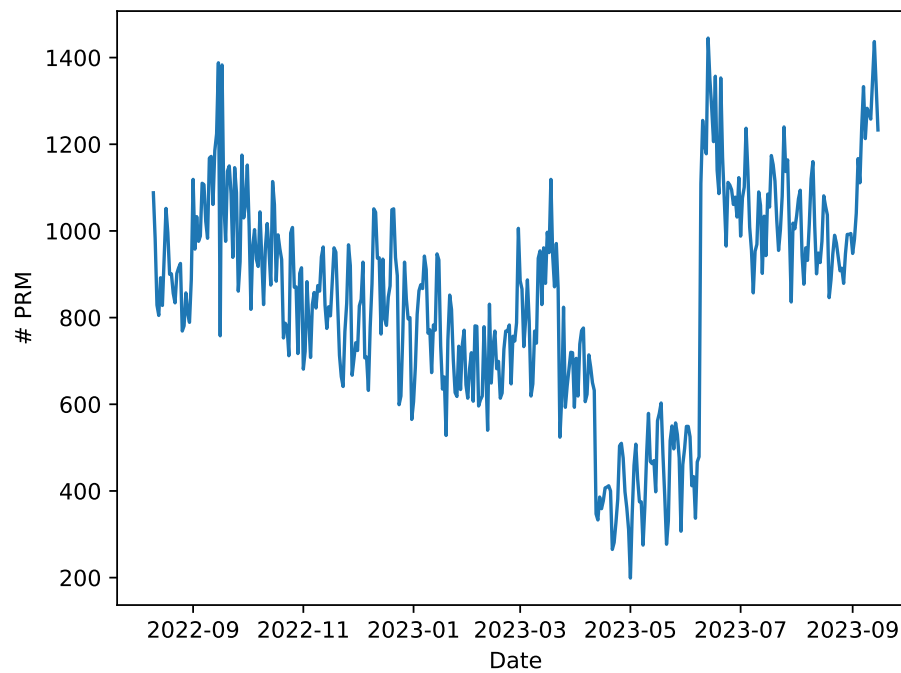


Figure 6.6: Evolution of the PRM number during the period at Orly Airport.

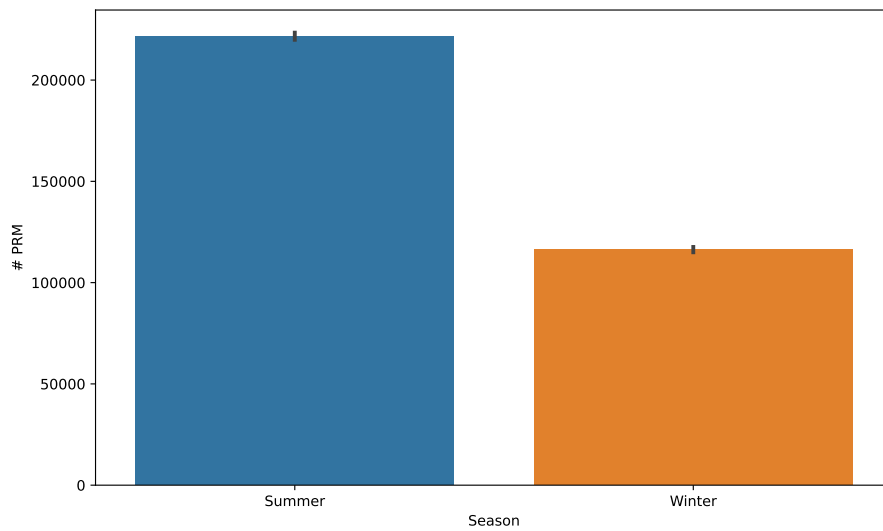


Figure 6.7: Number of PRM by IATA season at Orly Airport.

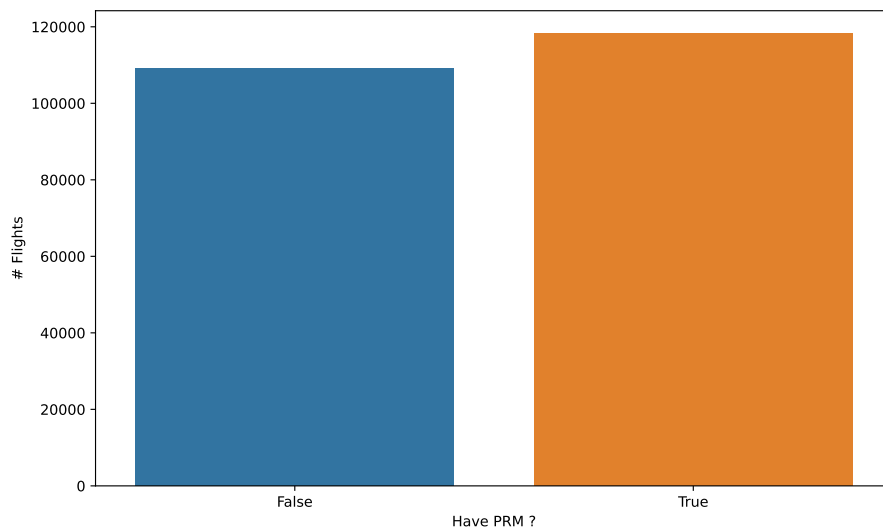


Figure 6.8: Number of flights with and without PRM at Orly Airport.

6.3 Model definition

In this section, we outline and underscore the key information available in the Paris operational information system, which holds significance for predicting the count of persons with limited mobility.

6.3.1 Basic flight features (BFF)

These are the basic characteristics of a flight (these features are listed in Table. 6.1). For example, the name of the airline operating the flight, the aircraft type, the IATA code of the flight's destination, and the IATA season (Summer or Winter).

Feature	Description	Type	Example
ArrDep	flight direction (arrival or departure)	Categorical	A or D
Airline	airline unique company code	Categorical	AF
AircraftType	aircraft type code	Categorical	77W
Destination	IATA code of the destination airport	Categorical	JFK
Season	The IATA Season	Categorical	W
Week	A week index calculated from a reference date.	Numeric	1000
Day	Day number in the week (1-7)	Numeric	1
SOBT	The Scheduled off-block time in minutes since midnight	Timestamp	360
Service Type	Transport category	Categorical	J

Table 6.1: Basic flight features (BFF)

6.3.2 PRM count features

Before the departure or arrival of a flight, passengers with disabilities confirm their care. We use the information known 36 and 24 hours in advance;

Feature	Description	Type	Example
PrmConfirmed36h	Number of PRM confirmed 36h before the flight	Numerical	2
PrmConfirmed24h	Number of PRM confirmed 24h before the flight	Numerical	2

Table 6.2: PRM count features

6.4 Model configuration

As explained above, we have used a technology that can be easily deployed in the Paris Airports system based on **FastTree** from the **C# ML.net** library. The objective was: 1. To evaluate if our forecasts are more accurate than those from providers or the PRM manager tool. 2. To promptly implement this solution in a production environment if the results are favorable.

We have found a good configuration for this model based on three main model parameters: the number of trees, leaves, and the learning rate. These hyperparameters are noted as *#Tree/#Leaves/LearningRate* and are 75/700/0.2.

6.5 Correlation Analysis

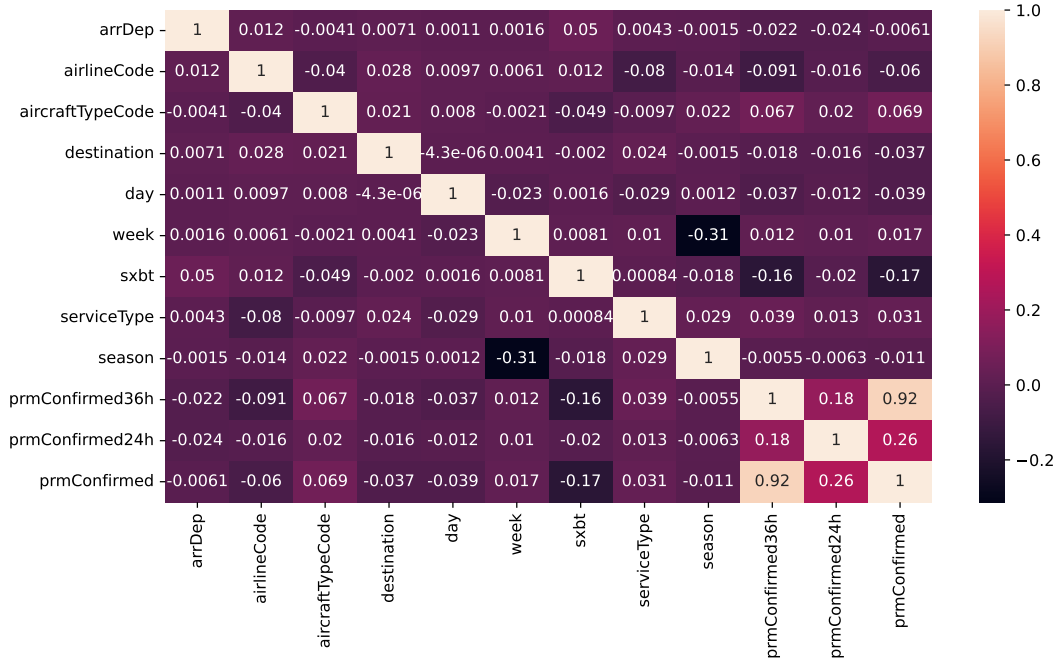


Figure 6.9: Results of statistical correlation measure for CDG.

Figure 6.9 and 6.10 show two heatmaps of the Pearson correlation score for CDG Airport and Orly Airport, respectively. These heatmaps showcase the correlation of various attributes, two attributes, namely **prmConfirmed36h** and **prmConfirmed24h**, stand out with a notably high correlation score with the target variable **prmConfirmed**. This elevated correlation is logical, given that both attributes represent the number of PRMs that have confirmed their care within respective time frames of 36 and 24 hours. On the other hand, attributes related to flight specifics exhibit a relatively low Pearson correlation score, underscoring a minimal correlation with the **prmConfirmed** target variable.

6.6 Results

This section delves into a comparative analysis of our **FastTree** model against other methods from ADP. The first method we have considered is to use the values known 36 (noted *PRM*

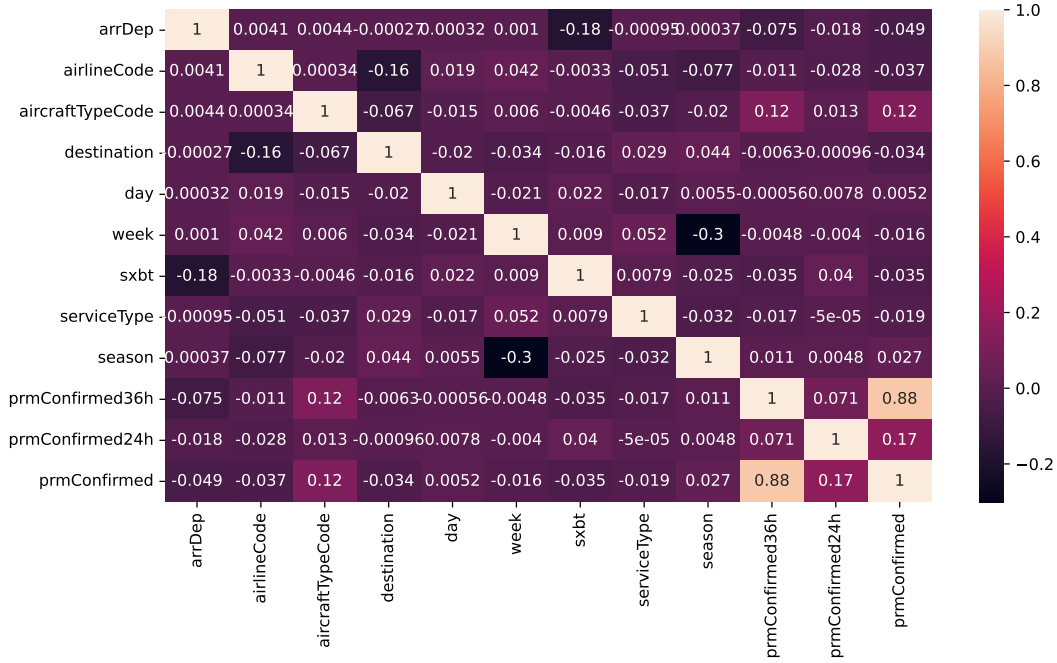


Figure 6.10: Results of statistical correlation measure for Orly

confirmed 36h) or 24 (noted *PRM confirmed 24h*) hours before the flight as a *prediction*. The second method, in contrast, leverages the predicted value generated by the *PRM Manager tool*, again with a time frame of 36 (noted *PRM manager 36h*) or 24 (noted *PRM manager 24h*) hours before the flight. As the PRM manager is a proprietary tool, we have no details on how it calculates the value it produces. We compare our model over 28 days from August 18th, 2023, to September 15th, 2023.

Figure 6.11 (resp. 6.12) shows the results for each model for CDG Airport (resp. Orly Airport). We can see that keeping the values of confirmed PRM, even 24 hours in advance, is not viable for obtaining a good prediction. We can also observe the similarity between the predicted PRM curve and the actual curve, demonstrating the quality of our prediction and the value of using a machine-learning model for this task. We also outperform the predictions made by the proprietary tool.

These results are confirmed in Tables 6.3 and 6.4, which present the error (see. Definition 34) between the real value and each model. Although our approach (column. $\epsilon_{\hat{y}}$) tends to over-approximate, the error is much smaller than other models.

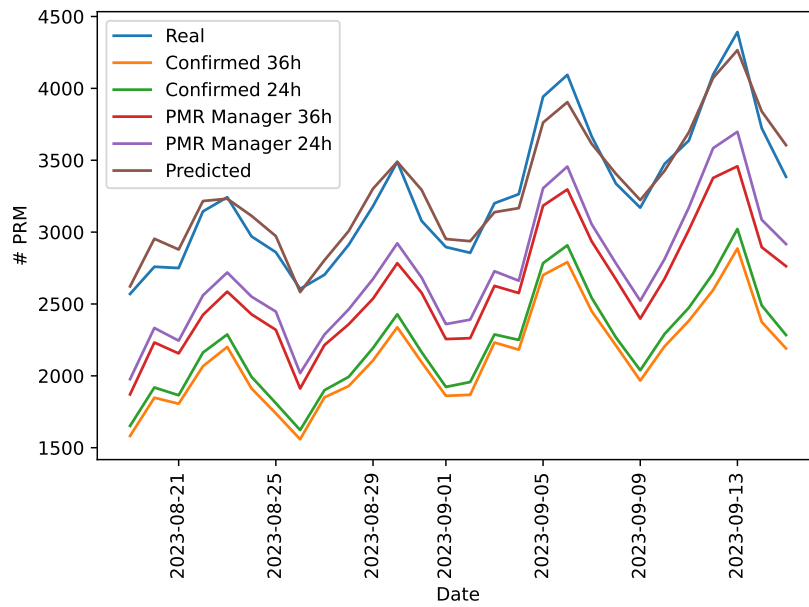


Figure 6.11: Comparison between actual presentation, estimated presentations, and presentations curves known in advance for CDG Airport.

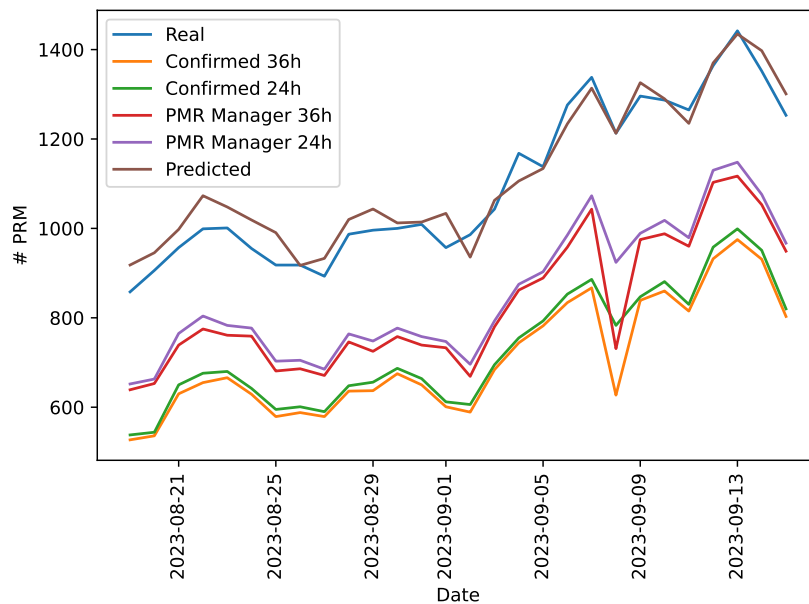


Figure 6.12: Comparison between actual, estimated, and presentations curves known in advance for Orly Airport.

date	$\epsilon_{\hat{y}}$	ϵ_{PMR36}	ϵ_{PMR24}	ϵ_{CONF36}	ϵ_{CONF24}
2023-08-19	-52.4	699.0	593.0	988.0	918.0
2023-08-20	-194.7	527.0	426.0	911.0	840.0
2023-08-21	-129.0	594.0	505.0	944.0	885.0
2023-08-22	-72.2	720.0	584.0	1077.0	982.0
2023-08-23	10.7	657.0	524.0	1042.0	955.0
2023-08-24	-143.2	542.0	419.0	1058.0	977.0
2023-08-25	-112.3	540.0	413.0	1121.0	1050.0
2023-08-26	22.5	693.0	586.0	1046.0	982.0
2023-08-27	-101.1	489.0	416.0	853.0	803.0
2023-08-28	-95.8	553.0	449.0	983.0	919.0
2023-08-29	-121.2	644.0	507.0	1074.0	987.0
2023-08-30	0.5	705.0	567.0	1151.0	1061.0
2023-08-31	-215.7	500.0	394.0	983.0	913.0
2023-09-01	-56.0	640.0	536.0	1035.0	973.0
2023-09-02	-80.8	594.0	465.0	988.0	899.0
2023-09-03	62.3	575.0	473.0	969.0	913.0
2023-09-04	97.0	688.0	602.0	1082.0	1014.0
2023-09-05	179.8	758.0	637.0	1241.0	1158.0
2023-09-06	189.9	797.0	638.0	1303.0	1186.0
2023-09-07	51.8	730.0	611.0	1216.0	1123.0
2023-09-08	-68.3	666.0	555.0	1126.0	1070.0
2023-09-09	-52.9	772.0	647.0	1203.0	1131.0
2023-09-10	47.6	799.0	662.0	1269.0	1183.0
2023-09-11	-57.2	622.0	466.0	1254.0	1163.0
2023-09-12	23.4	719.0	512.0	1499.0	1383.0
2023-09-13	126.3	934.0	694.0	1506.0	1370.0
2023-09-14	-117.1	828.0	637.0	1348.0	1233.0
2023-09-15	-220.0	622.0	468.0	1194.0	1101.0

Table 6.3: Daily error between actual presentation, estimated presentations, and presentations known in advance for CDG.

date	$\epsilon_{\hat{y}}$	ϵ_{PMR36}	ϵ_{PMR24}	ϵ_{CONF36}	ϵ_{CONF24}
2023-08-19	-60.1	219.0	206.0	331.0	320.0
2023-08-20	-39.5	253.0	243.0	370.0	362.0
2023-08-21	-40.8	218.0	192.0	327.0	307.0
2023-08-22	-74.1	224.0	195.0	344.0	323.0
2023-08-23	-46.9	240.0	218.0	335.0	321.0
2023-08-24	-63.8	196.0	178.0	326.0	313.0
2023-08-25	-72.5	237.0	215.0	339.0	323.0
2023-08-26	0.8	232.0	213.0	330.0	317.0
2023-08-27	-39.9	222.0	208.0	314.0	303.0
2023-08-28	-32.9	241.0	223.0	351.0	339.0
2023-08-29	-47.4	271.0	248.0	359.0	340.0
2023-08-30	-12.2	242.0	223.0	325.0	313.0
2023-08-31	-5.2	270.0	251.0	359.0	345.0
2023-09-01	-76.6	224.0	210.0	356.0	345.0
2023-09-02	50.3	317.0	290.0	397.0	380.0
2023-09-03	-19.8	263.0	250.0	359.0	348.0
2023-09-04	62.0	306.0	293.0	424.0	413.0
2023-09-05	4.1	249.0	235.0	356.0	345.0
2023-09-06	42.1	318.0	291.0	442.0	423.0
2023-09-07	24.3	295.0	265.0	471.0	452.0
2023-09-08	0.2	482.0	289.0	586.0	430.0
2023-09-09	-30.0	321.0	307.0	457.0	449.0
2023-09-10	-2.8	299.0	269.0	427.0	406.0
2023-09-11	30.1	305.0	286.0	450.0	435.0
2023-09-12	-6.1	261.0	234.0	432.0	406.0
2023-09-13	7.1	325.0	294.0	467.0	443.0
2023-09-14	-45.5	299.0	276.0	421.0	401.0
2023-09-15	-47.9	304.0	286.0	450.0	433.0

Table 6.4: Daily error between actual presentation, estimated presentations, and presentations known in advance for Orly.

