Titre: Title:	Apprentissage machine et génération de colonnes
Auteur: Author:	Mouad Morabit
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Morabit, M. (2022). Apprentissage machine et génération de colonnes [Ph.D. thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10709/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Apprentissage machine et génération de colonnes

MOUAD MORABIT
Département de mathématiques et de génie industriel

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

Décembre 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Apprentissage machine et génération de colonnes

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

François SOUMIS, président
Guy DESAULNIERS, membre et directeur de recherche
Andrea LODI, membre et codirecteur de recherche
Quentin CAPPART, membre
Marco LÜBBECKE, membre externe

DÉDICACE

A ma très chère mère,
mon plus grand amour, ma plus grande fierté et la femme de ma vie,
je t'aime.
...
A mon père, décédé trop tôt,
tu me manques.
...
A mes soeurs, ma nièce et mes neveux,
je vous aime tous.

REMERCIEMENTS

Je souhaiterais adresser mes remerciements les plus sincères à mes directeurs de recherche, M. Guy Desaulniers et M. Andrea Lodi, pour leur disponibilité tout au long de cette thèse, leurs judicieux conseils ainsi que leur encadrement exceptionnel. Le travail avec eux était très agréable et je me considère très chanceux d'être un de leurs étudiants. Je remercie particulièrement Guy, pour m'avoir ouvert les portes du GERAD. Je tiens également à le remercier pour sa confiance, son écoute, son soutien constant et ses encouragements.

Je tiens à exprimer ma gratitude à François Lessard, Benoît Rochefort et Khalid laaziri pour leur support technique et soutien informatique.

Mes remerciements vont aussi à tous mes amis et mes collègues au GERAD que j'ai côtoyé durant ces années, et avec qui j'ai vécu des moments mémorables. Je remercie notamment Rachid, Adil, Abderrahmane, Yassine, Mohamed Amine, Ilyas, Tayeb, El Mehdi.

J'adresse les remerciements les plus chaleureux à ma mère Malika et mes soeurs Rajaâ, Mouna et Asmaa pour leur amour et leur soutien continu.

Je tiens à exprimer toute ma gratitude aux membres du jury pour avoir accepté de juger ce travail.

Finalement, je remercie toutes les personnes qui ont contribué de près ou de loin à l'accomplissement de cette thèse.

RÉSUMÉ

Au cours des dernières années, l'apprentissage machine (ML - *Machine Learning*) a connu un développement sans précédent, surtout après l'émergence de l'apprentissage profond. Le domaine a connu des avancées remarquables qui ont suscité beaucoup d'intérêt auprès des chercheurs de plusieurs disciplines, incluant la communauté de recherche opérationnelle. Récemment, plusieurs travaux cherchant à exploiter les méthodes ML pour résoudre des problèmes d'optimisation combinatoire, ou bien pour accélérer le processus de résolution, ont vu le jour. Ceci a mené au développement de nombreuses méthodes qui ont montré un grand potentiel sur différents problèmes d'optimisation.

Dans cette thèse, nous nous intéressons particulièrement à la méthode de génération de colonnes (GC), qui est une méthode d'optimisation utilisée pour résoudre efficacement des programmes linéaires comportant un grand nombre de variables (colonnes). Il s'agit d'une méthode itérative qui commence avec un petit sous-ensemble de variables du programme linéaire original, et génère de nouvelles colonnes au besoin jusqu'à atteindre une solution optimale. La méthode a prouvé son efficacité sur plusieurs problèmes industriels, tels que des problèmes de planification, d'ordonnancement, et de transport.

Nous présentons dans cette thèse trois approches différentes pour intégrer l'apprentissage machine dans le contexte de la GC, soit pour accélérer l'optimisation et réduire les temps de calcul des méthodes exactes, ou bien pour développer des heuristiques qui produisent de bonnes solutions en des temps raisonnables. L'objectif général est de tirer profit des données qui sont collectées lors des exécutions antérieures afin d'obtenir des modèles de prédiction, qui seront capables de guider l'optimisation et de réduire l'espace de recherche.

Dans le premier sujet, nous proposons une méthode de sélection de colonnes qui, à chaque itération de la GC, applique un modèle de prédiction pour sélectionner un sous-ensemble des colonnes générées. L'objectif est de réduire le temps de calcul consacré à la réoptimisation du problème maître restreint en sélectionnant les colonnes les plus prometteuses. Nous traitons ce problème en utilisant de l'apprentissage supervisé, un classifieur binaire est utilisé pour classer les colonnes générées dans une des deux classes : choisie ou rejetée. La méthode est considérée générale et peut être appliquée à divers problèmes d'optimisation, du moment qu'un grand nombre de colonnes est généré à chaque itération. Nous démontrons l'efficacité de cette approche sur deux applications différentes, à savoir le problème de tournées de véhicules avec fenêtres de temps et le problème d'horaires d'équipages et de véhicules en transport public. Les résultats ont montré un gain en temps de calcul allant jusqu'à 30%.

Dans le deuxième sujet, nous nous intéressons aux problèmes résolus avec la GC où le sous-problème est défini sur un graphe et correspond à un problème de plus court chemin avec contraintes de ressources ou une de ses variantes, ce qui est le cas de plusieurs problèmes de planification et de transport. Nous proposons une heuristique pour la résolution du sous-problème, qui consiste à réduire la taille du réseau sous-jacent en sélectionnant les arcs qui ont de grandes probabilités de faire partie d'une solution optimale du problème maître. Un modèle de classification binaire est entraîné sur les données des instances précédemment résolues. Le réseau réduit obtenu à partir des prédictions du modèle est utilisé pour générer de nouvelles colonnes à chaque itération. Puisque les prédictions ne sont pas nécessairement parfaites, et afin d'assurer l'optimalité de la solution obtenue pour le problème maître original, le réseau complet est utilisé dans les dernières itérations pour générer les dernières colonnes. Cette nouvelle approche a été testée sur les mêmes problèmes considérés dans le premier sujet (à quelques changements près) et les résultats ont montré des gains atteignant 40% de réduction en temps de calcul.

Dans le troisième sujet, nous développons une heuristique pour la réoptimisation d'un problème après un changement mineur de ses paramètres. Plus précisément, nous considérons le cas du problème de tournées de véhicules avec contraintes de capacité, où les clients sont les mêmes mais avec des demandes légèrement différentes. L'objectif est de prédire les parties de la solution qui ont une grande probabilité de rester inchangées. Au lieu de résoudre le problème à partir de zéro, le solveur peut se baser sur ces prédictions pour réduire l'espace de recherche. Cette prédiction partielle de la solution réduit la complexité du problème et accélère sa résolution, tout en produisant des solutions de bonne qualité. L'approche proposée a été expérimentée sur différentes instances et a permis d'obtenir des solutions avec un écart d'optimalité se situant entre 0% et 1,7%, et ce en des temps de calcul raisonnables.