6.7 Conclusion

Throughout this chapter, the significance of accurately predicting the number of passengers with reduced mobility in airports was underscored, primarily emphasizing the operational efficiencies and the overarching goal of fostering inclusivity in the aviation industry. With their bustling nature and vast numbers of passengers, Paris Airports face the critical task of ensuring seamless travel for PRM. Hence, the main objective was to predict these numbers with greater accuracy.

By examining the historical data and leveraging machine learning techniques, particularly the FastTree model, we endeavored to optimize these predictions. Our exploratory data analysis revealed key trends and patterns that informed our modeling approach. Additionally, the feature engineering process, distinguishing between Basic flight features and PRM count features, enabled a comprehensive understanding of the factors impacting PRM numbers.

Our comparative analysis against existing methods underscored the value proposition of our machine-learning model. Our model surpassed the predictions made by existing tools and methods. These findings highlight the potential of harnessing advanced technologies and machine learning for airport operational enhancements.

Chapter 7

Predicting and explaining off-block delays

Contents

7.1	Milestones before off-block	162
7.2	Exploratory Data Analysis - Off-block delays at CDG	163
7.3	Problem statement and objectives	165
7.3.1	Real-time off-block delay prediction	165
7.3.2	Off-block delay forecast	165
7.4	Model definition	165
7.4.1	Basic Flight Features (BFF)	165
7.4.2	Off-block Milestone Features (OMF)	166
7.4.3	Previous and Current Flights Delay Features (PCFDF)	167
7.4.4	Weather condition features (WCF)	167
7.4.5	Passenger Flow Features (PFF)	168
7.5	Data and feature extraction, preprocessing, and selection	168
7.5.1	Data extraction and preprocessing	168
7.5.2	Feature selection	168
7.6	Model selection	170
7.6.1	Gradient-boosted decision trees	171
7.6.2	Recurrent neural networks	172
7.6.3	Baseline model	172
7.7	Model evaluation	172
7.7.1	Forecast	172
7.7.2	Real-time predictions	175
7.8	Explaining off-block delay predictions	181
7.9	Deployment	184
7.10	Conclusions	184

At Paris-CDG, there are more than one flight departure per minute. There are about 498,000 aircraft movements per year, corresponding to about 1,300 movements per day²³. On average, there are 145 passengers per flight. At Paris-CDG, resources are planned using solutions powered by constraint programming solvers (see Chapters 3 and 4.2.3). Among the critical resources are the stands assigned to the flights. When a stand is released late, this can lead to complex scheduling changes and cascading delays. It is, therefore, essential to anticipate and predict these delays as accurately as possible, explain them, and propose actions to limit them. Recall that a *rotation* is a set composed of arrival and departure flights. This set generally consists of two flights, but there may be only one, in which case the flight begins a new rotation. The rotation period is the time between the arrival of an aircraft (landing) and its departure (takeoff). A flight has a scheduled departure time called **Scheduled Off-Block Time**, where it is supposed to leave its parking area. The moment when the flight leaves its parking position is called **Actual Off-Block Time**. The delay is the period between **AOBT** and **SOBT**.

7.1 Milestones before off-block

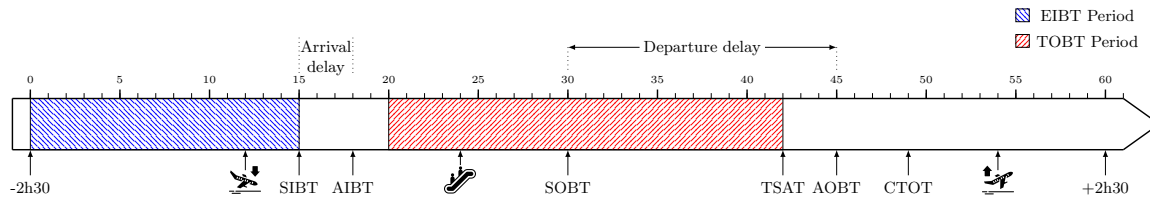


Figure 7.1: Milestones of a flight at Paris-CDG airport

We present here the main milestones preceding the pushback of an aircraft. These milestone stands are presented in Figure 7.1. Before the flight arrives at the airport, the flight estimates its arrival time at the block: **Estimated In-Block Time**. The flight refines its estimation during its journey, which is why we obtain not one but several **EIBT** materialized by the blue zone on the Figure 7.1. The **Actual In-Block Time** is the right time when the flight arrives at its stand. We call *arrival delay* the difference between AIBT and SIBT. The boarding starts between 1 hour and 30 minutes before **SOBT**. During the turn-around period (or rotation), the airline sends various estimated times that the aircraft is ready (**Target Off-Block Time**) to the management system (materialized by the red zone in Figure 7.1). For heavy traffic in the sky or congestion on the runway, air traffic control can “slot” a flight, i.e., force its takeoff between a calculated time called **Calculated Take-Off Time** and CTOT plus 15 minutes. If the aircraft fails to take off during this period, it can be slotted again. The last milestone is the **Target Start-Up Approval Time** is the time provided by air control, taking into account **TOBT**, **CTOT**, and/or the traffic situation that an aircraft can expect startup/pushback approval.

²³<https://www.parisaeroport.fr/docs/default-source/groupe-fichiers/finance/rerelations-investisseurs/information-financi%C3%A8re/rapports-annuels/report-on-activity-and-sustainable-development-2019.pdf#page=38>

7.2 Exploratory Data Analysis - Off-block delays at CDG

Punctuality is a sensitive issue in large airports and hubs for passenger experience and controlling costs at the airport level. Paris-CDG is ranked²⁴ in 2018 in 10th place in terms of punctuality. Around half of the flights arrive on time, but only 20% take off on time.

A study of delays at Paris-CDG has highlighted different causes (e.g., extreme weather conditions, congestion, breakdowns, incidents at the airport, passenger processes, etc.) of these delays at different phases (parking/pushback, taxi-ing, etc.). Figure 7.2 gives an overview of off-block delays over the year considered in our study (from August 12th 2021 to September 21st, 2022). It should be noted that during this period, some terminals were closed and are still closed, while others reopened. Over this period, the proportion of flights with off-block delays is 82% (Figure 7.2a). Figure 7.2b shows the cumulated delays (in minutes) for each day of the considered period. We can observe an increase during summer 2022, corresponding to a resumption of traffic and some terminals' reopening. Figure 7.2c shows the delay mean for each terminal. Terminal 2E has an average delay of 37% above the airport average. Figure 7.2d shows the number of delayed flights per terminal.

We can observe that the two terminals corresponding to Air France (2E and 2F) have the most delayed flights with 30% and 35% of delayed flights, respectively. Finally, Figure 7.2h depicts the number of delayed flights per day period, ranging from $p1$ (6 am - 8 am) to $p6$ (8 pm - 11 pm). We can observe that the morning periods, particularly $p2$ (9 am - 11 am) and $p3$ (11 am - 2 pm), accumulate most of the delayed flights with 25% and 23% respectively. Since most delays occur in the morning and there is a cascading delay effect, it is crucial to predict and manage these delays in these time slots accurately.

Wake Category	% Flights	Average delay
light	0.124481	14.189239
medium	0.630896	18.203765
heavy	0.244623	27.333156

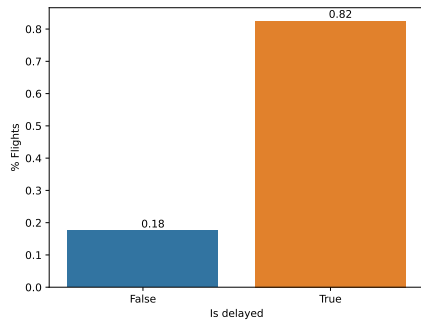
Table 7.1: Average delay per wake category.

Day period	% Flights	Average delay
p1	0.118049	11.882946
p2	0.237615	22.400423
p3	0.216767	23.048363
p4	0.171058	22.116601
p5	0.156253	17.804391
p6	0.100258	16.462603

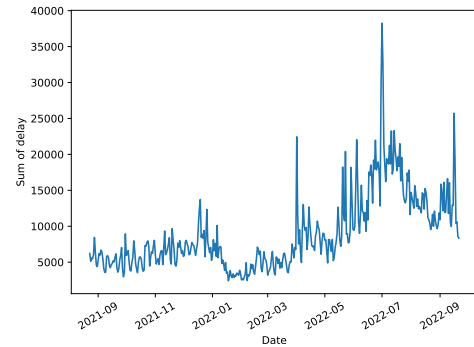
Table 7.2: Average delay per day period.

Table 7.1 presents some information about wake categories. For each type, we had the percentage of flights and the average delay for this category. Table 7.2 presents the same information for the day periods.

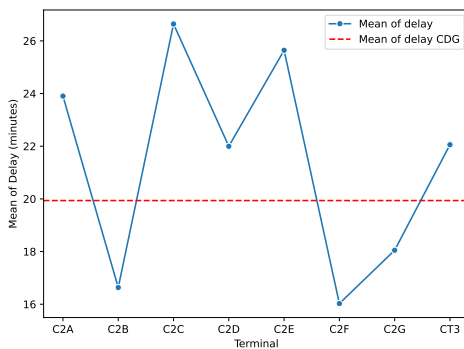
²⁴According to OAG Flightview.



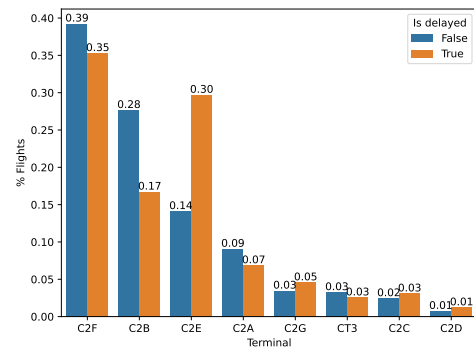
(a) Proportion of on-time flights vs. Off-Block delayed flights.



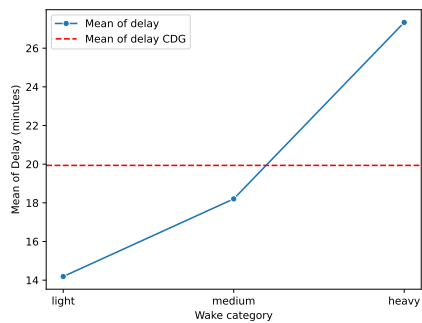
(b) Cumulative delay for each day of the considered period.



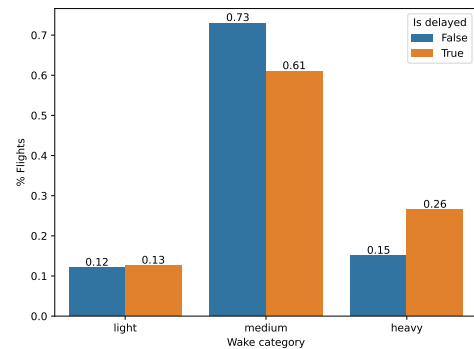
(c) Average delay per terminal.



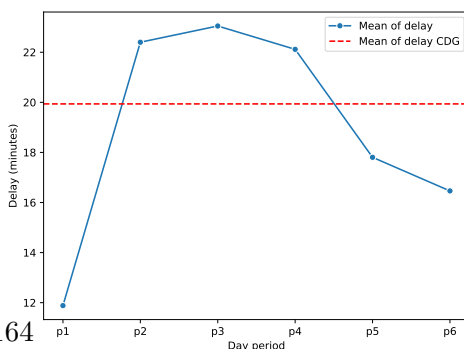
(d) Proportion of on-time flights vs. Off-Block delayed flights per terminal.



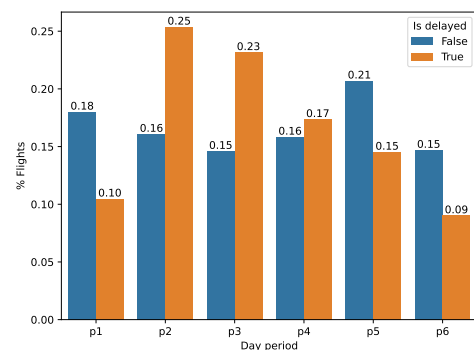
(e) Average delay by wake category.



(f) Proportion of on-time vs. Off-Block delayed flights per wake category.



(g) Average delay by day period.



(h) Proportion of on-time flights vs. Off-Block delayed flights per day period.

7.3 Problem statement and objectives

This section presents the problem and the objectives of our off-block delay prediction approach at Paris-CDG. Recall that we call off-block delay the time separating the moment that the aircraft leaves the boarding gate (this operation is called *pushback*) from the scheduled off-block time (between [AOBT](#) and [SOBT](#)).

Let Y be the target variable to predict for an input sample describing the flight under study. We distinguish two regression tasks: the real-time off-block delay prediction and the forecast of off-block delay prediction.

7.3.1 Real-time off-block delay prediction

The problem considered here is the one of predicting at any time t , the off-block delay y (expressed in minutes) that this flight will indeed have. These predictions are updated every 5 minutes until the flight leaves its stand. Indeed, for every slot (each slot lasts 5 minutes), we can acquire new relevant data from the Paris-CDG operational information system. These data can be used to update the predictions (this is valid for dynamic variables such as weather conditions, the progress of passenger processes, etc.). Real-time off-block delay predictions are intended mainly to:

- Draw managers' attention in real-time to flights likely to have significant delays and which may have cascading consequences;
- Explain and identify actionable causes, if necessary, to resolve the situation.

7.3.2 Off-block delay forecast

We call forecasts the prediction of delays before the opening of the flights. This may be a few hours or several days before the flight. In our case, the forecast cannot rely on dynamic information on the progress of passenger processes, weather conditions, etc. A such forecast may serve to:

- Establish several plans and mitigation measures according to the expected delays. Then, managers can activate the plan provided for each situation when it occurs;
- Identify the causes and anticipate the chaining effect and the consequences;
- Use forecasts to make plausible simulations on congestion and queues according to delay forecasts, then consider solutions and management plans.

7.4 Model definition

In this section, we present and motivate the essential information currently available in the Paris-CDG operational information system, which is likely relevant for predicting off-block delays. The proposed features come from the analysis of a recent report on takeoff delays at Paris-CDG and an entire year of real data.

7.4.1 Basic Flight Features (BFF)

Like the previous chapter, we used the basic characteristics of a flight, and they do not change over time (these features are listed in Table. [7.3](#)). For example, the name of the airline operating the flight, the aircraft type, the IATA code of the flight's destination, the terminal, the type of customs (national, Schengen, or international), and the IATA season (Summer or Winter).

Feature	Description	Type	Example
Airline	Airline unique company code	Categorical	AF
AircraftType	Aircraft type code	Categorical	77W
Destination	IATA code of the destination airport	Categorical	JFK
Terminal	CDG Terminal	Categorical	C2E
Customs	Customs Criteria	Categorical	Schengen
Season	The IATA Season	Categorical	W
Week	A week index calculated from a reference date.	Numeric	1000
Day	Day number in the week (1-7)	Numeric	1
Bus	True if the flight has bus access.	Boolean	True
Parking	True if the arrival flight and the departure flight have the same parking	Boolean	False
SOBT	The Scheduled off-block time in minutes since midnight	Timestamp	360
Rotation	Duration between landing and takeoff in minutes	Numeric	300
Pax Count	Number of passengers on the flight (estimated)	Numeric	140
Total Pif Passenger	Number of passengers that must pass through the security point (estimated)	Numeric	75
Service Type	Transport category	Categorical	J

Table 7.3: Basic flight features (BFF)

7.4.2 Off-block Milestone Features (OMF)

These milestones correspond to each off-block and are managed by various stakeholders, such as air traffic control, the airline, and the airport. For this study, we focused on milestones around the [SOBT](#) (-2h30 to +2h30, see Figure. 7.1). This period is split into 60 slots of 5 minutes. Each new milestone (EIBT, TOBT, CTOT, [TSAT](#)) has a timestamp. The OMF features are listed in Table. 7.4.

Feature	Description	Type
Arrival Delay	Time in minutes between the last EIBT and the SIBT.	Numeric
$TOBT_{diff}$	Time in minutes between the last TOBT and the SOBT .	Numeric
$TOBT_{count}$	The number of TOBTs	Numeric
$CTOT_{diff}$	Time in minutes between the last CTOT and the SOBT .	Numeric
$TSAT_{diff}$	Time in minutes between the last TSAT and the SOBT .	Numeric

Table 7.4: Off-block milestone features (OMF)

7.4.3 Previous and Current Flights Delay Features (PCFDF)

Relevant information on the probability of a delay for a given flight is the proportion of flights scheduled just before the flight under study and which are late. This could, for example, be due to congestion, a breakdown at the airport, or extreme weather conditions.

Feature	Description	Type
Delay Airport	Mean off-block delay from all the airports (regardless of the terminal)	Numeric
Delay Terminal	Mean off-block delay from the same terminal	Numeric
Delay airline	Mean off-block delay from the same airline	Numeric
Percent delayed flights Airport	Proportion of off-block delayed flights over all the airport (regardless of the terminal)	Numeric
Percent delayed flights Terminal	Proportion of off-block delayed flights from the same terminal	Numeric
Percent delayed flights Airline	Proportion of off-block delayed flights from the same airline	Numeric

Table 7.5: Previous and Current Flights Delay Features (PCFDF)

We compute the proportion of late flights and the average duration of these delays for each flight slot. At a time t , these features are calculated during a time window w , ranging from a few minutes to a few hours. As we will show empirically, the optimal window duration is a few minutes. For example, if $w = 10$ minutes and the current flight slot is 25, we compute the different values from the flights whose AOBT is between slot 23 and slot 25.

7.4.4 Weather condition features (WCF)

Certain weather conditions, such as low visibility and strong winds, are known to be factors that can cause takeoff delays and, therefore, delay the flight's departure from its stand. Table. 7.6 shows weather-related features.

Feature	Description	Type
Low Visibility Procedures	These procedures are applied at an airport to ensure safe operations when there is low visibility.	Boolean
Humidity rate (in percent)	Humidity rate	Numeric
Wind speed (in meter/sec)	Wind speed	Numeric
Air pressure (in hectoPascal)	Air pressure	Numeric
Temperature (in degrees Celsius)	Temperature	Numeric

Table 7.6: Weather Condition Features (WCF)

7.4.5 Passenger Flow Features (PFF)

Feature	Description	Type
Security Checkpoint Progression	The progression of the number of passengers having passed the security point.	Numeric
Boarding Progression	The progression of the boarding.	Numeric

Table 7.7: Passenger flow features (PFF)

These features provide information at any time on the progress of specific passenger processes, which may cause an off-block delay. In particular, the relevant information is the percentage at slot t of passengers who have already passed boarding or passed security checkpoints. These features are used only to predict and update the off-block delay prediction in real-time (each flight slot).

7.5 Data and feature extraction, preprocessing, and selection

In this section, we present our main findings concerning the selection of variables and the selection of data (in particular, the choice of the best parameters for the time window duration, the training data amount, ...).

7.5.1 Data extraction and preprocessing

Paris-CDG's operational information system (called AOP for Airport Operation Plan) collects much information about each flight and its progress. A new flight entry is created for our prediction tasks with the associated timestamp and updated data at each time slot. It is possible to trace the status of a flight back to its departure. Therefore, for static characteristics, we extract them only once. For dynamic characteristics, such as delays of other flights, these are calculated variables that we perform with queries on past flights. For example, to compute the proportion of flights that have been delayed in the last w minutes, we need to review all flights involved in the w time window. The delay characteristics of previous and current flights (PCFDF) are computed after extraction with different w windows. Our study considered one year of data (from August 2021 to September 2022). We constructed a dataset with 10,633,920 rows and 31 columns (each flight is repeated 60 times with dynamic values updated in each slot).

7.5.2 Feature selection

Once our data set was extracted and preprocessed we proceeded to the selection of variables to confirm our intuitions and to eliminate attributes that would prove irrelevant to our prediction tasks. We first performed a simple correlation analysis between each characteristic and the target variable (delay to parking departure). We use the well-known **Pearson correlation coefficient** that measures a linear correlation. It is a number between -1 and 1 the strength and direction of the relationship between two variables. Figures. 7.3 and 7.4 show the results of this coefficient.

It can be seen in Figure. 7.3 that the most relevant variables at the slot 0 are:

- the difference between the **SOBT** and the **TSAT** with a score of 0.4,

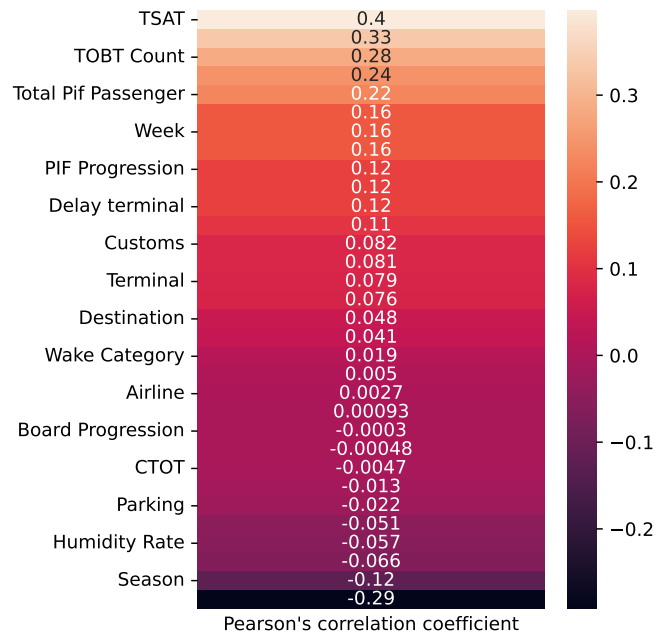


Figure 7.3: Results of statistical correlation measure at slot 0

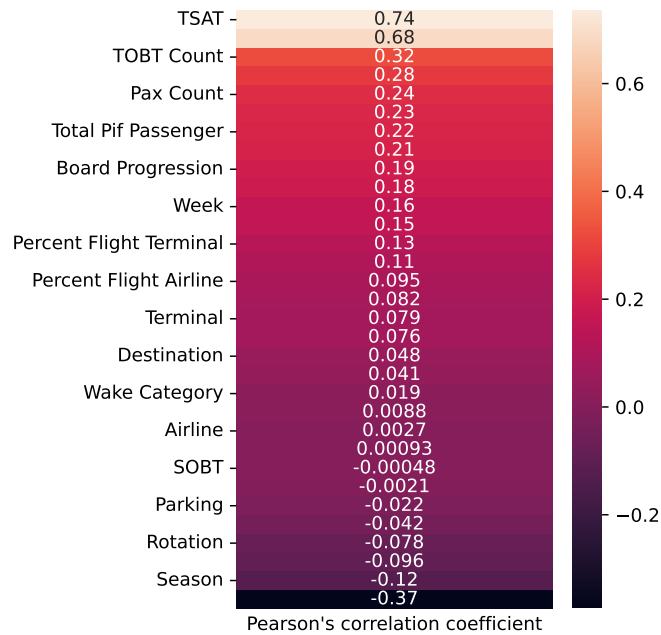


Figure 7.4: Results of statistical correlation measure at slot 30

- the number of TOBT with a *Pearson* score of 0.33.

In contrast, the variables representing the humidity rate (*Humidity Rate*) and IATA season of a flight (*Season*) seem to have little relevance. These variables have a negative *Pearson* score of -0.057 and -0.12 , respectively. For the slot 30, the order of the essential variables is confirmed.

The durations between **SOBT** and **TSAT** or the number of **TOBT** are the essential variables with a *Pearson* score of 0.74 and 0.32.

The importance of the **TSAT** variable is explained by the fact that it is one of the last pieces of information obtained for a flight before it leaves its stand and that departure is most often at the time indicated by the **TSAT**. Nevertheless, air traffic control may send this information very early before the departure of the aircraft. The variable *Percent Flight Airport*, representing the proportion of delayed flights airport (for the calculation of the values, we used the 10-minute time window), becomes more important with a progression of its score from 0.16 to 0.23. Finally, we can note that the dynamic variables (in particular, the variable concerning the progression of the boarding) have a score in progression. This progression shows the importance of using dynamic flight data for real-time predictions.

To validate the findings on the correlation scores obtained, we performed another empirical analysis by varying the set of variables used for delay prediction. We also varied the window w used to calculate the dynamic variables from the nearby history and the amount of history to use (twelve months, nine months, six months, or three months). We used an ensemble model of boosted regression trees called **LightGBM**[KMF⁺17].

For our study, we tested our configuration for 40 days (from August 12 to September 21, 2022). In the following, we denote by $\mathcal{D}_w^m$ the dataset with m the number of months used and w the duration of the window used for the computation of the dynamic variables. The set of datasets are defined as follows:

$$\left\{ \mathcal{D}_w^m \mid \begin{array}{l} m \in \{3, 6, 9, 12\} \\ w \in \{10, 30, 60\} \end{array} \right\} \quad (\mathcal{I}_{12})$$

Concerning, models we have tested different configurations defined as follows:

$$\left\{ \text{FastTree}_{v}^{t/l/lr} \mid \begin{array}{l} t/l/lr \in \{32, 75\} \times \{64, 128, 256, 512\} \times \{0.05\} \\ v \in \mathcal{V} \end{array} \right\} \quad (M_1)$$

where t, l, lr represents the hyperparameters and correspond to $\#Tree/\#Leaves/LearningRate$, and $\mathcal{V}$ corresponds to the set of features used by the model:

$$\mathcal{V} = \{\{BFF\}, \{BFF, WCF, OMF, PFF\}, \{BFF, PCFDF, WCF, OMF, PFF\}\}$$

For each day, we trained the model until the day before the test day and evaluated it on the test day. Table. 7.8 shows the optimal configurations. The errors and R^2 score presented in this Table are for the 40 days tested.

Table. 7.8 presents the results for the different configurations. Using dynamic variables brings a real gain, reducing the error by half and significantly improving the R^2 score. Thus, the best configuration uses all of the history (12 months), with a time window of 60 minutes.

7.6 Model selection

As with the prediction task in the previous chapter, the Paris Airports are the end-users of our approach and are already positioned on technologies based on *decision trees*, particularly **FastTrees** from **ML.net** library.

This led us to consider in this case study **LightGBM**, which is similar to **FastTrees** as they are an *ensemble of boosted decision trees*. Nevertheless, we have also considered deep learning techniques since these latter are among the best for regression problems in many areas. In the following section, we provide some details on the model selection.

Dataset	Features ($\mathcal{V}$)	Hyperparameters	MAE	RMSE	R^2
$\mathcal{D}_{60}^{12}$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.645	13.858	0.731
$\mathcal{D}_{60}^9$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.672	13.858	0.731
$\mathcal{D}_{30}^{12}$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.705	13.928	0.728
$\mathcal{D}_{10}^{12}$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.706	13.926	0.728
$\mathcal{D}_{10}^{12}$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.709	14.009	0.725
$\mathcal{D}_{60}^6$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.720	14.044	0.723
$\mathcal{D}_{30}^{12}$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.730	14.047	0.723
$\mathcal{D}_{30}^9$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.737	14.038	0.723
$\mathcal{D}_{60}^{12}$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.746	14.075	0.722
$\mathcal{D}_{10}^9$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.762	14.048	0.723
$\mathcal{D}_{10}^6$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.787	14.224	0.716
$\mathcal{D}_{30}^9$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.792	14.173	0.718
$\mathcal{D}_{60}^9$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.796	14.214	0.717
$\mathcal{D}_{30}^6$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.801	14.221	0.716
$\mathcal{D}_{10}^9$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.803	14.184	0.718
$\mathcal{D}_{60}^6$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.823	14.318	0.712
$\mathcal{D}_{60}^3$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.831	14.309	0.713
$\mathcal{D}_{10}^3$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.866	14.386	0.710
$\mathcal{D}_{10}^6$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.866	14.412	0.709
$\mathcal{D}_{30}^6$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.871	14.440	0.707
$\mathcal{D}_{30}^3$	<i>BFF, PCFDF, WCF, OMF, PFF</i>	75/256/0.05	9.897	14.388	0.710
$\mathcal{D}_{30}^3$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.912	14.543	0.703
$\mathcal{D}_{60}^3$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.915	14.491	0.705
$\mathcal{D}_{10}^3$	<i>BFF, WCF, OMF, PFF</i>	75/256/0.05	9.919	14.479	0.706
$\mathcal{D}_{60}^9$	<i>BFF</i>	75/256/0.05	17.443	26.576	0.010
$\mathcal{D}_{10}^{12}$	<i>BFF</i>	75/256/0.05	17.456	26.667	0.004

Table 7.8: Results of feature selection on historical data.

7.6.1 Gradient-boosted decision trees

Ensemble methods are machine learning techniques that build a strong learner from many weak learners. Boosting is a well-known ensemble method that consists of iteratively training a sequence of weak learners and using the error of the previous learners for weighting the sample for the next learner. **GBDT** is a popular method for solving prediction problems in classification and regression domains. **XGBoost** [CG16] and **LightGBM**[KMF⁺17] are two classical implementations of GBDT. **LightGBM** introduces Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) to deal with datasets with many features and many examples. We use a **LightGBM** model for evaluation and experiments in the following. We choose **LightGBM** for this accuracy and this compatibility with the library for computing SHAP values [LL17], or other forms of explanations, because there exists an implementation in Python and C#. To facilitate the experiments, we have developed a command-line interface to configure a *LightGBM* model easily. Thus, we can choose the columns to include or ignore and identify the categorical and numerical columns with the command. It is also possible from the command line to configure the model (specifying the number of *leaves*, *trees* or the *learning-rate* value, for example).

Hyperparameters	value
Stacked RNNs layers	256-tanh-0 $\rightarrow$ 256-tanh-0
Stacked Dense layers	64-relu-0.6
Optimizer	Adam
Learning Rate	0.0001

Table 7.9: Hyperparameters for the LSTM-model.

7.6.2 Recurrent neural networks

[RNN](#) is a well-known technique based on neural networks. For our study, we used [LSTM](#) [[HS97](#)] (*Long Short Time Memory*) a classical implementation of RNN. [LSTM](#) is useful for ML tasks with dependencies on recent history, such as speech-to-text, translation, time series, etc. We have tested a simple configuration on [LSTM](#) (see [Table 7.9](#)) for our dataset.

Because the model has a very long training time, we had to limit the dataset used to windows of 3 months only. The number of `epochs` is also limited to 10. The models have been run on a cluster of computers equipped with 768 GB of RAM and five 80-core Intel Xeon Gold 6248 (2.5 GHz). We have set the wall-clock time limit to 96 hours and the memory limit to 760 GB. The model took more than 5 hours to perform a single iteration on the data, plus the error of the model was quite close to the [LightGBM](#) model, so we decided to renounce the use of this model for our study.

7.6.3 Baseline model

We also used a `baseline model` to get an idea of the predictions' quality and the errors' magnitude. This latter is a naive model that always predicts the mean of the delay (computed on the training data).

7.7 Model evaluation

In this section, we answer the question of the choice of model type: a standard regression model or a model taking into account the history of predictions to consider, for example, the chaining effect of delays. We provide results with an ensemble model of boosted regression trees and an [LSTM](#)-based model. Note that the objective is not to find the best model or to tune a model to find its best parameters; we first want to know:

- how much an ML model could significantly reduce the prediction error compared to a baseline model (that predicts the average delay)
- whether a sequential model significantly improves the results compared to a flat model.

7.7.1 Forecast

We first study the result for the off-block delay forecast as explained in [Section 7.3.2](#). Recall that the off-block delay forecast gives a prediction a few days or hours before the flight opens (1 day before the flight in our experiments).

For this task, we use only one dataset of [\$\mathcal{I}_{12}\$](#) that take account 12 months of historical data defined as follows:

$$\{\mathcal{D}^{12}\} \quad (\mathcal{I}_{13})$$

Note that we have omitted the m parameter as we will not be using dynamic variables. Concerning the model, we use a configuration from M_1 and the baseline model:

$$\{\text{FastTree}_{BFF}^{75,256,0.05}, \text{Baseline}\} \quad (M_2)$$

Summary of the experiments 13 (Forecast - , ,)

	$\mathcal{I}_{13}$
	M_2
	2 Hours
	350 GB
	Linux CentOS Stream 8.3
	80 Intel Xeon Gold 6248 (2.5 GHz)
	768 GB

For each day, we trained the model until the day before the test day and evaluated it on the test day. Table 7.10 presents, for each day, the mean of delay, the maximum delay, the standard deviation, the R^2 score, and finally, the errors between our model or baseline model and the target variable. Despite the superiority of the **LightGBM** model, we can observe the difficulty of making very accurate off-block delay forecasts.

Day	$\bar{y}$	y^{max}	σ	R^2	RMSE	RMSE _{Baseline}	MAE	MAE _{Baseline}
2022-08-13	24.83	266	29.39	0.11	27.79	29.53	17.41	18.66
2022-08-14	24.81	247	26.41	0.01	26.21	26.56	17.52	18.47
2022-08-15	21.16	186	24.48	0.11	23.09	24.49	16.47	17.31
2022-08-16	24.79	260	30.67	0.12	28.84	30.80	18.01	19.94
2022-08-17	27.00	255	27.03	0.09	25.77	27.50	17.83	18.74
2022-08-18	24.07	219	24.72	0.16	22.62	24.81	15.39	16.87
2022-08-19	19.34	239	22.49	0.09	21.42	22.64	16.07	16.85
2022-08-20	18.81	293	22.56	0.08	21.64	22.77	16.14	17.18
2022-08-21	16.86	281	22.72	0.03	22.42	23.29	15.65	16.82
2022-08-22	15.25	210	22.75	-0.02	22.98	23.72	16.15	16.93
2022-08-23	16.76	236	21.32	0.08	20.40	21.94	14.59	17.17
2022-08-24	19.57	238	26.45	-0.09	27.62	26.56	16.62	18.89
2022-08-25	17.88	156	22.68	0.08	21.76	23.04	15.15	17.17
2022-08-26	17.79	259	21.86	0.09	20.84	22.26	14.46	16.84
2022-08-27	17.74	179	22.88	0.06	22.13	23.27	14.49	16.40
2022-08-28	16.97	295	21.95	0.09	21.00	22.51	14.51	16.89
2022-08-29	15.50	295	21.73	-0.36	25.30	22.67	15.67	17.46
2022-08-30	18.27	236	23.89	0.08	22.95	24.18	14.60	17.38
2022-08-31	21.04	217	25.84	0.09	24.67	25.85	15.66	18.19
2022-09-01	18.91	290	24.55	0.11	23.12	24.74	14.36	17.36
2022-09-02	27.55	190	25.45	0.08	24.40	26.05	16.10	18.10
2022-09-03	26.03	209	27.03	0.18	24.42	27.33	15.79	18.56
2022-09-04	21.03	281	26.98	0.14	25.03	27.00	16.26	18.19
2022-09-05	25.38	208	26.42	0.03	26.01	26.64	17.22	18.34
2022-09-06	21.93	224	26.92	-0.04	27.40	26.92	18.13	18.66
2022-09-07	21.42	277	26.43	0.19	23.85	26.43	16.28	17.86
2022-09-08	25.02	274	27.74	0.07	26.75	27.91	17.70	18.69
2022-09-09	28.50	256	30.69	-0.03	31.15	31.38	19.32	21.04
2022-09-10	20.79	279	23.30	0.01	23.17	23.33	16.74	16.12
2022-09-11	27.90	245	26.97	-0.01	27.11	27.61	18.50	19.10
2022-09-12	19.00	243	26.16	-0.17	28.36	26.33	17.95	17.47
2022-09-13	17.42	240	19.80	-0.02	19.97	20.31	14.54	15.42
2022-09-14	22.01	199	23.79	0.06	23.01	23.79	15.16	16.71
2022-09-15	23.74	185	26.07	0.10	24.76	26.13	16.85	17.70
2022-09-16	83.04	280	50.42	-1.47	79.22	79.20	61.76	64.55
2022-09-17	32.30	276	31.35	-1.32	47.73	33.01	39.09	21.92
2022-09-18	18.75	161	21.58	-0.72	28.28	21.82	21.93	15.91
2022-09-19	17.70	295	22.73	-0.08	23.58	23.12	15.00	16.41
2022-09-20	16.06	140	18.56	-0.09	19.34	19.47	14.06	15.94
2022-09-21	15.18	262	20.61	0.01	20.54	21.70	14.41	16.60

Table 7.10: Results for forecast task.

7.7.2 Real-time predictions

For this task, the models have been run on a cluster of computers equipped with 128 GB of RAM and two quadcore Intel XEON E5-2637 v4 (3.5 GHz). For **LightGBM** model, the wall-clock time limit was set to 2 hours and the memory limit to 64 GB.

For this task, we use only one dataset from $\mathcal{I}_{12}$, $\mathcal{I}_{14}$ defined as follows:

$$\{\mathcal{D}_{60}^{12}\} \quad (\mathcal{I}_{14})$$

We use the best model from Section 7.7.1 and the baseline model.

$$\{\text{FastTree}_{BFF,PCFDF,WCF,OMF,PFF}^{75,256,0.05}, \text{Baseline}\} \quad (M_3)$$

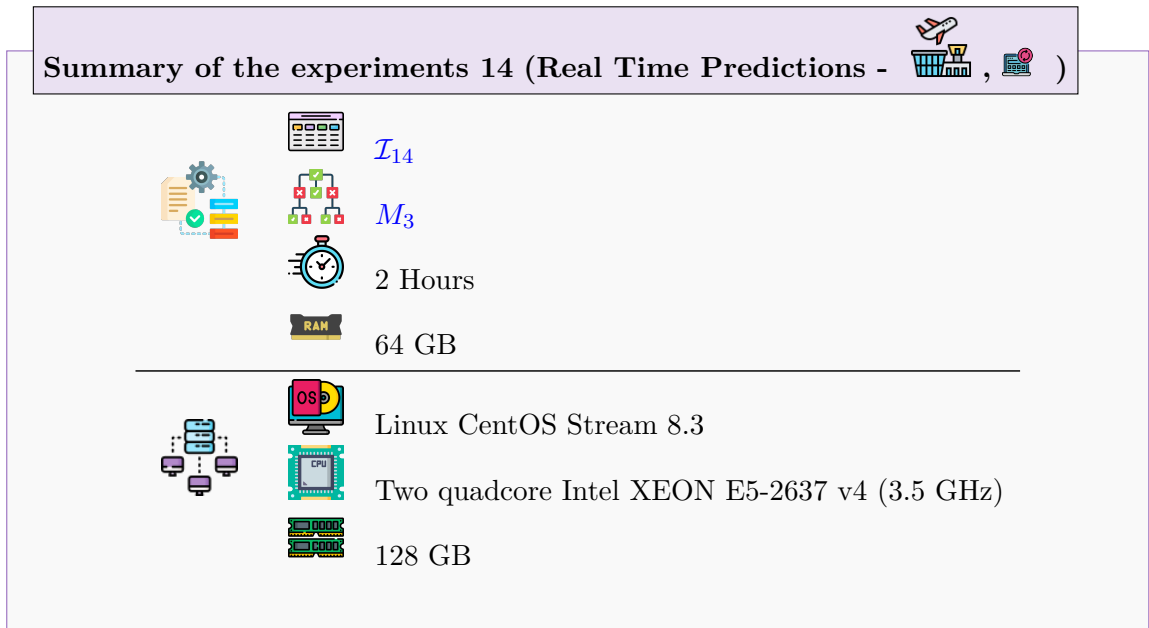


Table 7.11 presents the R^2 score, the **RMSE** and the **MAE** for the days tested with models from M_3 . We notice that the proposed model is more accurate and stable regarding error than the baseline model. For example, for September 1st, the mean absolute error for our model is 6.70 while the error for the baseline model is more than twice (17.26). However, we can also notice that September 16th significantly increased errors²⁵. The delays could go up to 4h30 (the average delay over the day was 1h20). In addition, as the period covered for each flight is between -2h30 and +2h30, we only collect some of the data on these delays.

Tables 7.12, 7.13, and 7.14 display scores corresponding to predictions made 2 hours, 1 hour, and 30 minutes prior to the flight's actual departure, respectively. It's evident from these tables that prediction accuracy improves, with errors diminishing as the departure time approaches.

Figures 7.5 and 7.6 provide insights into the prediction errors relative to actual delays. The x -axis denotes the time, in minutes, leading up to the flight's departure. Specifically, Figure 7.5 illustrates the error trajectory for an actual delay of 15 minutes, whereas Figure 7.6 delineates the error trend for a 75-minute delay. Across both scenarios, it is notable that the error gravitates towards 0. For instance, an hour ahead of the flight departure, the average error stands at about 3 minutes for both delay durations.

²⁵On Thursday, September 16, the delays are due to an air traffic controllers' strike at CDG.