ABSTRACT

In the last few years, machine learning (ML) has experienced a remarkable development, especially after the emergence of deep learning. The field has seen significant advances that have attracted a lot of interest among researchers from several disciplines, including the operations research community. Recently, several studies seeking to exploit ML methods to solve combinatorial optimization problems, or to accelerate the resolution process, have been conducted. This has led to the development of many methods that have shown great potential on different optimization problems.

In this thesis, we are particularly interested in the column generation (CG) method, which is an optimization method used to efficiently solve linear programs involving a large number of variables (columns). It is an iterative method that starts with a small subset of variables from the original linear program, and generates new columns as needed until an optimal solution is obtained. The method has proven its effectiveness on several industrial problems, such as routing and scheduling problems.

We present in this thesis three different approaches for integrating machine learning in the context of CG. The goal is either to accelerate the optimization and reduce the computing time of existing methods, or to develop new heuristics that produce good quality solutions in reasonable computing times. The overall goal is to take advantage of the data that can be collected from previous executions, in order to obtain prediction models capable of guiding the optimization and reducing the search space.

In the first subject, we propose a column selection method that, at each CG iteration, applies a prediction model in order to select a subset of the generated columns. The goal is to reduce the computing time dedicated to the reoptimization of the restricted master problem by selecting the most promising columns. We tackle this problem using supervised learning, a binary classifier is used to classify the generated columns into one of the two classes : chosen or rejected. The method is considered general enough to be applied to various optimization problems, as long as a large number of columns is generated at each iteration. We demonstrate the effectiveness of this approach on two different applications, namely the vehicle routing problem with time windows, and the crew and vehicle scheduling problem. The results have shown a reduction in computing time of up to 30% on instances of different sizes.

In the second subject, we focus on problems solved using CG where the subproblem is defined on a graph and corresponds to a shortest path problem with resource constraints or one of its variants, which is the case of several routing and scheduling problems. We propose a heuristic

for solving the subproblem, which consists in reducing the size of the underlying network by selecting the arcs that have a high probability of being part of an optimal solution. A binary classification model is trained on the data of previously solved instances. The reduced network resulting from the model predictions is used to generate new columns at each iteration, as long as it generates a satisfactory number of columns. Since the predictions are not always perfect, and in order to ensure the optimality of the solution obtained for the original master problem, the full network is used when the reduced one fails to generate new columns. This novel approach was tested on the applications considered in the first subject (with some changes to the instances), and the results have shown reductions in computing time of up to 40%.

In the third subject, we present a learned heuristic for reoptimizing a problem after a minor change in its data. More precisely, we focus on the case of the capacited vehicle routing problem with static clients (i.e., same client locations) but with slightly different demands. The objective is to predict the parts of the solution that have a high probability of remaining unchanged after a perturbation of the instance data. Instead of solving the problem from scratch, the solver can use the obtained predictions in order to reduce the search space. This partial prediction of the solution reduces the complexity of the problem and speeds up its resolution, while producing good quality solutions. The proposed approach has been demonstrated on different instances and has resulted in solutions with an optimality gap ranging from 0% to 1.7% computed in reasonable computing times.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ML	Machine Learning
CG	Column Generation
MP	Master Problem
RMP	Restricted Master Problem
SP	Subproblem
MILP	Mixed-Integer Linear Program
SPPRC	Shortest Path Problem with Resource Constraints
ESPPRC	Elementary Shortest Path Problem with Resource Constraints
VCSP	Vehicle and Crew Scheduling Problem
CVRP	Capacited Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
B&P	Branch-and-price
B&B	Branch-and-bound
RL	Reinforcement Learning
RF	Random Forest
SVM	Support Vector Machine
ANN	Artificial Neural Network
GNN	Graph Neural Network

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

La méthode de génération de colonnes (GC) est une méthode itérative qui permet de résoudre efficacement les problèmes d'optimisation linéaire comportant un grand nombre de variables (colonnes). La méthode exploite le fait que la majorité de ces colonnes ne feront pas partie d'une solution optimale. Il est donc possible de commencer avec un sous-ensemble de variables, et en générer de nouvelles au fur et à mesure jusqu'à obtenir une solution optimale. Aujourd'hui, les méthodes exactes basées sur la GC sont considérées les méthodes de pointe pour résoudre plusieurs problèmes d'optimisation, tels que les problèmes de tournées de véhicules, d'horaires de véhicules et d'équipages, de rotations d'équipages, ... La vaste application de la méthode est principalement due à l'avancée remarquable des solveurs au cours des deux dernières décennies, ce qui a permis à la méthode de gagner une grande popularité. Néanmoins, le besoin d'amélioration du temps de calcul reste toujours présent vu que la taille des problèmes à résoudre continue à croître avec les besoins de l'industrie.
D'un autre côté, l'apprentissage machine a connu des avancées remarquables lors des dernières années, principalement dues à la croissance exponentielle des données et à l'amélioration de la capacité de calcul des processeurs et des cartes graphiques. L'apprentissage machine est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques dont le but est de permettre aux ordinateurs d'apprendre à partir des données. Aujourd'hui, les machines ont la capacité d'exécuter des tâches complexes qui seraient autrement difficiles à programmer explicitement, tels que la reconnaissance vocale, le traitement du langage naturel, la vision par ordinateur, les assistants conversationnels, les moteurs de recommandation, etc.
Le but de cette thèse est d'explorer de nouvelles approches basées sur les méthodes d'apprentissage machine, afin de guider l'optimisation et d'accélérer la résolution de problèmes par GC.

1.1 Notions de base sur la génération de colonnes

La GC consiste à décomposer le problème original en deux parties. La première partie correspond au problème maître restreint (PMR) qui est le programme linéaire original mais restreint à un sous-ensemble de ses variables. Ce dernier est résolu à chaque itération, souvent à l'aide de la méthode du simplexe. La seconde partie correspond au sous-problème (SP), qui est également résolu à chaque itération et son rôle consiste à générer des colonnes de coût réduit négatif (dans le cas d'un problème de minimisation) qui peuvent améliorer la solution courante du PMR. Le processus s'arrête lorsqu'il n'y a plus de colonnes susceptibles

d'améliorer la solution, i.e., aucune colonne de coût réduit négatif n'est trouvée. Sinon, les nouvelles colonnes sont ajoutées au PMR pour commencer une nouvelle itération.
Plus formellement, considérons le programme linéaire suivant appelé problème maître (PM) dans le contexte de la GC :

$$\min_{\theta} \quad \sum_{p\in\mathcal{P}} c_p\theta_p \tag{1.1}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}} \mathbf{a_p}\theta_p \geq \mathbf{b}, \tag{1.2}$$