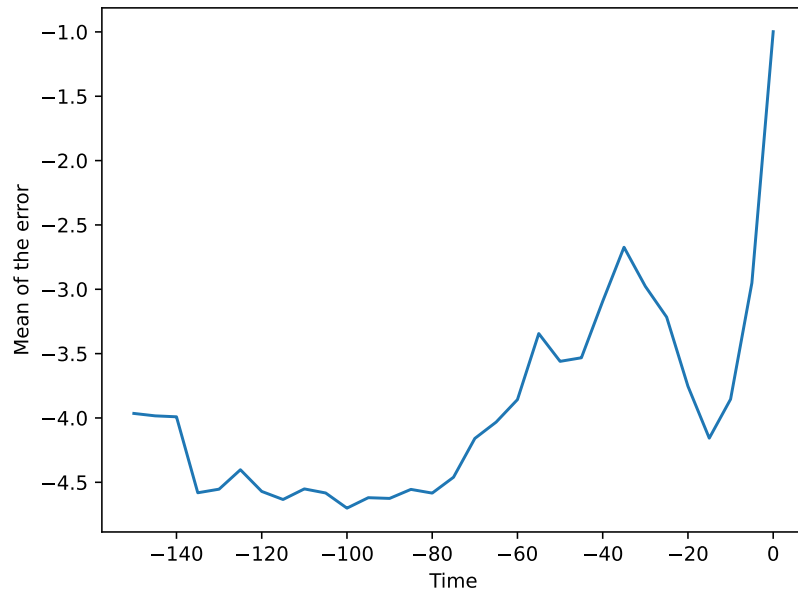


Figure 7.5: Evolution of the error for a delay of 15 minutes.

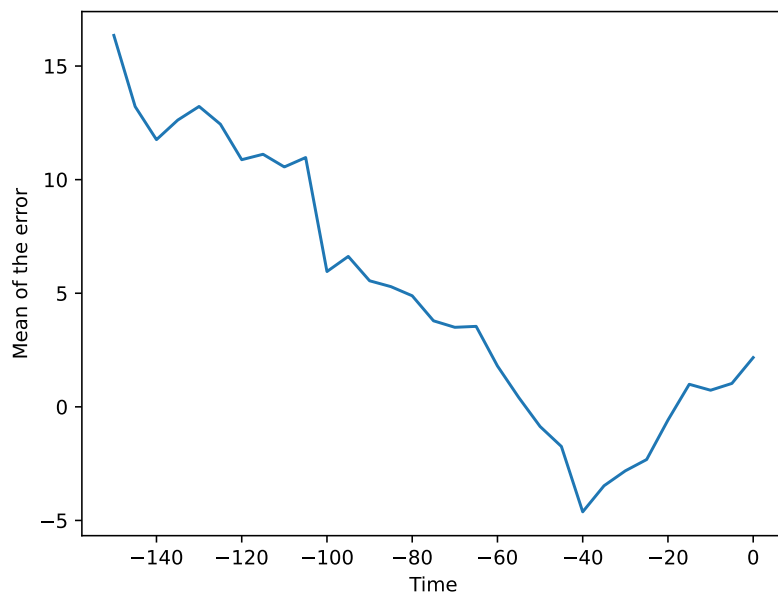


Figure 7.6: Evolution of the error for a delay of 75 minutes.

Date	R^2	RMSE	RMSE _{Baseline}	MAE	MAE _{Baseline}
2022-08-13	0.81	14.07	32.56	8.19	18.89
2022-08-14	0.74	14.00	28.15	8.49	18.25
2022-08-15	0.75	12.53	25.06	8.03	16.99
2022-08-16	0.67	21.06	37.14	9.92	20.99
2022-08-17	0.74	13.70	27.76	8.28	18.69
2022-08-18	0.78	12.18	26.19	7.56	16.92
2022-08-19	0.67	13.29	23.12	7.34	16.19
2022-08-20	0.77	12.25	25.30	7.32	16.83
2022-08-21	0.77	13.75	28.82	7.12	17.03
2022-08-22	0.42	23.24	30.70	7.48	16.75
2022-08-23	0.52	22.70	32.65	7.49	17.28
2022-08-24	0.66	17.58	30.35	8.13	18.89
2022-08-25	0.65	13.21	22.54	7.24	16.28
2022-08-26	0.71	26.08	48.80	7.97	18.76
2022-08-27	0.76	16.33	33.41	7.06	16.93
2022-08-28	0.81	10.98	25.13	6.53	16.40
2022-08-29	0.78	11.77	25.13	6.78	16.91
2022-08-30	0.80	11.14	24.84	6.53	16.62
2022-08-31	0.77	12.64	26.47	7.07	17.57
2022-09-01	0.70	15.22	27.71	6.70	17.26
2022-09-02	0.79	11.93	27.14	7.34	18.37
2022-09-03	0.78	13.05	28.65	7.40	18.71
2022-09-04	0.79	14.17	30.67	7.37	18.31
2022-09-05	0.66	24.65	43.05	9.30	19.81
2022-09-06	0.80	12.72	28.95	7.50	18.70
2022-09-07	0.85	11.53	30.12	6.89	18.05
2022-09-08	0.84	14.64	37.48	8.03	20.06
2022-09-09	0.80	15.53	36.18	8.76	21.84
2022-09-10	0.65	16.11	27.46	7.43	16.48
2022-09-11	0.70	15.90	30.54	8.32	19.67
2022-09-12	0.74	14.33	28.25	7.08	17.13
2022-09-13	0.53	17.83	26.11	7.24	15.40
2022-09-14	0.51	23.20	33.23	8.62	17.45
2022-09-15	0.79	13.23	29.40	7.55	18.06
2022-09-16	0.32	44.39	81.82	31.24	65.07
2022-09-17	0.76	18.14	40.11	9.55	23.83
2022-09-18	0.84	11.25	28.29	6.88	15.98
2022-09-19	0.65	17.61	29.89	8.16	16.89
2022-09-20	0.69	10.44	18.98	6.38	15.03
2022-09-21	0.69	13.29	24.28	7.27	16.16

Table 7.11: Daily results for real-time prediction.

Date	R^2	RMSE	RMSE _{Baseline}	MAE	MAE _{Baseline}
2022-08-13	0.74	16.40	32.84	11.63	19.02
2022-08-14	0.63	17.17	28.96	12.76	18.89
2022-08-15	0.57	15.82	24.12	12.93	16.39
2022-08-16	0.76	17.24	35.83	12.48	20.98
2022-08-17	0.59	16.89	27.33	12.24	18.46
2022-08-18	0.65	15.06	26.01	11.54	16.75
2022-08-19	0.61	14.09	22.45	10.58	16.01
2022-08-20	0.58	14.62	22.63	11.05	16.39
2022-08-21	0.71	13.23	24.59	10.76	16.28
2022-08-22	0.68	13.14	23.59	9.88	15.72
2022-08-23	0.59	14.39	22.66	10.49	16.48
2022-08-24	0.62	17.28	28.04	11.10	18.53
2022-08-25	0.45	16.49	22.29	10.81	16.17
2022-08-26	0.67	14.45	25.21	10.66	16.72
2022-08-27	0.64	13.41	22.34	9.88	15.36
2022-08-28	0.68	12.80	22.69	9.97	16.14
2022-08-29	0.63	13.83	22.91	10.23	16.65
2022-08-30	0.67	14.07	24.50	10.07	16.40
2022-08-31	0.67	15.06	26.24	10.05	17.51
2022-09-01	0.68	14.17	25.07	9.17	16.67
2022-09-02	0.71	14.29	27.65	10.66	18.67
2022-09-03	0.72	14.56	28.40	10.37	18.56
2022-09-04	0.74	14.56	28.51	10.37	17.80
2022-09-05	0.62	17.35	28.83	12.61	18.86
2022-09-06	0.68	15.89	28.30	11.17	18.15
2022-09-07	0.69	14.05	25.06	10.16	16.66
2022-09-08	0.72	15.17	29.26	10.96	18.86
2022-09-09	0.71	17.38	33.64	11.82	21.50
2022-09-10	0.65	14.51	24.66	10.41	15.66
2022-09-11	0.71	16.04	31.00	11.68	19.90
2022-09-12	0.76	13.46	27.55	9.62	16.73
2022-09-13	0.58	14.01	21.58	10.24	15.02
2022-09-14	0.50	18.02	25.60	11.49	16.95
2022-09-15	0.67	14.88	26.19	10.50	17.09
2022-09-16	0.24	44.87	83.65	34.47	68.75
2022-09-17	0.64	18.51	33.26	12.57	21.95
2022-09-18	0.51	14.33	20.55	10.50	14.67
2022-09-19	0.64	14.85	24.83	10.05	15.98
2022-09-20	0.47	13.65	19.06	9.86	15.15
2022-09-21	0.58	15.40	24.11	10.50	15.98

Table 7.12: Results for the real-time task, 2 hours before the departure.

Date	R^2	RMSE	RMSE _{Baseline}	MAE	MAE _{Baseline}
2022-08-13	0.81	13.35	30.77	10.39	18.49
2022-08-14	0.71	14.09	26.60	10.67	18.14
2022-08-15	0.71	13.41	25.04	10.70	16.75
2022-08-16	0.80	13.72	31.08	10.30	19.42
2022-08-17	0.77	13.20	28.77	10.19	18.94
2022-08-18	0.72	12.89	24.89	9.87	16.65
2022-08-19	0.69	12.13	21.85	9.73	16.08
2022-08-20	0.65	13.48	22.69	10.13	16.41
2022-08-21	0.66	12.22	21.25	9.78	15.54
2022-08-22	0.78	10.93	23.66	8.70	15.79
2022-08-23	0.63	12.45	20.79	9.22	16.11
2022-08-24	0.73	13.61	26.16	9.62	18.03
2022-08-25	0.68	12.81	22.81	9.35	16.39
2022-08-26	0.63	11.88	19.60	9.20	15.43
2022-08-27	0.75	12.02	23.87	8.85	15.87
2022-08-28	0.76	10.79	22.07	8.55	15.84
2022-08-29	0.72	11.60	22.13	8.91	16.36
2022-08-30	0.74	11.85	23.15	8.77	16.20
2022-08-31	0.80	11.22	25.24	8.35	17.38
2022-09-01	0.79	11.70	25.62	8.39	16.85
2022-09-02	0.78	12.05	27.10	9.05	18.29
2022-09-03	0.82	11.66	28.56	8.89	18.74
2022-09-04	0.80	12.05	27.24	9.28	17.61
2022-09-05	0.77	12.69	27.31	9.84	18.14
2022-09-06	0.75	13.04	26.02	9.62	17.73
2022-09-07	0.77	12.12	25.23	9.15	16.95
2022-09-08	0.80	12.55	28.29	9.41	18.59
2022-09-09	0.79	13.79	31.39	10.01	20.62
2022-09-10	0.68	12.11	21.47	8.90	15.25
2022-09-11	0.70	13.82	26.50	9.88	18.59
2022-09-12	0.82	11.11	26.20	8.82	16.71
2022-09-13	0.58	11.92	18.53	8.93	14.35
2022-09-14	0.65	14.69	25.02	9.72	16.76
2022-09-15	0.77	12.46	26.44	9.17	17.20
2022-09-16	0.35	40.53	81.29	29.68	66.79
2022-09-17	0.78	15.09	34.94	10.46	22.62
2022-09-18	0.67	12.53	21.80	9.15	15.21
2022-09-19	0.69	12.80	22.95	9.01	15.67
2022-09-20	0.55	12.19	18.65	8.92	14.91
2022-09-21	0.46	14.09	19.71	9.64	15.33

Table 7.13: Results for the real-time task, 1 hour before the departure.

Date	R^2	RMSE	RMSE _{Baseline}	MAE	MAE _{Baseline}
2022-08-13	0.81	11.53	26.65	9.06	17.19
2022-08-14	0.83	10.49	26.12	8.22	17.73
2022-08-15	0.76	12.00	24.48	9.03	16.82
2022-08-16	0.84	10.67	27.27	8.20	18.29
2022-08-17	0.83	10.71	27.27	8.36	18.43
2022-08-18	0.73	12.10	23.88	8.42	16.47
2022-08-19	0.79	10.39	22.57	8.16	16.25
2022-08-20	0.76	10.89	22.20	8.42	16.42
2022-08-21	0.73	10.36	20.44	8.32	15.19
2022-08-22	0.77	9.58	20.70	7.43	15.27
2022-08-23	0.74	10.58	20.79	7.96	16.11
2022-08-24	0.80	11.26	24.88	8.34	17.49
2022-08-25	0.78	11.08	23.47	8.21	16.53
2022-08-26	0.75	10.13	20.27	7.82	15.69
2022-08-27	0.84	9.52	24.02	7.36	15.98
2022-08-28	0.79	9.43	21.00	7.48	15.57
2022-08-29	0.80	9.62	21.80	7.48	16.21
2022-08-30	0.80	10.56	23.43	7.37	16.35
2022-08-31	0.84	9.96	25.12	7.49	17.29
2022-09-01	0.82	9.87	23.56	7.17	16.35
2022-09-02	0.84	9.41	24.77	7.31	17.58
2022-09-03	0.86	9.40	25.90	7.19	17.98
2022-09-04	0.87	9.12	25.11	7.23	17.04
2022-09-05	0.84	10.07	25.40	7.93	17.29
2022-09-06	0.86	10.03	26.59	7.63	18.14
2022-09-07	0.84	9.94	24.65	7.64	16.98
2022-09-08	0.84	10.38	26.12	8.00	17.66
2022-09-09	0.86	10.61	29.89	8.07	20.18
2022-09-10	0.76	10.82	21.92	7.91	15.54
2022-09-11	0.78	11.78	26.37	8.37	18.52
2022-09-12	0.83	9.96	24.38	7.57	16.19
2022-09-13	0.73	9.88	19.28	7.43	14.50
2022-09-14	0.71	10.91	20.38	7.70	14.95
2022-09-15	0.84	10.67	26.80	7.88	17.85
2022-09-16	0.36	39.04	77.93	27.26	63.84
2022-09-17	0.83	12.78	33.01	8.33	21.87
2022-09-18	0.75	11.59	23.23	8.04	15.94
2022-09-19	0.67	12.05	21.10	8.13	15.14
2022-09-20	0.71	10.17	19.24	7.40	15.13
2022-09-21	0.60	12.20	19.84	8.52	15.43

Table 7.14: Results for the real-time task, 30 minutes before the departure.

7.8 Explaining off-block delay predictions

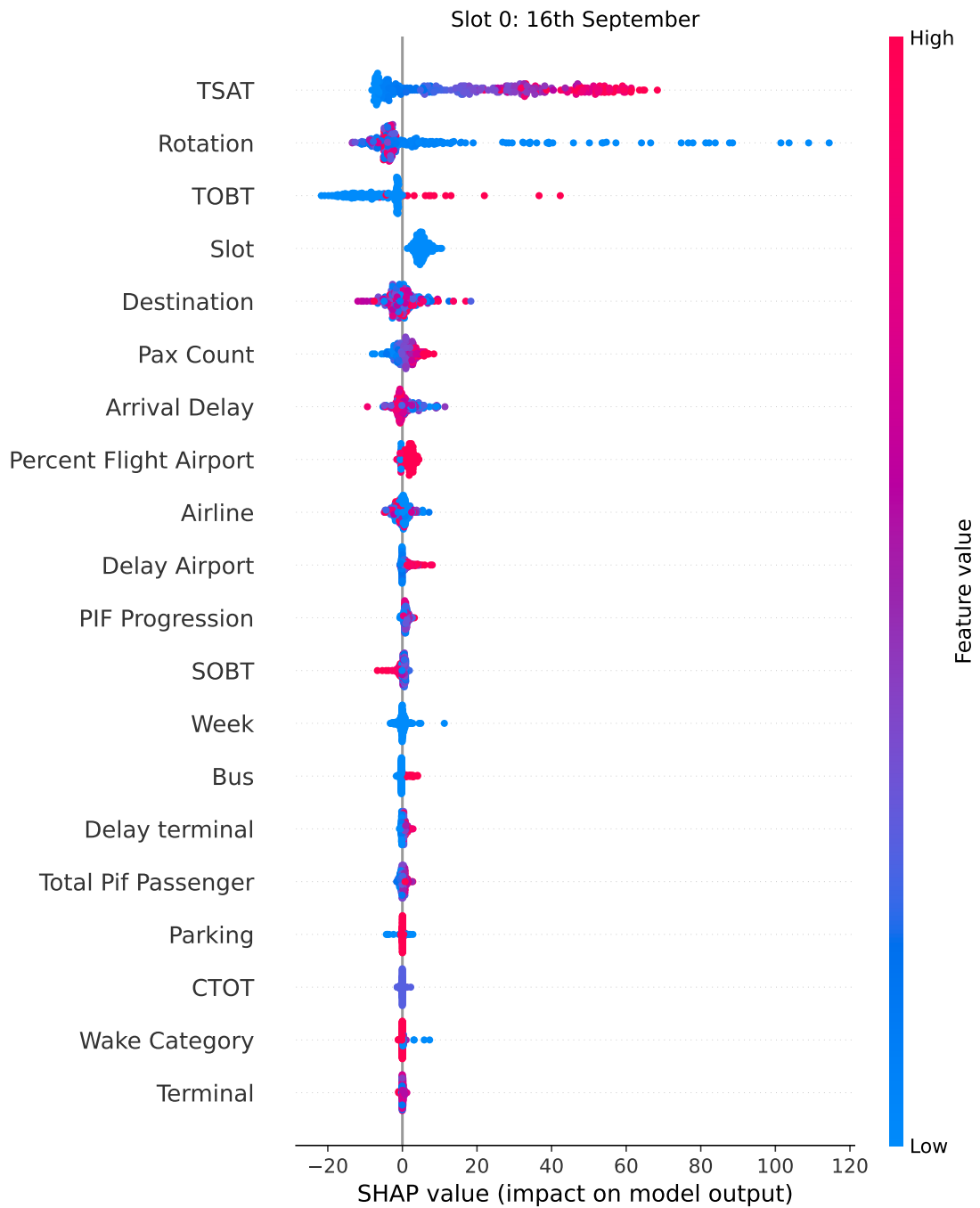


Figure 7.7: SHAP values for September 16th 2022 at slot 0

Figures 7.7 and 7.8 give an overview of the most essential features of our model. The features are sorted by the sum of the SHAP values over all samples at slots 0 and 30. The color represents the value of the feature (*red* corresponds to *high*, *blue* to *low*). This reveals, for example, that when the slot is 30, a high value for TSAT increases the predicted delay. Finally, the SHAP values largely confirm the coefficients of *Pearson* from Section 7.5.2 and the importance of the