$$\theta_p \geq 0, \quad \forall p \in \mathcal{P}, \tag{1.3}$$

où $\mathcal{P}$ est l'ensemble des indices des variables. Selon le problème à résoudre, les variables θ_p peuvent être associées à différents objets, e.g., des routes de véhicules, des plans de découpe, des horaires d'employés, ... Soit $c_p \in \mathbb{R}$ et $\mathbf{a_p} \in \mathbb{R}^m$ le coût des variables θ_p ainsi que leurs vecteurs de coefficients des contraintes, respectivement, et soit $\mathbf{b} \in \mathbb{R}^m$ le vecteur des membres de droite des contraintes (1.2) où m est le nombre de contraintes du modèle. On suppose que le nombre de variables θ_p est excessivement grand et qu'il ne serait pas possible d'énumérer toutes les variables du problème, c'est pourquoi on choisit de résoudre ce problème avec de la GC. Par conséquent, on considère un sous-ensemble restreint $\Omega \in \mathcal{P}$ de ces variables pour obtenir ainsi ce que l'on appelle le PMR. Ce dernier est résolu à chaque itération, résultant en une solution primale $\tilde{\boldsymbol{\theta}}$ et une solution duale donnée par les valeurs duales $\boldsymbol{\pi} \in \mathbb{R}^m$ associées aux contraintes (1.2). Cette solution duale est ensuite utilisée pour identifier de nouvelles variables $\theta_p, p \in \mathcal{P} \setminus \Omega$ de coût réduit négatif en résolvant le SP ci-dessous :

$$\min_{p\in\mathcal{P}} \{c_p - \boldsymbol{\pi}^T\mathbf{a_p}\}. \tag{1.4}$$

Si des colonnes de coût réduit négatif sont trouvées, elles sont ajoutées au PMR qui est ensuite réoptimisé. Sinon, la solution actuelle du PMR est optimale pour le PM, et le processus s'arrête. Généralement, le SP correspond à un problème d'optimisation (e.g., problème de sac à dos, problème de plus court chemin, etc.) et ses solutions correspondent aux objets associés aux variables θ_p. Dans de nombreuses applications, on cherche à générer et à ajouter plusieurs colonnes de coût réduit négatif à la fois, ce qui diminue le nombre d'itérations de l'algorithme et accélère la convergence de ce dernier.

1.2 Objectifs de recherche

Maintenant qu'on a présenté les idées de base de la GC, on peut se demander comment l'apprentissage machine peut être intégré à la méthode et comment il contribuera à l'accélération du processus d'optimisation. Dans cette thèse, nous explorons trois directions différentes, avec un objectif commun qui est d'exploiter les données collectées durant les exécutions antérieures afin d'entraîner des modèles capables de guider et d'accélérer le processus d'optimisation des futures exécutions. En outre, nous veillons également à ce que les méthodes proposées dans nos travaux soient applicables à un grand nombre de problèmes résolus par la GC.
Dans un premier temps, on s'intéresse aux problèmes d'optimisation où un grand nombre de variables est généré à chaque itération. Étant donné que le temps de calcul consacré aux PMRs dépend du nombre de colonnes ajoutées, l'objectif est d'entraîner un modèle capable de sélectionner, à chaque itération, les colonnes les plus prometteuses parmi les colonnes générées. Une sélection efficace permettra d'atteindre l'optimalité en ajoutant moins de colonnes dans le PMR, ce qui réduira le temps de calcul total consacré aux PMRs. Ceci signifie que cette méthode est plus adaptée aux problèmes qui prennent plus de temps dans la résolution des PMRs que les SPs. Dans un deuxième temps, toujours dans le contexte de la GC, nous nous intéressons aux problèmes d'optimisation où le SP est un problème de plus court chemin avec contraintes de ressources (SPPRC : *Shortest Path Problem with Resource Constraints*) ou une de ses variantes. Ceci est le cas de plusieurs problèmes définis sur un réseau, tels que les problèmes de tournées de véhicules et les problèmes de planification d'horaires. Contrairement au premier objectif, nous nous concentrons cette fois-ci sur la réduction du temps de calcul du SP en proposant une méthode heuristique basée sur l'apprentissage machine. Puisque le problème est défini sur un réseau et que le temps de calcul du SP dépend de la taille de ce dernier, le but est de construire un réseau réduit de plus petite taille à l'aide d'un modèle appris, qui permettra une résolution plus rapide du SP tout en générant des colonnes prometteuses, i.e., qui ont de grandes chances de faire partie de la solution optimale. Dans le troisième objectif, nous proposons une heuristique pour la réoptimisation d'un problème après un changement mineur de ses paramètres. Au lieu de réoptimiser le problème en partant de zéro, le but est de prédire les parties de la solution originale qui ont une grande chance de rester inchangées lorsque les données de l'instance sont perturbées.

1.3 Plan de la thèse

Le présent document est organisé comme suit. Le chapitre 2 est dédié à la revue de littérature couvrant la GC, les types et les méthodes d'apprentissage machine, ainsi que quelques méthodes récentes dans la littérature qui combinent la GC et l'apprentissage machine. Le

chapitre 3 décrit brièvement l'organisation des trois chapitres principaux de cette thèse. Dans le chapitre 4, nous présentons notre premier sujet qui est la sélection de colonnes basée sur l'apprentissage machine. Le chapitre 5 décrit notre deuxième sujet de recherche concernant l'utilisation de l'apprentissage machine pour la sélection d'arcs dans les problèmes impliquant un SPPRC. Le troisième et dernier sujet est présenté dans le chapitre 6, soit une heuristique pour la résolution d'un problème d'optimisation après un changement mineur de ses données. Le chapitre 7 est consacré à une discussion générale des méthodes présentées dans le cadre de cette thèse. Finalement, dans le dernier chapitre, une conclusion et quelques perspectives de recherche sont présentées.

CHAPITRE 2 REVUE DE LITTÉRATURE

Dans cette revue de littérature, une première partie sera consacrée à la méthode de GC, incluant l'algorithme de branch-and-price, les difficultés de convergence, quelques domaines d'applications ainsi que des stratégies d'accélération qui peuvent être adoptées. La deuxième partie de la revue sera dédiée à l'apprentissage machine. Les concepts de base seront présentés avec une attention particulière aux algorithmes que nous utilisons dans le cadre de cette thèse. La dernière partie de la revue portera sur les méthodes existantes qui combinent la GC et l'apprentissage machine ainsi que quelques travaux annexes.

2.1 Génération de colonnes

La GC [1] est utilisée pour résoudre de nombreux problèmes d'optimisation combinatoire, un exemple classique étant le problème de découpe [2]. Ce dernier consiste à découper des rouleaux d'une longueur donnée en des morceaux de plus petite taille afin de répondre aux demandes des clients, tout en minimisant le nombre de rouleaux utilisés. Ce problème peut être modélisé comme un programme en nombres entiers où chaque variable représente un patron de découpe possible. Puisqu'il y a un grand nombre de patrons possibles, il est possible de résoudre la relaxation linéaire du modèle avec la GC. Depuis, la méthode a été appliquée avec succès à d'autres problèmes, entre autres, les problèmes de tournées de véhicules, les problèmes de planification d'équipage, etc.

2.1.1 Branch-and-bound

Dans le cas où certaines variables du PM sont restreintes à être entières, ce qui est le cas dans plusieurs applications, la GC est souvent combinée à un algorithme d'énumération implicite, aussi appelé algorithme par *séparation et évaluation*, ou encore *branch-and-bound* (B&B) en anglais.

L'algorithme de B&B est un algorithme très utilisé pour résoudre des programmes linéaires en nombres entiers. Il repose principalement sur le paradigme "diviser pour régner", représenté par un arbre de recherche dans lequel, à chaque nœud, une relaxation linéaire du problème est résolue. Si à un nœud donné, le problème est non réalisable ou bien si la solution de sa relaxation linéaire est naturellement entière (ou mixte), le nœud n'a pas besoin d'être développé. Sinon, il existe au moins une variable fractionnaire parmi celles supposées être entières sur laquelle on peut imposer une règle de *séparation* (ou règle de branchement), i.e., la valeur de la variable fractionnaire est restreinte, de façon à créer des nœuds fils dont les régions réalisables sont disjointes et de façon à ce qu'elles ne contiennent pas la solution

de la relaxation linéaire du nœud parent. Au cours de l'exploration de l'arbre, les solutions des relaxations linéaires ainsi que les solutions entières rencontrées procurent des bornes inférieures et supérieures, respectivement. Ces bornes permettent d'élaguer les nœuds non prometteurs durant la phase d'*évaluation* évitant ainsi d'explorer l'arbre dans son intégrité. La performance de B&B dépend de trois composantes qui peuvent grandement influencer son efficacité, à savoir les règles de séparation, les stratégies d'exploration (i.e., stratégies suivies pour choisir le prochain nœud à explorer) ainsi que les règles d'élagage. Le lecteur est référé à l'article de synthèse [3] pour plus de détails sur ces trois aspects.

2.1.2 Branch-and-price

En combinant l'algorithme de B&B et la GC, on obtient une méthode connue sous le nom de *branch-and-price* (B&P) [1,4] où la relaxation linéaire à chaque nœud de l'arbre de branchement est résolue à l'aide de la GC. Puisque la performance du B&B dépend beaucoup de la qualité des bornes, des coupes valides peuvent être ajoutées à chaque nœud afin de renforcer les bornes inférieures, donnant ainsi lieu à une méthode appelée *branch-price-and-cut*.

2.1.3 Difficultés de convergence

L'algorithme de GC est bien connu pour avoir des difficultés de convergence, surtout sur les problèmes formulés comme des problèmes de partitionnement d'ensemble. Ces difficultés peuvent ralentir la convergence de la méthode et causer l'algorithme à effectuer plusieurs itérations sans amélioration significative de la solution, voire aucune amélioration du tout. De plus, l'instabilité des valeurs duales peut favoriser la génération de colonnes qui ont peu de chances de faire partie de la solution optimale.

Dans la littérature, plusieurs stratégies ont été développées pour remédier à ces difficultés [5,6], comme par exemple la suppression des contraintes redondantes du problème ou bien la perturbation de ces dernières en ajoutant des variables d'écart et de surplus [7]. D'autres techniques de stabilisation peuvent être utilisées pour réduire l'oscillation des valeurs duales d'une solution à une autre. Ces techniques peuvent par exemple forcer les variables duales à rester dans des intervalles relativement petits autour de ce qui est appelé un centre de stabilité, qui correspond à la meilleure estimation des valeurs duales optimales, avec l'utilisation d'une fonction de stabilisation pour pénaliser les solutions duales qui sont loin du centre de stabilité [8]. Les pénalités et les intervalles peuvent être ajustés dynamiquement tout au long de l'exécution de la GC. Dans [9], les auteurs proposent, entre autres, une technique de lissage, qui considère un nouveau vecteur de valeurs duales $\tilde{\boldsymbol{\pi}} = \alpha\hat{\boldsymbol{\pi}} + (1-\alpha)\boldsymbol{\pi}$ où le paramètre $\alpha \in (0, 1]$ indique le niveau de lissage, et les vecteurs $\boldsymbol{\pi}$ et $\hat{\boldsymbol{\pi}}$ représentent, respectivement, le vecteur des valeurs duales de la solution actuelle ainsi qu'un vecteur contenant

les informations duales des itérations précédentes (e.g., une somme pondérée des vecteurs duaux calculée lors des itérations précédentes). Le nouveau vecteur $\tilde{\boldsymbol{\pi}}$ est ensuite utilisé à chaque itération de la GC par le SP pour générer de nouvelles colonnes. Si des colonnes de coût réduit négatif sont trouvées par rapport à $\tilde{\boldsymbol{\pi}}$ et $\boldsymbol{\pi}$, elles sont ajoutées au PMR et la valeur de α est maintenue. Dans le cas où des colonnes de coût réduit négatif ont été identifiés par rapport à $\boldsymbol{\pi}$ mais pas $\tilde{\boldsymbol{\pi}}$, ces dernières sont ajoutées au PMR et la valeur de α est diminuée. Finalement, si aucune colonne de coût réduit négatif n'est trouvée, la valeur de α est diminuée et le SP est résolu de nouveau.

Une autre possibilité pour remédier au problème de dégénérescence consiste à utiliser une méthode de points intérieurs [10]. Cependant, cette dernière a tendance à produire des solutions très fractionnaires, ce qui peut augmenter considérablement la taille de l'arbre de branchement.