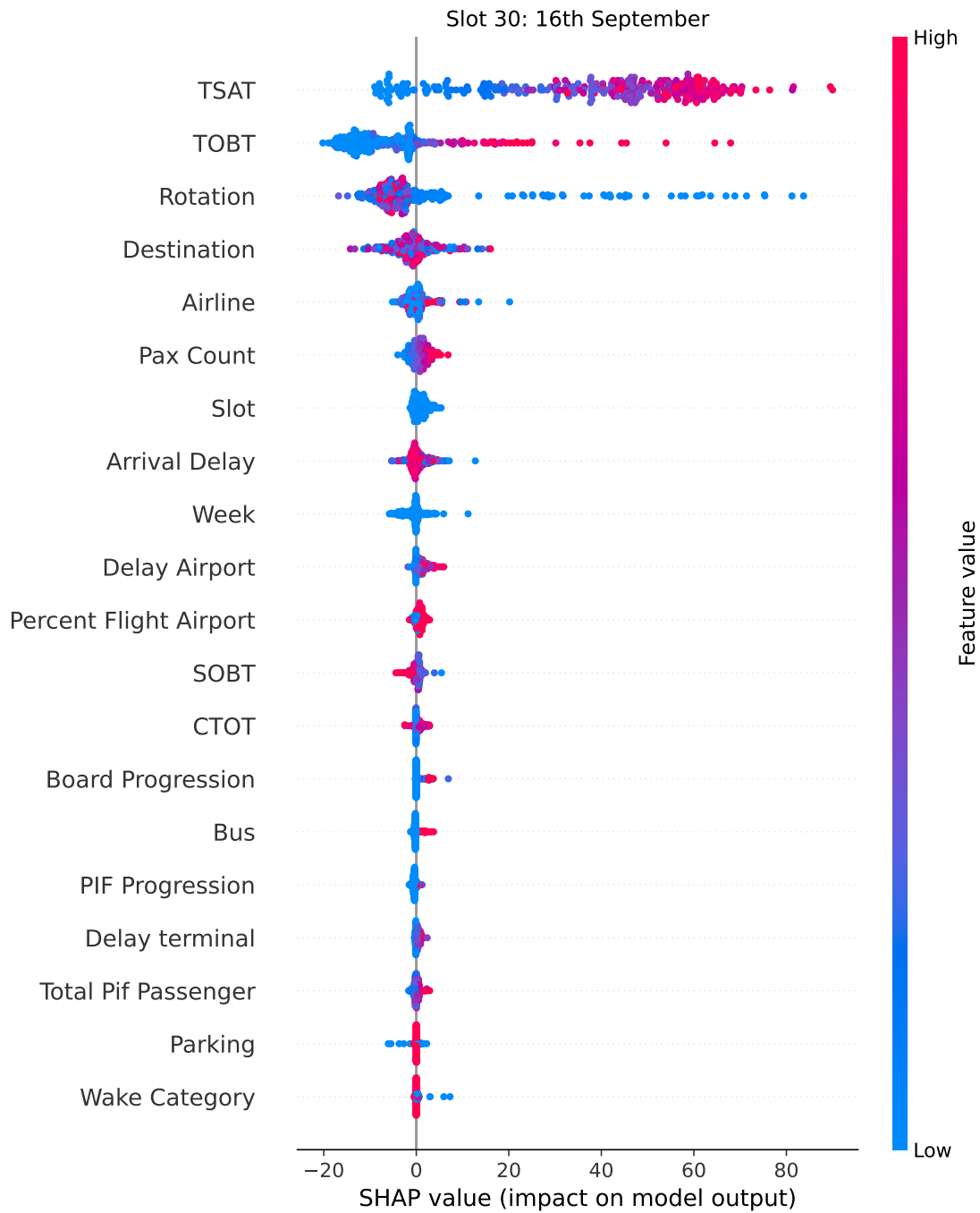
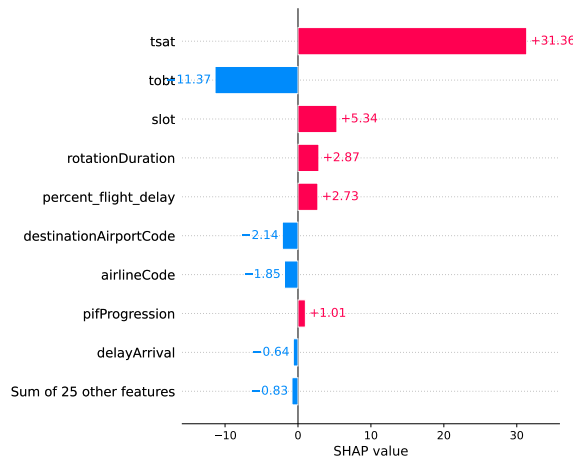


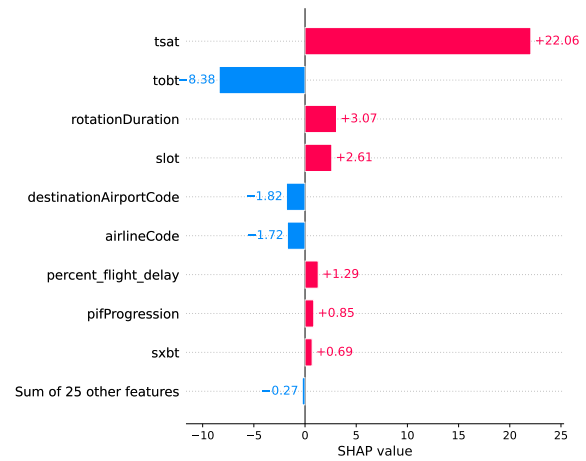
Figure 7.8: SHAP values for September 16th 2022 at slot 30

impact of the TSAT and TOBT variables.

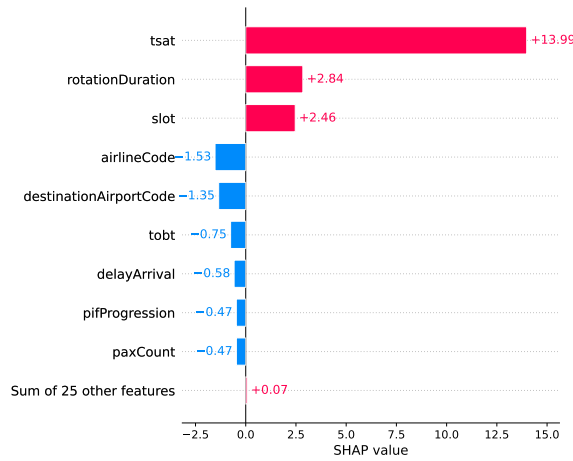
7.8. Explaining off-block delay predictions



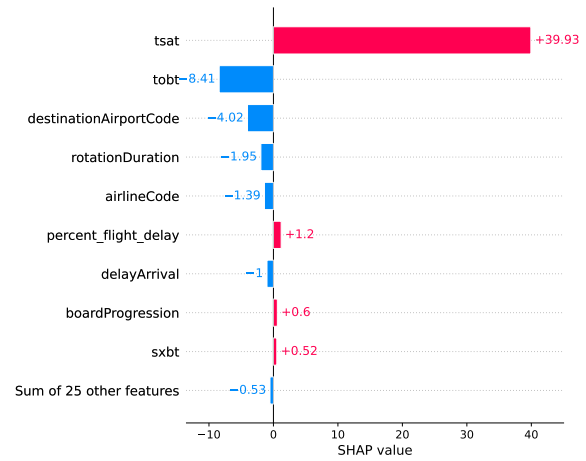
(a) 2h30 before scheduled off-block time



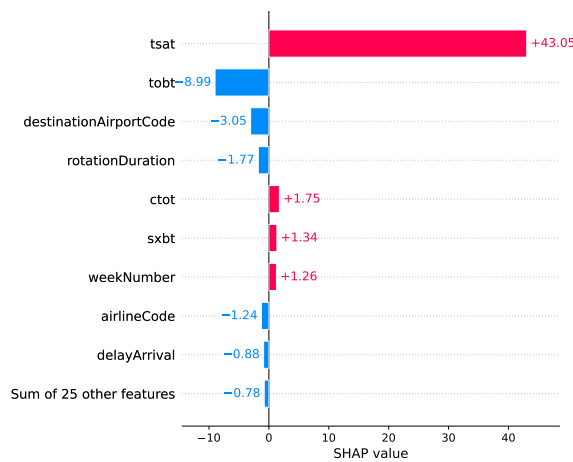
(b) 1h40 before scheduled off-block time



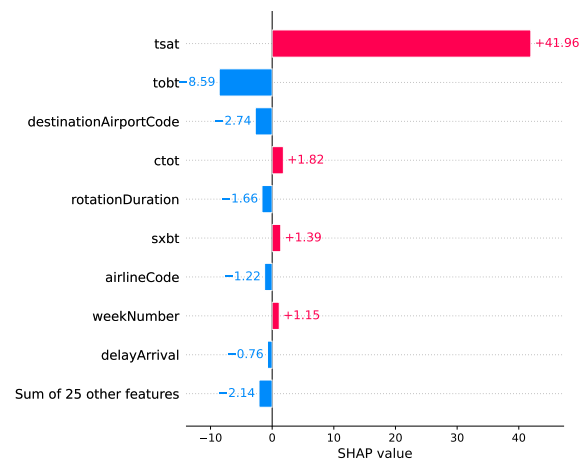
(c) 40 minutes before scheduled off-block time



(d) At scheduled off-block time



(e) 30 minutes after scheduled off-block time



(f) At actual off-block time

Figure 7.9: Evolution of the feature importance for a flight of 16th september with a delay of 60 minutes.

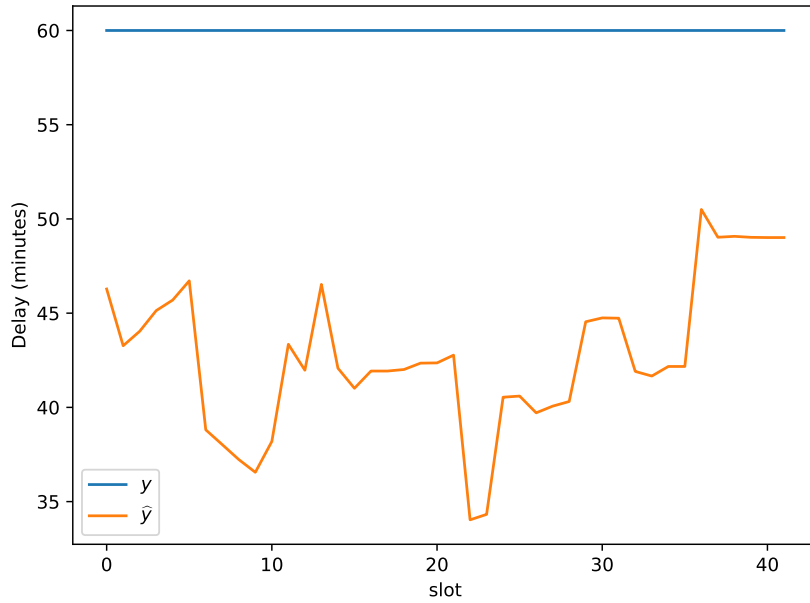


Figure 7.10: Evolution of the prediction for flight with a delay of 60 minutes.

Figure 7.9 shows an overview, for a Lufthansa flight with a delay of 60 minutes, of the feature-importances at different moments of the prediction. We can see that the evolution of the TSAT estimate will play a role in the prediction and influence positively or negatively.

Figure 7.10 shows the evolution of the prediction for the same flight. The average error of our prediction is about 13 minutes. We can observe that the prediction decreases at some points and ends at 49 minutes.

7.9 Deployment

AOP is a software written in C# using a PostgreSQL database. It creates snapshots of the evolution of flights and allows us to find the state of a flight at any time from its arrival to its departure. For space reasons, the data is compressed in the database, so the AOP has an engine that allows us to retrieve the information we want on the fly. The AOP uses different machine-learning models to fill in missing data and predict the use of specific resources such as security checkpoints. The machine learning models use the `ML.net` library that proposes an implementation for `LightGBM` model. Each model is rebuilt every night to take into account the last day. Generating the dataset used in this study takes about 20 hours. This time makes it difficult to use the model directly in production and rebuild them every night.

To solve this problem, we can use a cache by generating the complete dataset once and simply adding the new day to the dataset. This solution is currently being deployed and tested. The learning time for the model is less than 1 hour in Python and should remain reasonable in C#.

7.10 Conclusions

Punctuality is a sensitive issue in major airports and hubs for the passenger experience. In this chapter, we have addressed the problem of predicting delays at the departure of parking lots at Paris-CDG airport, one of the largest airports in the world and the hub of Air France.

Our study started with analyzing the problem (its magnitude, form, causes, ...) and the needs (real-time forecasting and prediction) at Paris-CDG. Based on this analysis, we proposed two types of feature categories that could be useful for delay prediction: static features used for the forecasting task and dynamic features (which can be updated in real-time) used for real-time delay prediction. The next step was to build a pipeline to extract the raw data we needed from the operational information system of Paris-CDG. This allowed us to build a dataset representing one year of activity. We conducted an empirical study to select the characteristics and the data and then to select the models. One of the specificities of our work is that we worked with very fluctuating data due to the COVID-19 pandemic and its consequences in terms of air travel restrictions on several occasions and other air traffic hazards. The results show that some delays can be predicted much better than the baseline model. This result can be significantly improved by systematically exploring other models and their best hyperparameters. It is also possible to exploit information from other flight milestones related to airside operations, such as the progress of luggage loading or its progression on the baggage circuit. In addition to the accuracy improvement, one of the crucial elements for our application will be the explicability of the predictions, particularly the identification of explanations that can help in delay management.

General Conclusion and Perspectives

Throughout this manuscript, we embarked on a journey through the world of airport operations, guided by the powerful lens of *Constraint Programming* (CP) and *Machine Learning* (ML).

Chapter 1 introduced us to the realm of CP [Mon74, RvW06, Lec09], a paradigm rooted in representing and solving combinatorial problems. The essence of this chapter revolved around framing real-world problems, particularly airport-centric challenges, as Optimization Problems. Herein, decision variables and constraints are crafted in a manner that mimics natural language descriptions. This chapter introduces two fundamental problems study in this manuscript: *Stand Allocation Problem* (SAP) and *Check-in Desk Allocation Problem* (CDAP).

Following the detailed exploration of CP in Chapter 1, Chapter 2 introduced readers to another powerful computational tool: ML. Rooted in the ability to learn from data and make predictions, ML serves as an invaluable asset in the dynamic, data-rich environment of modern airports. Machine Learning is characterized by its various paradigms, including *Supervised Learning*, *Unsupervised Learning*, and *Reinforcement Learning*. Within the context of airport operations, as showcased in Chapter 2, the focus has predominantly been on *Supervised Learning*. This paradigm, which deals with labeled datasets, holds great potential for airports like Paris Airports. With vast amounts of flight and passenger data available, supervised learning algorithms have the capability to discern patterns, predict outcomes, and subsequently optimize resource allocation. Using a fictive task of predicting the number of hot dog sales the chapter introduces a complete pipeline and analysis from the data analysis to the prediction and explanation task. The rest of the manuscript follows the same approach. The chapter also highlighted the critical role of machine learning within the AOP system at Paris Airports. The ability to calculate and predict passenger presentations at each airport resource for a given day represents a significant leap in operational efficiency and passenger experience. Two primary applications were underlined in Chapter 2: predicting passengers with reduced mobility and off-block delays.

First of all, this manuscript was an opportunity to present the Metrics²⁶ analysis tool [FWW21], a library designed to standardize the methods of extracting and analyzing constraint solvers and, broadly, any software generating time-stamped outcomes. All campaigns featured in this manuscript were orchestrated using the Metrics tool, streamlining the demonstration process and enabling readers to easily understand and replicate the analyses.

Chapter 3 [FALM23] focuses on the modeling of the *Check-in Desk Allocation Problem* (CDAP). The core of this chapter was dedicated to exploring diverse modeling strategies for different variants of this problem: **no – overlapping**, **overlapping** and **overlapping – nbmax**. For this latter, we propose 3 types of modeling based on some refinements of specific variables introduced and kinds of constraints.

With experimental environments serving as a backdrop, the chapter casts light on how various solvers perform under different circumstances, emphasizing the paramount importance of

²⁶<https://github.com/crillab/metrics>

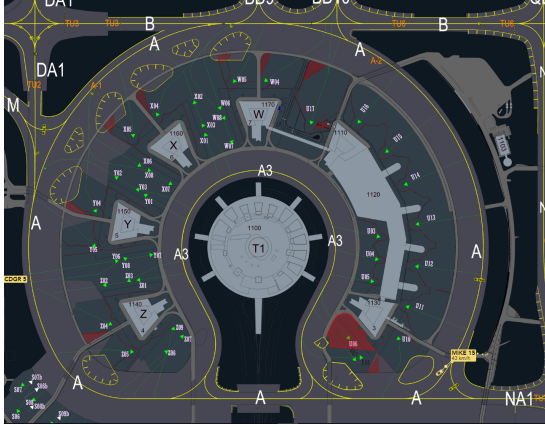
aligning solver configurations with specific problem instances. A standout revelation from this exploration was the performance of the **Gather** method, for handling with the large number of constraint **AllDiffExcept**. Beyond highlighting the superiority of specific techniques, the chapter also revealed the oscillating performance of various solver configurations in different scenarios. This underscores the need for a thorough understanding of the problem in order to correctly model and select the right solver configuration. Although choosing the right solver configuration is at least as complex a task as modeling the problem itself. Moreover, juxtaposing our strategies with the ADP algorithm offered an enriched comparison. While it is tempting to gravitate towards simpler algorithms for their straightforwardness, this chapter has shown that there are tangible benefits—performance, solution quality, and adaptability—in turning to more robust, generic solvers. Currently, a first version using Python for modeling [LS20] is being used in conjunction with the old version for “visual” comparisons made by the users. It will go definitively into production before 2024. A perspective could be to work on a modeling interface with a language faster than Python while retaining the modeling simplicity that Python brings.

Chapter 4.2.3 introduces a modelization for *Stand Allocation Problem* (SAP) based on the rules and constraints of CDG airport [FLMT22, FMLT22]. After introducing a first model of this problem, the chapter introduces a first variant proposing to adapt what is done in the state-of-the-art by adding **allDifferent** constraints based on the interval graph of temporally overlapping tasks. Finally, the chapter proposes an adaptation of the objective function to take into account the minimization of aircraft movements. Since we use a solver that does not handle multi-objectives, we have done this directly in the objective function, by adding to the satisfaction calculation the number of times we have not moved an aircraft for rotations with at least two tasks. Initial experiments have shown our approach to be superior to the current solver used by ADP. However, as we reflect on the in-depth analysis presented, it is essential to understand the context: this chapter primarily served as a proof of concept. In this light, while the methodologies introduced lay a strong foundation and demonstrate the potential for comprehensive solutions, they also underscore a need for further refinement. A first perspective must be to model a specific case for terminal 1 of CDG based on specific infrastructure of this terminal. Let us take a look at Figure 11b: if a plane is placed in Y07 and another plane is placed in Y06, we need to make sure that the plane in Y07 leaves before the plane in Y06, because the planes arrive on the outside (route marked by A) and leave on the inside (route A3). Note that the direction is reversed if we take stands Z02 and Z01.

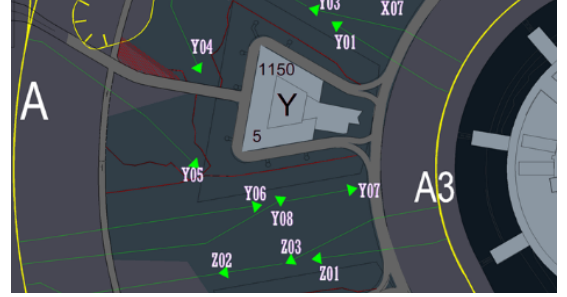
Another perspective could be to view the stand allocation problem as a genuine multi-objective issue, and to address this, one could use solvers such as Choco [PFL16] for resolution.

Chapter 5 delves deep into the realm of advanced resolution techniques with the primary objective of amplifying the efficiency and adaptability of solving Constraint Programming problems. This chapter starts by introducing the *Aggressive Bound Descent* technique, which ventures into aggressive adjustments of objective constraints. This risk-centric approach, where solvers occasionally plunge into potentially UNSAT search space, showcased a notable performance enhancement for constraint optimization problems. Although the immediate performance metrics may seem modest, further research, as detailed in [FLMW22], not only confirmed the method’s potential but also highlighted an increase in solver efficiency. In particular, the integration of ABD into the PB Sat4j solver has demonstrated the full capabilities of this approach.

After that, the chapter explores the usage of PB encoding for resolving CSP problems. By crafting new encodings grounded in PB constraints, we aimed to capitalize on the potent inference capabilities of PB solvers, notably their intrinsic prowess in counting. While empirical studies signaled the potency of our encodings for problems dominated by sum and cardinality constraints [FW22b, FW22a], this enhanced performance was not universal. Native CP solvers



(a) Terminal 1 of CDG.



(b) Example of a special traffic situation linked to the CDG1 infrastructure.

Figure 11: Overview of the terminal 1 of CDG.

retained their edge in scenarios involving a wide range of constraints. This observation underlines the necessity for future studies to rethink our current encodings and possibly create new variations that can truly tap into the raw power of PB solvers, with a penchant for pure PB constraints over traditional clauses.

Finally, the last part of the chapter unveiled a novel framework **PANORAMYX**²⁷ [FLW23] designed to spearhead the development of parallel and distributed constraint solvers. This tool is part of a larger universe of tools dedicated to constraint programming: **TOOTATIS**²⁸. **PANORAMYX** is based on the interfaces offered by the **Universe**²⁹ library. Through an ensemble of meticulously designed object-oriented solver interfaces (**Universe**), this framework paves the way for integrating a diverse array of SAT, PB, or CP solvers. We propose an **EPS** approach based on the hypergraph decomposition of the problem. This approach improves classical **EPS** approaches. It would now be interesting to extend this approach to optimization problems and to compare the performance of airport problems such as **CDAP** by automatic decomposition rather than based on in-depth knowledge of the problem structure as initiated in this manuscript. Finally, it would be possible to combine the different configurations to achieve good performance using a portfolio so that they become complementary.