D'autres méthodes plus spécifiques aux problèmes impliquant des contraintes de partitionnement d'ensemble ont été explorées. Entre autres, une méthode d'agrégation dynamique de contraintes (ADC) a été introduite par Elhallaoui et *al.* [11]. Comme son nom l'indique, cette méthode consiste à réduire le nombre de contraintes de partitionnement d'ensemble du PMR en regroupant certaines d'entre elles, et en ne conservant qu'une seule contrainte représentative par groupe. L'accélération obtenue par cette méthode résulte du fait que le PMR avec contraintes agrégées a moins de contraintes que le modèle original restreint, ainsi il est plus rapide à réoptimiser (i.e., moins de pivots lors de la résolution) et tend à être moins dégénéré. Cependant, afin de garantir l'optimalité de la solution obtenue (pour le PMR original sans contraintes agrégées), les agrégations sont mises à jour au cours de la résolution. Une version améliorée de l'ADC a été proposée plus tard [12] dans le but de maintenir un niveau d'agrégation plus élevé et de réduire davantage la dégénérescence. Ces travaux ont motivé le développement de l'algorithme du simplexe primal amélioré (IPS : *Improved Primal Simplex*) [13] qui peut résoudre les programmes linéaires sans GC tout en réduisant la dégénérescence. La méthode décompose le problème en deux sous-problèmes : un problème réduit (PR) et un problème complémentaire (PC). Le PR est un programme linéaire qui contient un sous-ensemble de contraintes et de variables appelées variables compatibles, c'est-à-dire celles dont les colonnes peuvent être écrites comme des combinaisons linéaires des colonnes qui font partie de la solution courante. Le PC est également un programme linéaire qui est résolu pour prouver l'optimalité de la solution du PR pour le problème original ou pour identifier les variables incompatibles à ajouter pour construire une nouvelle solution du PR. En exploitant les résultats théoriques de l'IPS, Zaghrouti et *al.* [14, 15] ont développé des variantes de l'algorithme du simplexe intégral reposant sur la décomposition (ISUD : *Integral Simplex Using Decomposition*) pour résoudre des problèmes de partionnement d'ensemble.

Cet algorithme résout alternativement un PR et un PC pour trouver une séquence de solutions entières améliorantes. Puis en 2019, Tahir et *al.* [16] ont introduit une combinaison d'ISUD et de la GC, sous le nom de génération de colonnes intégrale (ICG : *Integral Column Generation*). À chaque itération de la GC, ISUD résout le PMR pour obtenir une solution entière, à l'opposé de la GC standard qui ne produit que des solutions fractionnaires la plupart du temps. Un système d'équations linéaires impliquant les valeurs duales du dernier PC est résolu afin d'obtenir une solution duale. Cette dernière est ensuite utilisée par le SP de la GC pour générer de nouvelles colonnes à traiter par ISUD dans la prochaine itération du PMR. Une adaptation de la méthode ICG, nommée I^2CG, a été ensuite proposée par les mêmes auteurs [17] pour la prise en compte des contraintes supplémentaires du PMR, en proposant de nouvelles formulations pour le PR et le PC.

2.1.4 Problème de plus court chemin avec contraintes de ressources

La méthode de GC peut être appliquée à une variété de problèmes d'optimisation. Parmi les applications que nous considérons dans cette thèse sont les problèmes d'horaires d'équipages et de tournées de véhicules. Ces problèmes sont dans la plupart du temps représentés sur des graphes, où le SP est un SPPRC ou un de ses variantes. C'est pourquoi nous avons choisi de dédier cette section pour la description du SPPRC ainsi que les méthodes de base utilisées pour le résoudre.

Le SPPRC [18] est un problème défini sur un réseau où le but est de trouver un chemin de moindre coût entre un nœud source et un nœud de destination, tout en respectant des contraintes sur un ensemble de ressources. Dans le contexte de la GC, les chemins peuvent représenter différentes entités en fonction du problème à résoudre, par exemple, un itinéraire de véhicule, un horaire de chauffeur, etc. Quant aux ressources, des exemples possibles sont le temps, la capacité d'un véhicule (c'est-à-dire la charge maximale autorisée), la durée d'une pause de travail, etc. Les ressources sont en général des quantités dont les valeurs varient le long d'un chemin en fonction de ce que l'on appelle des fonctions d'extension de ressources [19] (REF : *Resource Extension Function*). De plus, les valeurs prises par les ressources à chaque nœud doivent être comprises dans des intervalles prédéfinis. Si on prend l'exemple du VRPTW, deux ressources sont considérées, à savoir le temps et la charge. La ressource temps est utilisée pour restreindre l'heure de début du service chez un client, tandis que la ressource charge garantit que la quantité totale d'articles livrés ne dépasse pas la capacité du véhicule. Le SPPRC est considéré comme un problème NP-difficile, mais il peut être résolu en temps pseudo-polynomial à l'aide d'algorithmes de programmation dynamique, plus précisément, un algorithme d'étiquetage est souvent utilisé.
De façon plus formelle, le SPPRC peut être représenté sur un graphe orienté $G = (V, A)$ où

$V = V' \cup \{s, t\}$ est l'ensemble des nœuds incluant le nœud source s et le nœud de destination t. Soit A l'ensemble des arcs et R l'ensemble des ressources. Pour chaque nœud $i \in V$, on définit les fenêtres de ressources $[l_i^r, u_i^r], l_i^r, u_i^r \in \mathbb{R}$ restreignant les valeurs que la ressource $r \in R$ peut prendre au nœud i, et pour chaque arc (i, j), on définit le coût de parcours c_{ij}.
Un chemin partiel $P = (v_0, v_1, \ldots, v_p)$ dans G représente une séquence de nœuds visités telle que $(v_{i-1}, v_i) \in A, v_i \in V, \forall i = 1, 2, \ldots, p$ où v_p est le dernier nœud visité par le chemin et $v_0 = s$. Soit T_i^r la quantité de la ressource $r \in R$ consommée le long d'un chemin partiel partant de la source s jusqu'au nœud i (pour simplifier la notation, les nœuds sont représentés par les indices i et non v_i). Un chemin P est considéré réalisable si pour toutes les ressources $r \in R$, les fenêtres de ressources de chaque nœud i du chemin P sont respectées, i.e., $T_i^r \in [l_i^r, u_i^r]$.
La consommation de ressources le long d'un arc $(i, j) \in A$ est définie pour chaque arc (i, j) et pour chaque ressource $r \in R$ et est généralement donnée par la REF $f_{ij}^r : \mathbb{R}^{|R|} \longrightarrow \mathbb{R}$. Cette dernière dépend d'un vecteur de ressources $\boldsymbol{T_i} = (T_i^0, T_i^1, \ldots, T_i^{|R|})^\intercal$. Pour le reste de la section, on suppose que la REF pour tout arc et toute ressource est classique, non décroissante et qu'il n'y a pas d'interdépendance entre les ressources. Cette dernière est donnée par la formulation $f_{ij}^r(\boldsymbol{T_i}) = T_i^r + t_{ij}^r$ où t_{ij}^r est une constante représentant la quantité de la ressource consommée en parcourant l'arc (i, j). Dans certains cas, comme pour le VRPTW, le chauffeur est autorisé à attendre s'il arrive avant l'heure de début du service du client. Pour tenir compte de cela, la fonction $f_{ij}^r(\boldsymbol{T_i}) = max\{l_j^r, T_i^r + t_{ij}^r\}$ est souvent utilisée à la place.
Dans le contexte de la GC, les valeurs duales π_i associées aux contraintes du PMR doivent être prises en compte lors de la résolution du SP. Dans le cas du SPPRC, ceci est effectué en incluant l'information duale sur chaque arc (i, j) en utilisant des coûts modifiés $\bar{c}_{ij} = c_{ij} - \sum_{k=1}^{m} \beta_{ij}^k \pi_k$, où β_{ij}^k est un paramètre binaire prenant la valeur 1 si l'arc (i, j) contribue à la contrainte k, 0 sinon. Si on prend l'exemple d'un problème de tournées de véhicules, chaque contrainte du problème maître (une formulation de partitionnement d'ensemble est souvent utilisée) est associée à la tâche de visite d'un client. Ainsi, pour un arc (i, j) où j est un nœud client, l'arc est considéré comme contribuant à la contrainte associée au client j. De ce fait, la valeur duale de la contrainte est incluse dans le coût modifié de l'arc (i, j). En utilisant ces coûts modifiés, résoudre le SPPRC est équivalent à identifier un chemin de plus petit coût réduit, si ce coût est négatif, sa variable correspondante peut être ajoutée dans le PMR.