Chapter 6 delved into the pivotal challenge of ensuring accessibility and inclusivity within the realm of air travel. As the world of aviation grows exponentially, ensuring convenience and safety for passengers with reduced mobility (PRMs) remains a paramount concern, especially for hubs as significant as Paris Airports. The chapter emphasized the operational importance of accurately predicting the number of PRMs transiting through the airports daily. Beyond operational efficacy, these predictions also have financial implications, as misestimations can result in suboptimal staffing and consequently, financial discrepancies. By focusing on historical data and contemporary machine learning methods, we sought to enhance the accuracy of these predictions. The heart of our exploration centered around the **FastTree** model, influenced by Paris Airports' inclination towards Microsoft's **ML.net** framework. Not only did this model fit well within the technological landscape of the Paris Airport, but its speed, ease of deployment,

²⁷<https://github.com/crillab/panoramyx>

²⁸<https://crillab.github.io/tootatis/>

²⁹<https://github.com/crillab/universe>

and proven performance made it an ideal choice for our purposes. The comprehensive data analysis brought forth intriguing insights, underscoring the correlations between various flight and PRM-related features and the target variable. With meticulous data parsing and feature engineering, our model exhibited the potential to improve PRM predictions. This work is currently used by the ADP group to indicate the number of PMR to service providers in replacement of the prediction taken by the commercial tool *PRM Manager*.

Chapter 7 tackled the intricate challenge of predicting flight delays at the bustling Paris-CDG airport. Given the airport’s massive scale, with an average of over one flight departure per minute and encompassing around 498,000 aircraft movements annually, efficient and accurate resource management becomes paramount. One of the most significant resources in this context is the flight stands. A late release of a stand can set off a domino effect, leading to a cascade of delays and operational challenges.

The chapter started with a detailed examination of flight rotations – sets comprising of arrival and departure flights. Delays were meticulously defined, centering on the discrepancy between the scheduled departure time (SOBT) and the actual off-block time (AOBT). The center of the research revolved around predicting these delays to foster operational efficiency and enhance the passenger experience [FMT23a, FMT23b].

Diving deep into the data, the chapter introduced two primary categories of features pertinent to delay prediction: static and dynamic. Static features, being constant, catered to forecasting tasks, while dynamic features, with the flexibility of real-time updates, were tailored for real-time delay predictions. The data extraction process was intricate, pulling from the operational information system of Paris-CDG to curate a comprehensive dataset spanning an entire year.

One of the unique aspects of our study was navigating the unpredictability induced by the COVID-19 pandemic. The pandemic’s influence introduced erratic fluctuations in flight data, given the fluctuating air travel restrictions and other operational challenges during this period.

Nevertheless, our empirical studies culminated in results surpassing the baseline model in predicting certain delays. This achievement underscores the potential of further enhancing accuracy by delving into other models, optimal hyperparameters, and exploring additional flight milestones.

Moreover, while accuracy is undoubtedly vital, the chapter also emphasized the significance of explicability in predictions. Being able to provide clear and coherent explanations for predictions not only fosters trust but also empowers operational teams with actionable insights to manage delays.

Through the research conducted in this thesis, we’ve transitioned from an antiquated, proprietary constraint programming approach to adopting modern methodologies rooted in open-source solutions. This shift promises enhanced flexibility and scalability for the continued evolution of the DCB. Moreover, the initial exploration into machine learning signifies the dawn of its broader application within airport environments, paving the way for pivotal predictions concerning factors like the volume of PRMs and potential aircraft delays.

Index

Symbols

Cartesian product, 7

A

agent, 29

aggregation, 33

aircraft

aircraft position, 14

allDifferent, 9

arity, 8

attribute, 37, 148

Boolean attribute, 32

categorical attribute, 32

numerical attribute, 32

B

bagging, 33

bank, 21

banks, 52

bound descent

aggressive bound descent, 109

bound descent, 110

C

campaign, 61

check-in

check-in desk, 44

check-in desk allocation problem, 20, 28

check-in desk, 21, 25, 51, 52

compatible check-in, 27

excluded check-in desks, 27

zone, 21

classification, 30, 37

classifier

Bayesian naive classifier, 43

binary classifier, 31

binary classifier, 31

multi-class classifier, 31

clique

clique partitioning formulation, 17

maximum clique, 91, 97

clustering, 30

coefficient of determination, 35

condition, 31

consecutive, 25

consistency

arc-consistency, 59, 138–140

generalized arc consistency, 60

consistent

arc-consistent, 11

arc-inconsistent, 11

constituent model, 34

constraint, 8

binary constraint, 8

constraint optimization problem, 11

extensional constraint, 9, 53

intension constraint, 67

intensional constraint, 8

n-ary constraint, 8

no-overlap constraint, 18

reduction constraint, 18

shading constraint, 18

table constraint, 53

ternary constraint, 8

unary constraint, 8

capacity constraint, 18

consecutive constraint, 25

constraint network, 10, 11, 28

constraint optimization problem, 11, 28

constraint satisfaction problem, 10, 28

exclusion constraint, 27

extension constraint, 67

global constraint, 9

no-overlap constraint, 25

pre-assignment constraint, 27

same zone constraint, 25

unavailable constraint, 26

correlation coefficient, 40

D

- data
 - unseen data, 29
- dataset, 37
 - labeled dataset, 29
- direction, 39
- distance
 - Manhattan distance, 35
- domain
 - current domain, 6
 - finite domain, 6
 - initial domain, 6

E

- error, 34, 35
 - mean absolute error, 35
 - mean squared error, 35
 - root mean squared error, 35
- example, 30
- experimental
 - experimental data, 61
- experimental environment, 52
- explanations
 - local explanations, 36

F

- fail
 - fail first principle, 13
- feature, 30, 37, 148
 - feature importance, 36, 37
- feedback, 30
- flight
 - departing flight, 45
 - arrival flight, 15
 - departure flight, 15
- flight turnaround, 14
- flight: arriving flight, 45
- forest
 - decision forest, 33, 34
 - random forest, 33, 36, 43
 - random forest, 33, 34
- function
 - decision function, 30

G

- gates, 14
- gradient boosting, 34

H

- heuristic
 - value ordering heuristic, 13
 - variable ordering heuristic, 13

I

- input
 - input data, 29, 36, 37
 - input files, 61
- inputs, 29
- instance, 30
- instantiation, 10
 - valid instantiation, 10
- integer
 - binary integer programming, 17
 - integer programming, 17
- interpretable, 36

L

- label, 30
 - output label, 29
- label encoding, 40
- leaf, 31
- learner, 29
- learning
 - learning problem, 30
 - learning task, 30
 - machine learning, 29
 - reinforcement learning, 30
 - supervised learning, 29
 - unsupervised learning, 30
- LightGBM, 40, 44

M

- maximized, 11
- measurements, 61
- minimized, 11
- model
 - black-box, 36
 - constituent model, 33
 - ensemble model, 32, 33
 - main model, 34
 - model-agnostic, 36
 - model-precise, 37
 - model-specific, 37
 - regression model, 31, 36, 37
 - stacked model, 32
 - strong model, 33, 34
- model-agnostic, 36, 37

modeling, [52](#), [86](#), [92](#)

N

node, [31](#)

O

objective

 objective function, [11](#)

one hot encoding, [40](#)

operation, [20](#)

outputs, [29](#)

overlap

 forbidden overlaps, [26](#), [27](#)

 overlapping rules, [27](#)

overlapping

 time overlapping, [25](#)

P

Pearson correlation coefficient, [39](#)

penalties, [30](#)

polyominoes, [21](#)

prediction, [33](#)

predictions, [29](#)

pseudo-Boolean, [109](#)

PyCSP³, [52](#)

R

ranking, [36](#)

registration, [21](#), [52](#)

regression

 logistic regression, [43](#)

regressor, [31](#)

relation, [7](#)

reward, [30](#), [53](#)

rotation, [14](#), [45](#), [46](#)

rule

 rule extraction, [36](#)

S

sampled with replacement, [33](#)

scope, [8](#)

search

 backtrack search, [14](#)

security checkpoint, [44](#)

set

 training set, [30](#)

 test set, [30](#)

shadow

 shadow restrictions, [20](#)

slot, [44](#)

 time slot, [46](#)

solution, [10](#)

stand, [14](#), [20](#), [44](#), [162](#)

 stand allocation problem, [14](#), [20](#), [28](#)

 contact stand, [14](#)

 hard stand, [14](#)

 remote stand, [14](#)

 stand operation, [15](#)

stands, [20](#)

 compatible stands, [20](#)

strength, [39](#)

successor, [20](#)

support, [10](#)

symmetrical, [18](#)

T

table

 negative table, [58](#)

task, [22](#)

tree

 binary tree, [31](#), [33](#)

 boosted decision tree, [31](#)

 boosted tree, [36](#), [37](#)

 decision tree, [31](#), [37](#), [40](#), [43](#)

 regression gradient boosted decision tree,
 [33](#)

tuple, [7](#)

 allowed tuple, [53](#)

 allowed tuple, [9](#)

 disallowed tuple, [9](#), [53](#)

 valid tuple, [10](#)

V

value

 predicted value, [35](#)

 true value, [35](#)

variable, [6](#), [52](#), [53](#)

 target variable, [37](#)

 discrete variable, [6](#)

 fixed variable, [6](#)

 unfixed variable, [6](#)

 variable assignment, [6](#)

variance, [36](#)

X

XCSP³, [iv](#), [52](#)

Bibliography

- [AAB⁺19] Zeeshan AHMED, Saeed AMIZADEH, Mikhail BILENKO, Rogan CARR, Wei-Sheng CHIN, Yael DEKEL, Xavier DUPRE, Vadim EKSAREVSKIY, Eric ERHARDT, Costin ESEANU, Senja FILIPI, Tom FINLEY, Abhishek GOSWAMI, Monte HOOVER, Scott INGLIS, Matteo INTERLANDI, Shon KATZENBERGER, Najeeb KAZMI, Gleb KRIVOSHEEV, Pete LUFERENKO, Ivan MATANTSEV, Sergiy MATUSEVYCH, Shahab MORADI, Gani NAZIROV, Justin ORMONT, Gal OSHRI, Artidoro PAGNONI, Jignesh PARMAR, Prabhat ROY, Sarthak SHAH, Mohammad Zeeshan SIDDIQUI, Markus WEIMER, Shauheen ZAHIRAZAMI, and Yiwen ZHU. « Machine Learning at Microsoft with ML .NET ». *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2448–2458, July 2019. [2.6](#)
- [ABB⁺22a] Gilles AUDEMARD, Steve BELLART, Louenas BOUNIA, Frédéric KORICHE, Jean-Marie LAGNIEZ, and Pierre MARQUIS. « Les Raisons Majoritaires : Des Explications Abductives Pour Les Forêts Aléatoires ». In *22ème Conférence Francophone Sur l'Extraction et Gestion Des Connaissances (EGC 2022)*, EGC 2022, pages 123–134, Blois, France, January 2022. Editions RNTI. [2.3](#)
- [ABB⁺22b] Gilles AUDEMARD, Steve BELLART, Louenas BOUNIA, Frédéric KORICHE, Jean-Marie LAGNIEZ, and Pierre MARQUIS. « Sur Le Pouvoir Explicatif Des Arbres de Décision ». In *Extraction et Gestion Des Connaissances, EGC*, volume E-38 of *EGC 2022*, pages 147–158, Blois, France, January 2022. Editions RNTI. [2.3](#)
- [AGL⁺19] Gilles AUDEMARD, Gael GLORIAN, Jean-Marie LAGNIEZ, Valentin MONTMIRAIL, and Nicolas SZCZEPANSKI. « pFactory: A Generic Library for Designing Parallel Solvers ». In Hans WEGHORN, editor, *The 16th International Conference on Applied Computing, AC'19*, pages 89–96, 2019. [5.3](#)
- [ALST17] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Nicolas SZCZEPANSKI, and Sébastien TABARY. « A Distributed Version of Syrup ». In Serge GASPERS and Toby WALSH, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2017. [5.3.1](#)
- [Apt03] K. R. APT. *Principles of Constraint Programming*. Cambridge University Press, 2003. ([document](#)), [1.1](#)

- [AR15] G. E ARAUJO and H. M REPOLHO. « Optimizing the Airport Check-In Counter Allocation Problem ». *Journal of Transport Literature*, 9(4):15–19, December 2015. [1.5.1](#)
- [AS14] Gilles AUDEMARD and Laurent SIMON. « Lazy Clause Exchange Policy for Parallel SAT Solvers ». In Carsten SINZ and Uwe EGLY, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 197–205. Springer, 2014. [5.3.1](#)
- [BADD⁺20] Alejandro BARREDO ARRIETA, Natalia DÍAZ-RODRÍGUEZ, Javier DEL SER, Adrien BENNETOT, Siham TABIK, Alberto BARBADO, Salvador GARCIA, Sergio GIL-LOPEZ, Daniel MOLINA, Richard BENJAMINS, Raja CHATILA, and Francisco HERRERA. « Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI ». *Information Fusion*, 58:82–115, 2020. [2.3](#)
- [BBW17] U BENLIC, E. K BURKE, and J. R WOODWARD. « Breakout Local Search for the Multi-Objective Gate Allocation Problem ». *Computers & Operations Research*, 78:80–93, February 2017.
- [BC94] N BELDICEANU and E CONTEJEAN. « Introducing Global Constraints in CHIP ». *Mathematical and Computer Modelling*, 20(12):97–123, 1994. [5.2.3.4](#)
- [BCDP07] N BELDICEANU, M CARLSSON, S DEMASSEY, and T PETIT. « Global Constraint Catalogue: Past, Present and Future ». *Constraints*, 12(1):21–62, March 2007. ([document](#)), [1.2](#)
- [BCMT21a] Ryma BOUMAZOUZA, Fahima CHEIKH-ALILI, Bertrand MAZURE, and Karim TABIA. « ASTERYX: A Model-Agnostic SaT-basEd appRoach for sYmbolic and Score-Based eXplanations ». In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 120–129, Virtual Event Queensland Australia, October 2021. ACM. [2.3](#)
- [BCMT21b] Ryma BOUMAZOUZA, Fahima CHEIKH-ALILI, Bertrand MAZURE, and Karim TABIA. « ASTERYX: A Model-Agnostic SaT-basEd appRoach for sYmbolic and Score-Based eXplanations ». pages 120–129, October 2021.
- [BDFS22] Alexis BRUN, Daniel DELAHAYE, Eric FERON, and Sameer SAMEER. « Predicting Passenger Flow at Charles De Gaulle Airport Using Dense Neural Networks », 2022. [2.5.2](#)
- [BG10] Giuseppe BRUNO and Andrea GENOVESE. « A Mathematical Model for the Optimization of the Airport Check-In Service Problem ». *Electronic Notes in Discrete Mathematics*, 36:703–710, August 2010.
- [BGSS14] A BOURAS, M. A GHALEB, U. S SURYAHATMAJA, and A. M SALEM. « The Airport Gate Assignment Problem: A Survey ». *The Scientific World Journal*, 2014:e923859, November 2014. [1.4.1](#)

-
- [BHH⁺06] Christian BESSIERE, Emmanuel HEBRARD, Brahim HNICH, Zeynep KIZILTAN, and Toby WALSH. « Filtering Algorithms for the NValue Constraint ». *Constraints*, 11(4):271–293, November 2006. [5.2.3.5](#)
- [BHLS04] Frédéric BOUSSEMARY, Fred HEMERY, Christophe LECOUTRE, and Lakhdar SAIS. « Boosting Systematic Search by Weighting Constraints. ». pages 146–150, January 2004. [1.3.3.1](#)
- [Bie16] Armin BIERE. « Splat, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016 ». In Tomáš BALYŮ, Marijn HEULE, and Matti JÄRVISALO, editors, *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, volume B-2016-1 of *Department of Computer Science Series of Publications B*, pages 44–45. University of Helsinki, 2016. [5.3.1](#)
- [Bih90] Richard A. BIHR. « A Conceptual Solution to the Aircraft Gate Assignment Problem Using 0, 1 Linear Programming ». *Computers & Industrial Engineering*, 19(1-4):280–284, January 1990. [1](#)
- [BKN⁺09] Christian BESSIERE, George KATSIRELOS, Nina NARODYTSKA, Claude-Guy QUIMPER, and Toby WALSH. « Decompositions of All Different, Global Cardinality and Related Constraints ». In Craig BOUTILIER, editor, *Proceedings of IJCAI 2009*, pages 419–424, 2009. [5.2.1](#)
- [BLAP16] F. BOUSSEMARY, C. LECOUTRE, G. AUDEMARD, and C. PIETTE. « XCSP3: An Integrated Format for Benchmarking Combinatorial Constrained Problems ». *CoRR*, abs/1611.03398, 2016. [3.2](#)
- [BLAP20] F. BOUSSEMARY, C. LECOUTRE, G. AUDEMARD, and C. PIETTE. « XCSP3-core: A Format for Representing Constraint Satisfaction/Optimization Problems ». *CoRR*, abs/2009.00514, 2020. ([document](#)), [1.2](#), [3.2](#), [3.3.2](#)
- [Blo05] Wolfgang BLOCHINGER. « Towards Robustness in Parallel SAT Solving ». In Gerhard R. JOUBERT, Wolfgang E. NAGEL, Frans J. PETERS, Oscar G. PLATA, P. TIRADO, and Emilio L. ZAPATA, editors, *Parallel Computing: Current & Future Issues of High-End Computing, Proceedings of the International Conference ParCo 2005, 13-16 September 2005, Department of Computer Architecture, University of Malaga, Spain*, volume 33 of *John von Neumann Institute for Computing Series*, pages 301–308. Central Institute for Applied Mathematics, Jülich, Germany, 2005. [5.3.1](#)
- [Bre96] Leo BREIMAN. « Stacked Regressions ». *Machine Learning*, 24(1):49–64, July 1996. [2.2](#)
- [Bre01] Leo BREIMAN. « Random Forests ». *Machine Learning*, 45(1):5–32, 2001. [2.2](#)
- [BRYZ05] C. BESSIERE, J.-C. RÉGIN, R. YAP, and Y. ZHANG. « An Optimal Coarse-Grained Arc Consistency Algorithm ». *Artificial Intelligence*, 165(2):165–185, 2005. [1.3.1](#)
- [BSS94] Belaid BENHAMOU, Lakhdar SAIS, and Pierre SIEGEL. « Two Proof Procedures for a Cardinality Based Language in Propositional Calculus ». In Patrice ENJALBERT, Ernst W. MAYR, and Klaus W. WAGNER, editors, *Proceedings of STACS 1994*, pages 71–82. Springer, 1994. [5.2.1](#), [5.2.3.3](#)

- [Bur10] Chris J.C. BURGESS. « From RankNet to LambdaRank to LambdaMART: An Overview ». Technical report MSR-TR-2010-82, June 2010. [2.2](#)
- [BZF04] C. BESSIERE, B. ZANUTTINI, and C. FERNANDEZ. « Measuring Search Trees ». In *Proceedings of ECAI'04 Workshop on Modelling and Solving Problems with Constraints*, pages 31–40, 2004. [1.3.4](#)
- [CBN⁺19] Ramon Dalmau CODINA, Seddik BELKOURA, Herbert NAESENS, Franck BALLERINI, and Sebastian WAGNICK. « Improving the Predictability of Take-off Times with Machine Learning : A Case Study for the Maastricht Upper Area Control Centre Area of Responsibility ». 2019. [2.5.3](#)
- [CG16] Tianqi CHEN and Carlos GUESTRIN. « XGBoost: A Scalable Tree Boosting System ». In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, San Francisco California USA, August 2016. ACM. [7.6.1](#)
- [Chu96] Hon Wai CHUN. « Scheduling as a Multi-Dimensional Placement Problem ». *Engineering Applications of Artificial Intelligence*, 9(3):261–273, June 1996. [1.5.1](#)
- [CLFZ22] Kaiquan CAI, Yue LI, Yi-Ping FANG, and Yanbo ZHU. « A Deep Learning Approach for Flight Delay Prediction Through Time-Evolving Graphs ». *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11397–11407, August 2022. [2.5.3](#)
- [COC97] Mats CARLSSON, Greger OTTOSSON, and Björn CARLSON. « An Open-Ended Finite Domain Constraint Solver ». In Hugh GLASER, Pieter HARTEL, and Herbert KUCHEN, editors, *Programming Languages: Implementations, Logics, and Programs*, pages 191–206, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. [5.2.3.4](#)
- [Coo71] Stephen A. COOK. « The Complexity of Theorem-Proving Procedures ». In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. [5.2.2.1](#)
- [DAHS07] G DIEPEN, J.M. AKKER, J.A. HOOGEVEEN, and J.W. SMELTINK. « Using Column Generation for Gate Planning at Amsterdam Airport Schiphol ». January 2007.
- [DCS18] E. DEMIROVIC, G. CHU, and P. STUCKEY. « Solution-Based Phase Saving for CP: A Value-Selection Heuristic to Simulate Local Search Behavior in Complete Solvers ». In *Proceedings of CP'18*, pages 99–108, 2018. [1.3.3.2](#)
- [DDNP07] Ulrich DORNDORF, Andreas DREXL, Yury NIKULIN, and Erwin PESCH. « Flight Gate Scheduling: State-of-the-art and Recent Developments ». *Omega*, 35(3):326–334, June 2007. [1.4.1](#), [1.4.1](#)
- [Der16] Guillaume DERVAL. « Parallelization of Constraint Programming Using Embarrassingly Parallel Search », 2016. [5.3.1](#), [5.3.3.2](#)
- [DG02] Heidi E. DIXON and Matthew L. GINSBERG. « Inference Methods for a Pseudo-Boolean Satisfiability Solver ». In *AAAI'02*, pages 635–640, 2002. [5.2.1](#)