Algorithme d'étiquetage

Le SPPRC est généralement résolu à l'aide d'un algorithme d'étiquetage (voir [18]) où les étiquettes représentent des chemins partiels dans le graphe G, commençant toujours par le

nœud source s. Chaque étiquette L encode des informations sur un chemin partiel P, telles que le coût réduit $\bar{c}(L)$, le dernier nœud visité par le chemin $v(L)$ et la quantité accumulée $T^r(L)$ de chaque ressource $r \in R$. L'algorithme commence par l'étiquette de base représentant le chemin partiel ne contenant que la source s, i.e., $P = (s)$. Cette étiquette est ensuite prolongée récursivement le long des arcs sortants résultant en des chemins partiels réalisables jusqu'à atteindre le nœud de destination t. En général, pour une étiquette L représentant un chemin partiel s'arrêtant au nœud $i \neq t$, une nouvelle étiquette L' est créée lorsqu'une extension est effectuée le long d'un arc $(i, j) \in A$, tel que :

$$v(L') = j, \tag{2.1}$$
$$\bar{c}(L') = \bar{c}(L) + \bar{c}_{ij}, \tag{2.2}$$
$$T^r(L') = \max\{l^r_j, T^r(L) + t^r_{ij}\}, \quad \forall r \in R. \tag{2.3}$$

L'étiquette L' garde également une référence à l'étiquette précédente L afin de construire le chemin complet lorsque l'algorithme se termine, et elle est considérée comme non réalisable si $\exists r \in R$ tel que $T^r(L') > u^r_j$. La performance de l'algorithme d'étiquetage est étroitement liée à ce qui est appelé une règle de dominance. Son rôle est de réduire l'espace de recherche et d'éliminer un maximum d'étiquettes non prometteuses. La règle de dominance à choisir dépend des propriétés des REFs utilisées. Pour les REFs non décroissantes, on dit qu'une étiquette L_1 domine une étiquette L_2 si les conditions suivantes sont vérifiées :

$$v(L_1) = v(L_2), \tag{2.4}$$
$$\bar{c}(L_1) \leq \bar{c}(L_2), \tag{2.5}$$
$$T^r(L_1) \leq T^r(L_2), \quad \forall r \in R. \tag{2.6}$$

À la fin de l'algorithme, plusieurs étiquettes sont obtenues au nœud de destination t. Les variables/colonnes représentant les étiquettes de coût réduit négatif sont ajoutées au PMR pour commencer la prochaine itération de la GC. Ainsi, résoudre le SPPRC par algorithme d'étiquetage permet de générer beaucoup de colonnes à la fois, ce qui est connu pour accélérer la convergence et réduire le nombre des itérations de la GC.
Une amélioration de l'algorithme SPPRC de base décrit ci-haut a été proposée par Righini et Salani [20]. Ces derniers ont proposé un algorithme d'étiquetage bidirectionnel qui propage les étiquettes dans les deux directions en même temps, c'est-à-dire du nœud source s vers la destination t, et de la destination vers la source. Les étiquettes en amont et en aval rencontrées

sur les nœuds intermédiaires sont ensuite fusionnées pour former des chemins complets. Dans la littérature, plusieurs techniques d'accélération ont été proposées pour accélérer la résolution du SPPRC, nous y consacrerons une prochaine section pour en discuter en détail.

Le cas du SPPRC élémentaire

Le ESPPRC est une variante du SPPRC qui est largement plus difficile à résoudre (i.e., NP-difficile au sens fort), principalement du à la contrainte d'élémentarité imposant qu'un nœud soit visité au plus une fois dans un chemin. Puisque les coûts modifiés $\bar{c}_{ij}$ peuvent prendre des valeurs négatives, des cycles de coût négatif peuvent donc exister. Il est possible de relaxer cette contrainte et résoudre le SPPRC standard à la place (i.e., autoriser qu'un nœud soit visité plusieurs fois), engendrant des bornes inférieures fournies par le PM de moindre qualité. Certains auteurs ont développé des méthodes exactes pour la résolution du ESPPRC. Feillet et *al.* [21] ont proposé d'étendre l'algorithme de base du SPPRC en gardant trace des nœuds déjà visités ainsi que les nœuds non atteignables à cause des contraintes sur les ressources. la règle de dominance est également adaptée en conséquence pour tenir compte de ce changement. Les résultats ont montré que les chemins élémentaires permettent d'obtenir de bien meilleures bornes, mais vu la complexité du problème, plusieurs auteurs ont eu recours à des relaxations, où, en général, un compromis entre la qualité des bornes et la difficulté à résoudre le problème est recherché. Parmi les relaxations, on trouve *k-cyc-SPPRC* [22] dont le but est d'empêcher la formation de cycles de longueur k ou moins. Une autre relaxation est celle de l'*élémentarité partielle* [23] qui force l'élémentarité à seulement une liste restreinte (et de taille limitée) de noeuds. Cette liste est mise à jour dynamiquement durant l'optimisation et l'algorithme d'étiquetage est ajusté afin d'interdire les visites multiples aux noeuds de la liste. En 2011, Baldacci et *al.* ont introduit une relaxation efficace appelée *ng-route* [24]. Étant donné un paramètre entier $\Delta \geq 0$, pour chaque noeud $i \in V$, un voisinage $\mathcal{N}_i$ est défini contenant les Δ plus proches voisins de i. Soit $V(L) = \{s, v_1, \ldots, v_p\}$ l'ensemble des noeuds visités par le chemin représenté par l'étiquette L. L'étiquette ne peut être prolongée le long d'un arc (i, j) si $j \in \Pi(L)$ où :

$$\Pi(L) = \{v_i \in V(L) \setminus \{s, v_p\}. : v_i \in \bigcap_{k=i+1}^{p} \mathcal{N}_{v_k}\} \cup \{v_p\}. \tag{2.7}$$

Dans le cas contraire, i.e., $j \notin \Pi(L)$, une nouvelle étiquette L' est créée et le nouveau ensemble $\Pi(L')$ est défini comme suit :

$$\Pi(L') = \{\Pi(L) \cap \mathcal{N}_j\} \cup \{j\}. \tag{2.8}$$

De plus, une nouvelle condition est ajoutée à la règle de dominance décrite dans (2.4)-(2.6) lorsque deux étiquettes sont comparées :

$$\Pi(L_1) \subseteq \Pi(L_2). \tag{2.9}$$

A noter qu'avec une valeur $\Delta = 0$, l'algorithme est équivalent à résoudre le SPPRC standard, alors qu'avec une valeur $\Delta = |V| - 1$, on se retrouve avec le cas du ESPPRC.