-
- [DGS20] G. S DAŞ, F GZARA, and T STÜTZLE. « A Review on Airport Gate Assignment Problems: Single versus Multi Objective Approaches ». *Omega*, 92:102146, April 2020. [1.4.1](#)
- [DHL⁺16] J. DEMEULENAERE, R. HARTERT, C. LECOUTRE, G. PEREZ, L. PERRON, J.-C. RÉGIN, and P. SCHAUS. « Compact-Table: Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets ». In *Proceedings of CP'16*, pages 207–223, 2016. [3.2](#)
- [DJP08] U DORNDORF, F JAEHN, and E PESCH. « Modelling Robust Flight-Gate Scheduling as a Clique Partitioning Problem ». *Transportation Science*, 42(3):292–301, August 2008. [1.4.1](#), [1.4.2](#)
- [DJP12] Ulrich DORNDORF, Florian JAEHN, and Erwin PESCH. « Flight Gate Scheduling with Respect to a Reference Schedule ». *Annals of Operations Research*, 194(1):177–187, April 2012. [3](#)
- [DLB08] Ines Lynce DANIEL LE BERRE. « CSP2SAT4J: A Simple CSP to SAT Translator ». In *Proceedings of the Second CSP Competition*, 2008. [5.2.1](#)
- [DLRZ04a] H. DING, A. LIM, B. RODRIGUES, and Y. ZHU. « Aircraft and Gate Scheduling Optimization at Airports ». In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of The*, Page 8 pp., Big Island, HI, USA, 2004. IEEE.
- [DLRZ04b] H DING, A LIM, B RODRIGUES, and Y ZHU. « New Heuristics for Over-Constrained Flight to Gate Assignments ». *Journal of the Operational Research Society*, 55(7):760–768, July 2004.
- [DLRZ05] H. DING, A. LIM, B. RODRIGUES, and Y. ZHU. « The Over-Constrained Airport Gate Assignment Problem ». *Computers & Operations Research*, 32(7):1867–1880, July 2005.
- [DN08] Andreas DREXL and Yury NIKULIN. « Multicriteria Airport Gate Assignment and Pareto Simulated Annealing ». *IIE Transactions*, 40(4):385–397, February 2008.
- [DS91] M DINCIBAS and H SIMONIS. « APACHE - A Constraint Based, Automated Stand Allocation System ». Automated Stand Allocation System Proc. Of Advanced Software Technology in Air Transport (ASTAIR'91) Royal Aeronautical Society, pages 267–282, October 1991. [1.4.1](#), [4.2.2](#), [4.4.2](#)
- [EM20] Ehsan ESMAELZADEH and Seyedmirsajad MOKHTARIMOUSAVI. « Machine Learning Approach for Flight Departure Delay Prediction and Analysis ». *Transportation Research Record: Journal of the Transportation Research Board*, 2674(8):145–159, August 2020. [2.5.3](#)
- [EN18] Jan ELFFERS and Jakob NORDSTRÖM. « Divide and Conquer: Towards Faster Pseudo-Boolean Solving ». In *Proceedings of IJCAI 2018*, pages 1291–1299, 2018. [5.2.1](#)
- [ES04] Niklas EÉN and Niklas SÖRENSON. « An Extensible SAT-solver ». In *Theory and Applications of Satisfiability Testing*, pages 502–518, 2004. [5.2.1](#)

- [FALM23] Thibault FALQUE, Gilles AUDEMARD, Christophe LECOUTRE, and Bertrand MAZURE. « Check-in Desk Scheduling Optimisation at CDG International Airport ». In *Doctoral Program of the 29th International Conference on Principles and Practice of Constraint Programming*, Toronto, Canada, August 2023. ([document](#)), 7.10
- [FFR16] Juliana FREIRE, Norbert FUHR, and Andreas RAUBER. « Reproducibility of Data-Oriented Experiments in e-Science (Dagstuhl Seminar 16041) ». *Dagstuhl Reports*, 6(1):108–159, 2016. 3.3.4
- [FGJ⁺07] A. FRISCH, M. GRUM, C. JEFFERSON, B. M. HERNANDEZ, and I. MIGUEL. « The Design of ESSENCE: A Constraint Language for Specifying Combinatorial Problems ». In *Proceedings of IJCAI'07*, pages 80–87, 2007. ([document](#)), 3.2
- [FH17] Nicholas FROSST and Geoffrey HINTON. « Distilling a Neural Network into a Soft Decision Tree ». November 2017. 2.3
- [FLMT22] Thibault FALQUE, Christophe LECOUTRE, Bertrand MAZURE, and Karim TABIA. « Optimisation Du Parcage Des Avions à l'aéroport Paris Charles de Gaulle. ». In *17es Journées Francophones de Programmation Par Contraintes (JFPC'22)*, Saint-Etienne, France, June 2022. ([document](#)), 7.10
- [FLMW21] T. FALQUE, C. LECOUTRE, B. MAZURE, and H. WATTEZ. « Descente Agressive de la Borne en Optimisation sous Contraintes ». In *Actes des 16es Journées Francophones de Programmation par Contraintes (JFPC'21)*, June 2021. ([document](#)), 5.1.1
- [FLMW22] Thibault FALQUE, Christophe LECOUTRE, Bertrand MAZURE, and Hugues WATTEZ. « Aggressive Bound Descent for Constraint Optimization ». In *Doctoral Program of the CP Conference*, Haïfa, Israel, August 2022. ([document](#)), 5.1.1, 5.1.5, 7.10
- [FLW23] Thibault FALQUE, Jean-Marie LAGNIEZ, and Romain WALLON. « Panoramyx: une bibliothèque pour le développement de solveurs de contraintes parallèles ». February 2023. 7.10
- [FMLT22] Thibault FALQUE, Bertrand MAZURE, Christophe LECOUTRE, and Karim TABIA. « Optimisation Du Parking Des Avions à Paris Charles de Gaulle ». In *23ème Congrès Annuel de La Société Française de Recherche Opérationnelle et d'Aide à La Décision*, Villeurbanne - Lyon, France, February 2022. INSA Lyon. ([document](#)), 7.10
- [FMT23a] Thibault FALQUE, Bertrand MAZURE, and Karim TABIA. « Predicting Off-Block Delays: A Case Study at Paris - Charles de Gaulle International Airport: ». In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence*, pages 180–189, Lisbon, Portugal, 2023. SCITEPRESS - Science and Technology Publications. ([document](#)), 7.10
- [FMT23b] Thibault FALQUE, Bertrand MAZURE, and Karim TABIA. « Prédire et expliquer les retards au décollage: Une étude de cas à l'aéroport international de Paris-Charles de Gaulle ». February 2023. ([document](#)), 7.10

-
- [FP17] J.-G FAGES and C PRUD'HOMME. « Making the First Solution Good! ». In *ICTAI 2017*, pages 1073–1077, Boston, MA, November 2017. IEEE. [1.3.3.2](#), [3.4.3](#)
- [Fri01] Jerome H. FRIEDMAN. « Greedy Function Approximation: A Gradient Boosting Machine. ». *The Annals of Statistics*, 29(5), October 2001.
- [Fri02] Jerome FRIEDMAN. « Stochastic Gradient Boosting ». *Computational Statistics & Data Analysis*, 38:367–378, February 2002. [2.2](#), [2.2](#)
- [FW22a] Thibault FALQUE and Romain WALLON. « Des Encodages PB Pour La Résolution de Problèmes CSP ». In *17es Journées Francophones de Programmation Par Contraintes (JFPC'22)*, Saint-Étienne, France, June 2022. ([document](#)), [7.10](#)
- [FW22b] Thibault FALQUE and Romain WALLON. « On PB Encodings for Constraint Problems ». In *Doctoral Program of the 28th International Conference on Principles and Practice of Constraint Programming*, Haifa, Israel, August 2022. ([document](#)), [5.2.5](#), [7.10](#)
- [FWW20] T. FALQUE, R. WALLON, and H. WATTEZ. « Metrics: Towards a Unified Library for Experimenting Solvers ». In *11th International Workshop on Pragmatics of SAT (POS'20)*, July 2020. ([document](#))
- [FWW21] T. FALQUE, R. WALLON, and H. WATTEZ. « Metrics : Mission Expérimentations ». In *Actes des 16es Journées Francophones de Programmation par Contraintes (JFPC'21)*, June 2021. ([document](#)), [3.3.4](#), [7.10](#)
- [GABG15] J. GUÉPET, R. ACUNA-AGOST, O. BRIANT, and J.P. GAYON. « Exact and Heuristic Approaches to the Airport Stand Allocation Problem ». *European Journal of Operational Research*, 246(2):597–608, 2015. [1.4.2](#)
- [Gav07] Marco GAVANELLI. « The Log-Support Encoding of CSP into SAT ». In Christian BESSIERE, editor, *CP 2007*, pages 815–822. Springer, 2007. [5.2.1](#)
- [GGDR18] Xiaojia GUO, Yael GRUSHKA-COCKAYNE, and Bert DE REYCK. « Forecasting Airport Transfer Passenger Flow Using Real-Time Data and Machine Learning ». *SSRN Electronic Journal*, 2018.
- [GL07] François GALEA and Bertrand LECUN. « Bobpp: A Framework for Exact Combinatorial Optimization Methods on Parallel Machines ». *PGCO'2007*, pages 779–, May 2007. [5.3](#)
- [Glo18] Gaël GLORIAN. « NACRE ». In *Solver Descriptions of XCSP3 Competition 2018*, 2018. [5.2.1](#)
- [GMN08] Ian P. GENT, Ian MIGUEL, and Peter NIGHTINGALE. « Generalised Arc Consistency for the AllDifferent Constraint: An Empirical Survey ». *Artificial Intelligence*, 172(18):1973–2000, 2008. [7](#)
- [GMR⁺18] Riccardo GUIDOTTI, Anna MONREALE, Salvatore RUGGIERI, Franco TURINI, Fosca GIANNOTTI, and Dino PEDRESCHI. « A Survey of Methods for Explaining Black Box Models ». *Acm Computing Surveys*, 51(5), August 2018. [2.3](#)

- [Gom58] Ralph E. GOMORY. « Outline of an Algorithm for Integer Solutions to Linear Programs ». *Bulletin of the American Mathematical Society*, pages 275–278, 1958. [5.2.1](#)
- [GPMC11] G. CHU, P. STUCKEY, M. GARCIA DE LA BANDA, and C. MEARS. « Symmetries and Lazy Clause Generation ». In *Proceedings of IJCAI’11*, pages 516–521, 2011.
- [GSCK00] C. GOMES, B. SELMAN, N. CRATO, and H. KAUTZ. « Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems ». *Journal of Automated Reasoning*, 24(1):67–100, 2000. [1.3.4](#)
- [Hak85] Armin HAKEN. « The Intractability of Resolution ». *Theoretical Computer Science*, 39:297–308, 1985. [5.2.1](#), [5.2.3.7](#)
- [HAP14] Thomas HERNDON, Michael ASH, and Robert POLLIN. « Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff ». *Cambridge Journal of Economics*, 38(2):257–279, 2014. [3.3.4](#)
- [HDM⁺11] Johan HUYSMANS, Karel DEJAEGER, Christophe MUES, Jan VANTHIENEN, and Bart BAESENS. « An Empirical Evaluation of the Comprehensibility of Decision Table, Tree and Rule Based Predictive Models ». *Decision Support Systems*, 51(1):141–154, 2011. [2.3](#)
- [HEKK19] Sara HOOKER, Dumitru ERHAN, Pieter-Jan KINDERMANS, and Been KIM. A Benchmark for Interpretability Methods in Deep Neural Networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2019. [2.3](#)
- [Hoo88] J. N. HOOKER. « Generalized Resolution and Cutting Planes ». *Annals of Operations Research*, 12(1):217–239, 1988. [5.2.1](#)
- [HS97] Sepp HOCHREITER and Jürgen SCHMIDHUBER. « Long Short-Term Memory ». *Neural Computation*, 9(8):1735–1780, November 1997. [2.5.3](#), [7.6.2](#)
- [IEM21a] Alaa IBRAHEM, Heba ELBEH, and Hamdy M MOUSA. « A Comparative Analysis of Models for Predicting Airline Arrival Delays ». Page 5, 2021. [2.5.3](#)
- [IEM21b] Alaa IBRAHEM, Heba ELBEH, and Hamdy M MOUSA. « A Comparative Analysis of Models for Predicting Airline Arrival Delays ». Page 5, 2021.
- [IM21] Yacine IZZA and João MARQUES-SILVA. « On Explaining Random Forests with SAT ». In Zhi-Hua ZHOU, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2584–2591. ijcai.org, 2021. [2.3](#)
- [IMM18] Alexey IGNATIEV, Antonio MORGADO, and Joao MARQUES-SILVA. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In Olaf BEYERSDORFF and Christoph M. WINTERSTEIGER, editors, *Theory and Applications of Satisfiability Testing – SAT 2018*, volume 10929, pages 428–437. Springer International Publishing, Cham, 2018. [2.3](#)

-
- [Jae10] Florian JAEHN. « Solving the Flight Gate Assignment Problem Using Dynamic Programming ». *Zeitschrift für Betriebswirtschaft*, 80(10):1027–1039, October 2010.
- [JKT16] Philippe JÉGOU, Hanan KANSO, and Cyril TERRIOUX. « Towards a Dynamic Decomposition of CSPs with Separators of Bounded Size ». In Michel RUEHER, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 298–315. Springer, 2016. [5.3.4](#)
- [KCBM16] Young Jin KIM, Sun CHOI, Simon BRICENO, and Dimitri MAVRIS. « A Deep Learning Approach to Flight Delay Prediction ». In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–6, Sacramento, CA, USA, September 2016. IEEE.
- [KKS⁺19] Walid KHERIJI, Sami KRAIEM, Guilhem SANMARTY, Ghazaleh KHODABANDELOU, and Fouad Hadj SELEM. « Prediction of Bus Passenger Flow Using Deep Learning ». 2019. [2.5.2](#)
- [KMF⁺17] Guolin KE, Qi MENG, Thomas FINLEY, Taifeng WANG, Wei CHEN, Weidong MA, Qiwei YE, and Tie-Yan LIU. « LightGBM: A Highly Efficient Gradient Boosting Decision Tree ». In I. GUYON, U. Von LUXBURG, S. BENGIO, H. WALLACH, R. FERGUS, S. VISHWANATHAN, and R. GARNETT, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. [2.5.3](#), [7.5.2](#), [7.6.1](#)
- [KPD18] Yang-Min KIM, Jean-Baptiste POLINE, and Guillaume DUMAS. « Experimenting with Reproducibility: A Case Study of Robustness in Bioinformatics ». *GigaScience*, 7(7):giy077, July 2018. [3.3.4](#)
- [LBP10] Daniel LE BERRE and Anne PARRAIN. « The SAT4J Library, Release 2.2, System Description ». *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010. [5.2.1](#), [5.2.4.1](#), [5.3.1](#), [5.3.4](#)
- [LC17] Lijuan LIU and Rung-Ching CHEN. « A Novel Passenger Flow Prediction Model Using Deep Learning Methods ». *Transportation Research Part C: Emerging Technologies*, 84:74–91, November 2017. [2.5.2](#)
- [LCG12] Yin LOU, Rich CARUANA, and Johannes GEHRKE. « Intelligible Models for Classification and Regression ». In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–158. ACM, 2012.
- [LCGN18] Zhongbin LI, Haiyan CHEN, Jiaming GE, and Kunpeng NING. « An Airport Scene Delay Prediction Method Based on LSTM ». In Guojun GAN, Bohan LI, Xue LI, and Shuliang WANG, editors, *Advanced Data Mining and Applications*, pages 160–169, Cham, 2018. Springer International Publishing.
- [LCKLD17] B. LE CHARLIER, M. T. KHONG, C. LECOUTRE, and Y. DEVILLE. « Automatic Synthesis of Smart Table Constraints by Abstraction of Table Constraints ». In *Proceedings of IJCAI’17*, pages 681–687, 2017.

- [Lec09] C. LECOUTRE. *Constraint Networks: Techniques and Algorithms*. ISTE/Wiley, 2009. ([document](#)), [1.1](#), [1.2](#), [7.10](#)
- [Lec11] C LECOUTRE. « STR2: Optimized Simple Tabular Reduction for Table Constraints ». *Constraints : an international journal*, 16(4):341–371, 2011. [3.2](#)
- [Lec21] C. LECOUTRE. *ACE: A Generic Constraint Solver*. Technical Report. v1on CoRR, to appear, October 2021.
- [Lec23] C. LECOUTRE. « ACE, a Generic Constraint Solver ». *CoRR*, abs/2302.05405, 2023. ([document](#)), [3.2](#), [5.3.4](#)
- [LEJ⁺21] Kyungeun LEE, Moonjung EO, Euna JUNG, Yoonjin YOON, and Wonjong RHEE. « Short-Term Traffic Prediction With Deep Neural Networks: A Survey ». *IEEE Access*, 9:54739–54756, 2021. [2.5.2](#)
- [LFBSK17] Ludovic LE FRIoux, Souheib BAARIR, Julien SOPENA, and Fabrice KORDON. « PaInleSS: A Framework for Parallel SAT Solving ». In *Proceedings of SAT 2017*, pages 233–250, 2017. [5.3](#)
- [LHC03] Richard D. LAWRENCE, Se June HONG, and Jacques CHERRIER. « Passenger-Based Predictive Modeling of Airline No-Show Rates ». In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 397–406, Washington, D.C., August 2003. ACM. [2.5.2](#)
- [Li08] C LI. « Airport Gate Assignment: New Model and Implementation ». *arXiv:0811.1618 [cs]*, November 2008.
- [Li09] C LI. « Airport Gate Assignment A Hybrid Model and Implementation ». *arXiv:0903.2528 [cs]*, March 2009.
- [Li10] Ping LI. « Robust Logitboost and Adaptive Base Class (ABC) Logitboost ». In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI’10, pages 302–311, Arlington, Virginia, USA, 2010. AUAI Press. [2.2](#)
- [LL17] Scott M. LUNDBERG and Su-In LEE. « A Unified Approach to Interpreting Model Predictions ». In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, pages 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. [2.3](#), [2.3](#), [7.6.1](#)
- [LLL⁺23] Lin LIN, Xiaochen LIU, Xiaohua LIU, Tao ZHANG, and Yang CAO. « A Prediction Model to Forecast Passenger Flow Based on Flight Arrangement in Airport Terminals ». *Energy and Built Environment*, 4(6):680–688, December 2023. [2.5.2](#)
- [LLY15] C. LECOUTRE, C. LIKITVIVATANAVONG, and R. YAP. « STR3: A Path-Optimal Filtering Algorithm for Table Constraints ». *Artificial Intelligence*, 220:1–27, 2015. [3.2](#)
- [LM22] T. R. LALITA and G. S. R. MURTHY. « The Airport Check-in Counter Allocation Problem: A Survey », August 2022. [1.5.1](#)

-
- [LMV21] J. L'ORTYE, M. MITICI, and H.G. VISSER. « Robust Flight-to-Gate Assignment with Landside Capacity Constraints ». *Transportation Planning and Technology*, 44(4):356–377, May 2021.
- [LRZ05] A. LIM, B. RODRIGUES, and Y. ZHU. « Airport Gate Scheduling with Time Windows ». *Artificial intelligence review*, 24(1):5–31, September 2005. [1](#)
- [LS20] C. LECOUTRE and N. SZCZEPANSKI. « PyCSP3: Modeling Combinatorial Constrained Problems in Python ». *CoRR*, abs/2009.00326, 2020. ([document](#)), [3.2](#), [7.10](#)
- [LSTV09] C. LECOUTRE, L. SAIS, S. TABARY, and V. VIDAL. « Reasoning from Last Conflict(s) in Constraint Programming ». *Artificial Intelligence*, 173(18):1592–1614, 2009. [18](#)
- [LSZ93] M. LUBY, A. SINCLAIR, and D. ZUCKERMAN. « Optimal Speedup of Las Vegas Algorithms ». *Information Processing Letters*, 47(4):173–180, 1993. [5.1.1](#)
- [LYL21a] H LI, M YIN, and Z LI. « Failure Based Variable Ordering Heuristics for Solving CSPs ». In L. D MICHEL, editor, *CP 21*, volume 210, pages 9:1–9:10, 2021. [1.3.3.1](#)
- [LYL21b] Hongbo LI, Minghao YIN, and Zhanshan LI. « Failure Based Variable Ordering Heuristics for Solving CSPs ». In Laurent D. MICHEL, editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:10, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [Mac77a] A. K. MACKWORTH. « Consistency in Networks of Relations ». *Artificial Intelligence*, 8(1):99–118, 1977. ([document](#)), [1.2](#)
- [Mac77b] A. K. MACKWORTH. « On Reading Sketch Maps ». In *Proceedings of IJCAI'77*, pages 598–606, 1977. [1.3.1](#), [1.3.1](#)
- [MHRS08] D. MARKOVIC, T. HAUF, P. RÖHNER, and U. SPEHR. « A Statistical Study of the Weather Impact on Punctuality at Frankfurt Airport ». *Meteorological Applications*, 15(2):293–303, June 2008. [2.5.3](#)
- [Mil19] Tim MILLER. « Explanation in Artificial Intelligence: Insights from the Social Sciences ». *Artificial Intelligence*, 267:1–38, 2019. [2.3](#)
- [MJB⁺20] Philippe MONMOUSSEAU, Gabriel JARRY, Florian BERTOSIO, Daniel DELAHAYE, and Marc HOUALLA. « Predicting Passenger Flow at Charles De Gaulle Airport Security Checkpoints ». In *2020 International Conference on Artificial Intelligence and Data Analytics for Air Transportation (AIDA-AT)*, pages 1–9, Singapore, February 2020. IEEE. [2.5.2](#), [2.6.4](#)
- [ML13] Tarek MENOUEUR and Bertrand LECUN. « A Parallelization Mixing OR-Tools/Gecode Solvers on Top of the Bobpp Framework ». In *Proceedings - 2013 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013*, pages 242–246, October 2013. [5.3](#)

- [MM85] R. S. MANGOUBI and D. F. X. MATHAISEL. « Optimizing Gate Assignments at Airport Terminals ». *Transportation Science*, 19(2):173–188, 1985. [1.4.1](#)
- [MML10] Ruben MARTINS, Vasco M. MANQUINHO, and Inês LYNCE. « Improving Search Space Splitting for Parallel SAT Solving ». In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 1*, pages 336–343. IEEE Computer Society, 2010. [5.3.1](#)
- [MMZ⁺01] Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG, and Sharad MALIK. « Chaff: Engineering an Efficient SAT Solver ». In *Proceedings of the 38th Annual Design Automation Conference, DAC '01*, pages 530–535, New York, NY, USA, 2001. ACM. [5.2.1](#)
- [Mon74] U MONTANARI. « Network of Constraints : Fundamental Properties and Applications to Picture Processing ». *Information Science*, 7:95–132, 1974. [1.2](#), [7.10](#)
- [MRR16] Arnaud MALAPERT, Jean-Charles RÉGIN, and Mohamed REZGUI. « Embarassingly Parallel Search in Constraint Programming ». *Journal of Artificial Intelligence Research*, 57:421–464, 2016. [5.3.1](#)
- [MS99] Joao MARQUES-SILVA and Karem A. SAKALLAH. « GRASP: A Search Algorithm for Propositional Satisfiability ». *IEEE Trans. Computers*, pages 220–227, 1999. [5.2.1](#)
- [NG17] Rahul NIGAM and K. GOVINDA. « Cloud Based Flight Delay Prediction Using Logistic Regression ». In *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, pages 662–667, December 2017. [2.5.3](#)
- [NGSH22] Nandini NAGARAJ, Harinahalli Lokesh GURURAJ, Beekanahalli Harish SWATHI, and Yu-Chen HU. « Passenger Flow Prediction in Bus Transportation System Using Deep Learning ». *Multimedia Tools and Applications*, 81(9):12519–12542, April 2022. [2.5.2](#)
- [NI20] Hidetomo NABESHIMA and Katsumi INOUE. « Reproducible Efficient Parallel SAT Solving ». In Luca PULINA and Martina SEIDL, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 123–138. Springer, 2020. [5.3.1](#)
- [NMBS18] Vijayarangan NATARAJAN, Swaminathan MEENAKSHISUNDARAM, Gautham BALASUBRAMANIAN, and Shubham SINHA. « A Novel Approach: Airline Delay Prediction Using Machine Learning ». In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1081–1086, December 2018. [2.5.3](#)
- [Nor15] Jakob NORDSTRÖM. « On the Interplay between Proof Complexity and SAT Solving ». *ACM SIGLOG News*, 2(3):19–44, August 2015. [5.2.1](#)
- [NSB⁺07] N. NETHERCOTE, P. STUCKEY, R. BECKET, S. BRAND, G. DUCK, and G. TACK. « MiniZinc: Towards a Standard CP Modelling Language ». In *Proceedings of CP'07*, pages 529–543, 2007. ([document](#)), [3.2](#)

-
- [Osc12] OSCAR TEAM. « OscaR: Scala in OR », 2012. ([document](#)), 3.2
- [P. 99] P. VAN HENTENRYCK. *The OPL Optimization Programming Language*. The MIT Press, 1999. ([document](#)), 3.2
- [PF07] Dirk PILAT and Yukiko FUKASAKU. « OECD Principles and Guidelines for Access to Research Data from Public Funding ». *Data Science Journal*, 6:4–11, June 2007. 3.3.4
- [PF22] Laurent PERRON and Vincent FURNON. « OR-Tools ». Google, 2022-08-11, 2022. 5.3
- [PFL16] C. PRUD'HOMME, J.-G. FAGES, and X. LORCA. « Choco-Solver, TASC, INRIA Rennes, LINA, Cosling S.A. ». 2016. ([document](#)), 3.2, 5.2.1, 5.3, 5.3.1, 5.3.4, 7.10
- [PKB14] V. PREM KUMAR and Michel BIERLAIRE. « Multi-Objective Airport Gate Assignment Problem in Planning and Operations: MULTI-OBJECTIVE AIRPORT GATE ASSIGNMENT ». *Journal of Advanced Transportation*, 48(7):902–926, November 2014. 2
- [RB14] Juan Jose REBOLLO and Hamsa BALAKRISHNAN. « Characterization and Prediction of Air Traffic Delays ». *Transportation Research Part C: Emerging Technologies*, 44:231–241, July 2014. 2.5.3
- [RDR07] Matthew RICHARDSON, Ewa DOMINOWSKA, and Robert RAGNO. « Predicting Clicks: Estimating the Click-through Rate for New Ads ». In *Proceedings of the 16th International Conference on World Wide Web, WWW '07*, pages 521–530, New York, NY, USA, 2007. Association for Computing Machinery. 2.2
- [Rég94] Jean-Charles RÉGIN. « A Filtering Algorithm for Constraints of Difference in CSPs ». In Barbara HAYES-ROTH and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, pages 362–367. AAAI Press / The MIT Press, 1994. 7
- [Rég96] Jean-Charles RÉGIN. « Generalized Arc Consistency for Global Cardinality Constraint ». In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1, AAAI'96*, pages 209–215, Portland, Oregon, 1996. AAAI Press. 5.2.3.3
- [RG15] K. V. RASHMI and Ran GILAD-BACHRACH. « DART: Dropouts Meet Multiple Additive Regression Trees ». *arXiv:1505.01866 [cs, stat]*, May 2015. 2.2, 2.6
- [Rou11] Olivier ROUSSEL. « Controlling a Solver Execution: The Runsolver Tool ». *Journal on Satisfiability, Boolean Modeling and Computation*, 7:139–144, 2011. 3.3.4
- [Rou12] Olivier ROUSSEL. « Description of Ppfolio 2012 ». In *Proc. SAT Challenge*, Page 46, 2012. 5.3.1
- [RR10] Carmen REINHART and Kenneth ROGOFF. « Growth in a Time of Debt ». *American Economic Review*, 100:573–78, May 2010. 3.3.4

- [RRM13] J.-C. RÉGIN, M. REZGUI, and A. MALAPERT. « Embarrassingly Parallel Search ». In *Proceedings of CP'13*, pages 596–610, 2013. [5.3](#), [5.3.1](#)
- [RSG16] Marco Tulio RIBEIRO, Sameer SINGH, and Carlos GUESTRIN. « ”Why Should I Trust You?”: Explaining the Predictions of Any Classifier ». In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. [2.3](#), [2.3](#)
- [RvW06] F. ROSSI, P. VAN BEEK, and T. WALSH, editors. *Handbook of Constraint Programming*. Elsevier, 2006. ([document](#)), [1.1](#), [7.10](#)
- [SBF10] P. STUCKEY, R. BECKET, and J. FISCHER. « Philosophy of the MiniZinc Challenge ». *Constraints*, 15(3):307–316, 2010. ([document](#)), [3.2](#)
- [SCD18] Andy SHIH, Arthur CHOI, and Adnan DARWICHE. « A Symbolic Approach to Explaining Bayesian Network Classifiers ». In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 5103–5111, Stockholm, Sweden, 2018. AAAI Press. [2.3](#)
- [SDC19] Andy SHIH, Adnan DARWICHE, and Arthur CHOI. « Verifying Binarized Neural Networks by Angluin-Style Learning ». In Mikoláš JANOTA and Inês LYNCE, editors, *Theory and Applications of Satisfiability Testing – SAT 2019*, pages 354–370, Cham, 2019. Springer International Publishing. [2.3](#)
- [SF94] Daniel SABIN and Eugene C. FREUDER. « Contradicting Conventional Wisdom in Constraint Satisfaction ». In Alan BORNING, editor, *Principles and Practice of Constraint Programming*, pages 10–20, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. [1.3.2](#)
- [SHG⁺22] Sebastian SCHLAG, Tobias HEUER, Lars GOTTESBÜREN, Yaroslav AKHREMTSEV, Christian SCHULZ, and Peter SANDERS. « High-Quality Hypergraph Partitioning ». *ACM Journal of Experimental Algorithmics*, March 2022. [5.3.3.3](#)
- [Sim07] H SIMONIS. « Models for Global Constraint Applications ». *Constraints : an international journal*, 12(1):63–92, March 2007. [1.4.1](#), [4.2.2](#), [4.4.2](#)
- [SMMFPMM21] P.M. SCALA, M.A. MUJICA MOTA, R.S. FELIX PATRON, and A. MURRIETA MENDOZA. « Airport Passenger Flow Prediction Using Simulation Data Farming and Machine Learning ». 2021. [2.5.1](#)
- [Tan21] Yuemin TANG. « Airline Flight Delay Prediction Using Machine Learning Models ». In *2021 5th International Conference on E-Business and Internet*, pages 151–154, Singapore Singapore, October 2021. ACM. [2.5.3](#)
- [TTKB09] Naoyuki TAMURA, Akiko TAGA, Satoshi KITAGAWA, and Mutsunori BANBARA. « Compiling Finite Linear CSP into SAT ». *Constraints An Int. J.*, 14(2):254–272, 2009. [5.2.1](#), [5.2.2.1](#), [5.2.3.2](#)
- [VAA⁺17] V. VENKATESH, A. ARYA, Pooja AGARWAL, S. LAKSHMI, and Sanjay BALANA. « Iterative Machine and Deep Learning Approach for Aviation Delay Prediction ». *2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)*, 2017. [2.5.3](#)

-
- [van01] Willem-Jan VAN HOEVE. « The Alldifferent Constraint: A Survey ». *CoRR*, cs.PL/0105015, May 2001. [7](#)
- [VB88] Godelieve VANDERSTRAETEN and Michel BERGERON. « Automatic Assignment of Aircraft to Gates at a Terminal ». *Computers & Industrial Engineering*, 14(1):15–25, January 1988. [2](#)
- [VHK06] W.-J VAN HOEVE and I KATRIEL. Global Constraints. In *Foundations of Artificial Intelligence*, volume 2, pages 169–208. Elsevier, 2006. ([document](#)), [1.2](#)
- [vLGB⁺18] Ulrike von LUXBURG, Isabelle GUYON, Samy BENGIO, Hanna WALLACH, Rob FERGUS, S. V. N. VISHWANATHAN, Roman GARNETT, and Neural Information Processing Systems FOUNDATION, editors. *Advances in Neural Information Processing Systems 30: 31st Annual Conference on Neural Information Processing Systems (NIPS 2017): Long Beach, California, USA, 4-9 December 2017*. Curran Associates, Inc, Red Hook, NY, 2018.
- [VLS17] H. VERHAEGHE, C. LECOUTRE, and P. SCHAUS. « Extending Compact-Table to Negative and Short Tables ». In *Proceedings of AAAI’17*, pages 3951–3957, 2017. [3.2](#)
- [VP17] J. VION and S. PIECHOWIAK. « Une Simple Heuristique Pour Rapprocher DFS et LNS pour les COP ». In *Proceedings of JFPC’17*, pages 39–45, 2017. [1.3.3.2](#)
- [vv06] Nico M. VAN DIJK and Erik VAN DER SLUIS. « Check-in Computation and Optimization by Simulation and IP in Combination ». *European Journal of Operational Research*, 171(3):1152–1168, June 2006.
- [Wal99] T. WALSH. « Search in a Small World ». In *Proceedings of IJCAI’99*, pages 1172–1177, 1999. [1.3.4](#)
- [Wal00] Toby WALSH. « SAT v CSP ». In Rina DECHTER, editor, *Proceedings of CP 2000*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000. [5.2.1](#), [5.2.2.1](#)
- [WSW03] WANG, SCHAEFER, and WOJCIK. « Flight Connections and Their Impacts on Delay Propagation ». In *22nd Digital Avionics Systems Conference Proceedings (Cat No 03CH37449) DASC-03*, pages 5.B.4–5.1, Indianapolis, IN, USA, 2003. IEEE. [2.5.3](#)
- [XSL08] Ning XU, Lance SHERRY, and Kathryn Blackmond LASKEY. « Multifactor Model for Predicting Delays at U.S. Airports ». *Transportation Research Record: Journal of the Transportation Research Board*, 2052(1):62–71, January 2008.
- [YDS⁺20] YOGITA BORSE, DHARUVIN JAIN, SHREYASH SHARMA, VIRAL VORA, AAKASH ZAVERI, and K J SOMAIYA COLLEGE OF ENGINEERING. « Flight Delay Prediction System ». *International Journal of Engineering Research and*, V9(03):IJERTV9IS030148, March 2020. [2.5.3](#)

- [YSC02] Shangyao YAN, Chi-Yuan SHIEH, and Miawjane CHEN. « A Simulation Framework for Evaluating Airport Gate Assignments ». *Transportation Research Part A: Policy and Practice*, 36(10):885–898, December 2002. [1](#)
- [YT07] Shangyao YAN and Ching-Hui TANG. « A Heuristic Approach for Airport Gate Assignments for Stochastic Flight Delays ». *European Journal of Operational Research*, 180(2):547–567, July 2007. [1](#)
- [YTC04] S YAN, C.-H TANG, and M CHEN. « A Model and a Solution Algorithm for Airport Common Use Check-in Counter Assignments ». *Transportation Research Part A: Policy and Practice*, 38(2):101–125, February 2004. [1.5.1](#)
- [YTC14] S YAN, C.-H. TANG, and Jun-Hsiu CHEN. « Common-Use Check-in Counter Reassignments with a Variable Number of Service Lines and Variable Length of Time Window ». *Journal of the Chinese Institute of Engineers*, 37(5):643–658, July 2014.
- [YZL⁺21] Jia YI, Honghai ZHANG, Hao LIU, Gang ZHONG, and Guiyi LI. « Flight Delay Classification Prediction Based on Stacking Algorithm ». *Journal of Advanced Transportation*, 2021:1–10, August 2021. [2.5.3](#)
- [ZBH96] Hantao ZHANG, Maria Paola BONACINA, and Jieh HSIANG. « PSATO: A Distributed Propositional Prover and Its Application to Quasigroup Problems ». *Journal of Symbolic Computation*, 21(4):543–560, 1996. [5.3.1](#)
- [ZKF17] N. F. ZHOU, H. KJELLERSTRAND, and J. FRUHMANN. *Constraint Solving and Planning with Picat*. Springer, 2017. ([document](#)), [3.2](#)

Résumé

L'industrie de l'aviation joue un rôle essentiel dans notre monde globalisé, permettant à des millions de personnes de voyager, de faire des affaires. Les aéroports, avec l'Aéroport Charles de Gaulle de Paris (CDG) comme exemple typique, incarnent les complexités des hubs mondiaux. Gérant des millions de passagers et se connectant à des centaines de destinations dans le monde entier, CDG est classé comme le deuxième aéroport le plus fréquenté d'Europe. Malgré son envergure, il n'y a pas de projets immédiats pour de nouveaux terminaux, soulignant l'importance de l'optimisation pour la gestion efficace des passagers et des ressources. Cette nécessité est encore amplifiée par la résurgence du trafic aérien après les perturbations liées à la COVID-19.

Pour répondre aux complexités des aéroports comme CDG, des solutions innovantes sont nécessaires. Les méthodes traditionnelles, basées sur des technologies plus anciennes, pourraient peiner à répondre aux défis actuels, surtout étant donné l'aspect imprévisible de l'aviation. Deux approches se démarquent: la programmation par contraintes, apte à gérer des problèmes combinatoires et d'optimisations, et l'apprentissage automatique, reconnu pour sa puissance prédictive.

Ce manuscrit se déroule en trois parties distinctes sur sept chapitres. La première partie aborde la programmation par contraintes et ses applications dans les opérations aéroportuaires, en particulier pour l'allocation des comptoirs d'enregistrement et des parkings avions. À travers des évaluations expérimentales, l'utilité des nouveaux modèles est démontrée, mettant en avant notre outil Metrics pour l'analyse des résultats. Cette section offre également un aperçu des méthodes de résolution génériques pour les problèmes d'allocation.

La dernière partie du manuscrit s'oriente vers le rôle de l'apprentissage automatique dans l'optimisation des opérations aéroportuaires. Elle se concentre sur la prévision du nombre de passagers à mobilité réduite (PMR) et la prédiction des retards "au bloc" des vols. Compte tenu de l'importance de ces deux aspects, le potentiel de l'apprentissage automatique pour prévoir et prévenir les défis est mis en évidence.

À mesure que le paysage de l'aviation évolue, l'optimisation des opérations, en particulier dans les hubs comme CDG, a des implications majeures. Ce manuscrit trace une voie pour exploiter la programmation par contraintes et l'apprentissage automatique afin de renforcer la résilience et l'efficacité des opérations aéroportuaires, garantissant leur capacité à servir un monde interconnecté.

Mots-clés: programmation par contraintes, optimisation, apprentissage automatique.

Abstract

The aviation industry serves as a critical connector in today's globalized world, enabling millions to travel, conduct business, and engage with diverse cultures. Key to this connectivity is the infrastructure provided by airports, with Paris Charles de Gaulle Airport (CDG) epitomizing the complexities inherent in global hubs. Handling millions of passengers and connecting to hundreds of destinations worldwide, CDG ranks as Europe's second busiest airport. Despite its vast scale, there are no imminent plans for new terminals, emphasizing the importance of optimization for efficient passenger and resource management. This urgency is further heightened by the resurgence in air traffic post-COVID-19 disruptions.

Addressing the complexities of mega hubs like CDG necessitates innovative solutions. Traditional methods, anchored in older technologies, might struggle to meet present-day challenges, especially given the aviation industry’s unpredictability. Two promising approaches emerge in the modern context: constraint programming, adept at managing large-scale problems, and machine learning, renowned for its predictive prowess.

This manuscript unfolds in three distinct parts across seven chapters. The initial part delves into constraint programming and its applications in airport operations, particularly in the allocation of check-in desks and aircraft stands. Through experimental evaluations, the utility of new models is demonstrated, showcasing our Metrics tool’s role in analyzing experimental results. This section also offers insights into generic resolution methods for allocation problems.

The latter part of the manuscript pivots to machine learning’s role in optimizing airport operations. It focuses on forecasting the number of passengers with reduced mobility (PRM) and predicting off-block flight delays. Given the importance of both these aspects in maintaining operational efficiency, machine learning’s potential in predicting and preempting challenges is underscored.

As the aviation landscape continually shifts, the optimization of operations, especially in hubs like CDG, has far-reaching implications. This manuscript charts a course towards harnessing constraint programming and machine learning to foster resilience and efficiency in airport operations, ensuring they remain adept at serving a connected world.

Keywords: constraint programming, optimization, machine learning.