Fonctions d'extension de ressources

Il est à noter que les REFs fournissent un outil puissant permettant de modéliser plusieurs contraintes pertinentes en pratique. Nous présentons dans cette section quelques exemples qui surgissent dans des problèmes de tournées de véhicules et d'horaires d'équipage où une interdépendance entre les ressources peut exister, ainsi que l'utilisation de fonctions non-linéaires.

Dans le cas des problèmes de tournées de véhicules, il est possible que le coût de déplacement le long d'un arc dépende de la quantité transportée par le véhicule. Dans un tel cas, le coût d'un arc (i, j) est donné par une fonction non-décroissante $c_{ij} : \mathbb{R}_+ \longrightarrow \mathbb{R}$. Considérons le cas d'un problème avec trois ressources $R = \{co\hat{u}t, charge, temps\}$. La mise à jour de la ressource coût le long d'un arc (i, j) peut être donnée par la REF suivante :

$$f_{ij}^{co\hat{u}t}(T_i^{co\hat{u}t}, T_i^{charge}) = T_i^{co\hat{u}t} + c_{ij}(T_i^{charge}) \tag{2.10}$$

Nous pouvons remarquer que la fonction $f_{ij}^{co\hat{u}t}$ dépend de la fonction de coût c_{ij} et qu'il y a une interdépendance entre les deux ressources *coût* et *charge* du véhicule. Dans d'autres cas, il est possible que les coûts et/ou les temps de parcours le long d'un arc (i, j) varient selon l'heure de visite des clients.

Dans le cas des problèmes d'horaires d'équipage, en plus des coûts sur les arcs (i.e., coûts de service et de parcours), il est également possible de prendre en considération les coûts engendrés par les temps d'attente. Le calcul d'horaires de moindres coûts peut être obtenu en définissant des ressources interdépendantes et des REFs non-linéaires [25].

Grâce à la flexibilité offerte par les REFs, il est possible de modéliser plusieurs cas intéressants qui peuvent survenir dans des problèmes réels. Nous invitons les lecteurs intéressés à consulter [18, 19] pour plus d'informations sur les REFs et leurs propriétés.

2.1.5 Exemples d'applications

Les méthodes basées sur la GC sont considérées les méthodes exactes de pointe pour de nombreux problèmes d'optimisation. Cependant, dans cette section, nous nous limiterons aux applications qu'on a considérées dans nos expérimentations numériques, à savoir le problème de tournées de véhicules avec contraintes de capacité (CVRP : *Capacited Vehicle Routing Problem*), sa variante avec fenêtre de temps (VRPTW : *Vehicle routing Problem with Time Windows*) ainsi que le problème d'horaires d'équipages et de véhicules (VCSP : *Vehicle and Crew Scheduling Problem*). Ces problèmes ont certains points en commun : **i)** ils peuvent être résolus efficacement avec un algorithme de branch-and-price, **ii)** ils peuvent être formulés comme un problème défini sur un graphe/réseau et **iii)** le sous-problème (dans le contexte de la GC) correspond à SPPRC ou sa variante élémentaire (ESPPRC : *Elementary SPPRC*).

Problème de tournées de véhicules

Le CVRP [21, 26, 27] est un problème classique qui est largement étudié dans la littérature. Étant donné une flotte de véhicules, le CVRP consiste à construire des routes pour desservir des clients géographiquement dispersés, tout en minimisant les coûts de déplacement et en respectant la capacité des véhicules. Chaque route doit commencer au dépôt, desservir un ensemble de clients et revenir au dépôt à la fin de la tournée. Dans nos expérimentations, nous considérons également la variante avec fenêtres de temps (VRPTW), qui est une généralisation du CVRP où, pour chaque client, une fenêtre de temps durant laquelle le service doit être effectué est définie. Les méthodes exactes les plus efficaces et les plus récentes pour résoudre le CVRP [28] et le VRPTW [29, 30] sont basées sur la méthode de *branch-price-and-cut* [27].

Formulation mathématique. Le problème maître est dans la plupart des cas un problème de partitionnement d'ensemble où chaque variable désigne une route réalisable et chaque contrainte correspond à la tâche de visiter un client. Soit C l'ensemble des clients du problème et Ω l'ensemble des routes réalisables (i.e., élémentaires et satisfaisant les contraintes de ressources). Pour chaque route $r \in \Omega$, on définit une variable binaire x_r, qui est égale à 1 si la route fait partie de la solution et 0 sinon. On définit également le coût c_r de la route, ainsi qu'un paramètre a_i^r qui prend la valeur 1 si la route $r \in \Omega$ visite le client $i \in C$, 0 sinon. Le PM en nombres entiers peut être formulé comme le programme en nombres entiers suivant :

$$\min_{x} \quad \sum_{r\in\Omega} c_r x_r \tag{2.11}$$

$$\text{s.t.} \quad \sum_{r\in\Omega} a_i^r x_r = 1, \quad \forall i \in C \tag{2.12}$$

$$x_r \in \{0,1\}, \quad \forall r \in \Omega. \tag{2.13}$$

La fonction objectif (2.11) vise à minimiser le coût total des tournées. Les contraintes (2.12) permettent de s'assurer que chaque client $i \in C$ est visité exactement une fois. Enfin, les contraintes (2.13) correspondent aux contraintes d'intégrité, forçant les variables x_r à prendre des valeurs binaires.
Cette formulation est résolue avec une méthode B&P où le PMR correspond à la relaxation linéaire de la formulation (2.11)-(2.13) mais restreint à un sous-ensemble $\mathcal{R} \subset \Omega$ de routes. Des nouvelles routes sont générées et ajoutées au problème en résolvant un ESPPRC (par un algorithme d'étiquetage [18]) défini sur un graphe $G = (V, A)$ où $V = C \cup \{s, t\}$ est l'ensemble des noeuds incluant les noeuds dépôt s et t. Soit π_i les valeurs duales correspondant aux contraintes (2.12) avec $\pi_s = 0$. Les coûts modifiés $\bar{c}_{ij}$ utilisés lors de la résolution du SP peuvent être calculés comme suit :

$$\bar{c}_{ij} = c_{ij} - \pi_i. \tag{2.14}$$

À noter que toutes les contraintes relatives aux ressources sont traitées dans le SP et que le nombre de ressources dépend du problème à résoudre (i.e., CVRP ou VRPTW). Nous invitons les lecteurs intéressés à consulter l'article de synthèse de Costa et *al.* [27] pour une revue complète sur les méthodes exactes pour le CVRP et le VRPTW y compris les coupes qui peuvent être ajoutées pour renforcer les relaxations et les décisions de branchement.

Problème d'horaires d'équipages et de véhicules

Dans un système de transport public, le processus de planification passe par plusieurs étapes. La première étape dans le processus consiste à déterminer les lignes, les arrêts sur chaque ligne et les fréquences des voyages. Sur la base de ces informations, les horaires sont déterminés, ce qui donne lieu à des trajets avec les heures et les lieux de début et de fin correspondants. Les deux prochaines étapes dans le processus sont la construction des itinéraires des véhicules et les horaires journaliers des chauffeurs, respectivement, ce qui revient à résoudre les deux problèmes de planification : le problème d'itinéraires de véhicules (VSP : *Vehicle Scheduling*

Problem) et le problème d'horaires d'équipages (CSP : *Crew Scheduling Problem*). Traditionnellement, ces deux problèmes sont résolus d'une façon séquentielle, où on cherche d'abord à déterminer les itinéraires des véhicules avant les horaires des chauffeurs. Cependant, cette approche ne garantit pas de fournir la meilleure solution, parce que la plupart du temps, les coûts des chauffeurs dominent les coûts de l'utilisation des véhicules. Une autre idée est de modifier le processus et de planifier les horaires des chauffeurs en premier. Malheureusement, ce n'est pas une bonne idée non plus, puisque plus de véhicules peuvent être nécessaires pour satisfaire les horaires des équipages, et en raison des coûts fixes élevés d'un véhicule, cela peut être très couteux.
Dans nos expérimentations, nous nous intéressons au VCSP qui combine les deux problèmes simultanément. Nous considérons l'approche décrite dans l'article de Haase et *al.* [31], qui est un modèle de partitionnement d'ensemble qui n'implique que les variables d'horaires des équipage. Des contraintes supplémentaires sont ajoutées au problème pour assurer l'obtention des itinéraires des véhicules en un temps polynomial par la suite. En outre, les coûts des véhicules sont inclus dans la fonction objectif afin de garantir l'obtention d'une solution optimale. Une méthode de branch-and-price est utilisée pour résoudre la formulation proposée. Le SP correspond cette fois-ci à un SPPRC puisque le graphe sous-jacent est acyclique.

Description du problème Nous nous intéressons ici au cas d'une compagnie d'autobus, disposant d'un seul dépôt et d'une flotte de véhicules homogène tel que décrit dans Haase et *al.* [31]. Chaque ligne de bus est définie par : un point de départ, un point de destination et plusieurs arrêts intermédiaires où les passagers peuvent monter et descendre du bus. Certains de ces points d'arrêt sont considérés comme des points de relève où on peut changer de chauffeur.
Pour chaque ligne, il existe un horaire prédéfini définissant un ensemble de trajets (trips en anglais) à effectuer durant la journée. Pour chacun de ces trajets, on dispose de l'heure de départ, de l'heure d'arrivée et des heures d'arrivée aux arrêts intermédiaires. Un trajet est desservi par le même bus. Un trajet est divisé en une séquence de segments consécutifs, appelés *d-trajets* qui sont définis en fonction des points de relève du trajet. Souvent, les autobus doivent se déplacer sans passagers (mouvements à vide) pour se rendre au début d'un trajet ou pour retourner au dépôt. Les d-trajets et les mouvements à vide effectués par un chauffeur dans le même bus forment une *pièce-de-travail* (p-d-t), et des pauses doivent être accordées au chauffeur entre les p-d-t consécutives.
En ce qui concerne les chauffeurs, il peut y avoir différents types de journées de travail (duties en anglais), chaque type pouvant avoir des contraintes spécifiques. Par exemple, un premier type peut imposer qu'un conducteur reste dans le même autobus tout au long de son service,

tandis qu'un deuxième type peut permettre jusqu'à deux changements d'autobus pendant le même service. D'autres contraintes concernant le nombre des p-d-t, le nombre de pauses et leurs durées, le temps de travail total, ... peuvent être prises en considération.

Formulation mathématique Le problème VCS est représenté à l'aide de réseaux $G^u = (N^u, A^u), u \in U$ avec U l'ensemble des types de journées de travail et N^u et A^u représentent respectivement les ensembles des nœuds et d'arcs de G^u. Étant donné que la structure de réseau du problème est assez complexe et contient plusieurs composants, on réfère les lecteurs intéressés à consulter [31] pour plus de détails.
Soit W l'ensemble des trajets, V l'ensemble des d-trajets, H l'ensemble des heures auxquelles un bus doit quitter le dépôt pour arriver à l'heure de départ d'un trajet. Soit $\Omega^u, u \in U$ l'ensemble des horaires possibles pour un type de journée de travail. Pour chaque horaire $\rho \in \Omega^u$, un coût c_ρ y est attribué, ainsi que les paramètres binaires suivants : e_ρ^v qui prend la valeur 1 si le d-trajet $v \in V$ est desservi par l'horaire ρ, 0 sinon ; f_ρ^w prend la valeur 1 si on a un mouvement de bus entrant au point de départ du trajet $w \in W$, 0 sinon ; g_ρ^w prend la valeur 1 si on a un mouvement de bus sortant du point de destination du trajet $w \in W$, 0 sinon ; q_ρ^h prend la valeur 1 si le trajet contient un mouvement de bus commençant à l'heure $h \in H$ ou avant et finissant après l'heure h, et prend la valeur 0 sinon. De plus, il y a un coût fixe pour chaque bus utilisé durant la journée, qu'on notera c.
On définit la variable binaire $\theta_\rho^u, \rho \in \Omega^u, u \in U$ qui prend la valeur 1 si un chauffeur est assigné à l'horaire ρ, 0 sinon et la variable non négative B qui correspond au nombre de bus utilisés pour desservir tous les trajets. Avec cette notation, on formule le problème VCSP comme le programme en nombres entiers suivant :

$$\min_{\theta} \quad cB + \sum_{u \in U} \sum_{\rho \in \Omega^u} c_\rho \theta_\rho^u \tag{2.15}$$

$$\text{s.t.} \quad \sum_{u \in U} \sum_{\rho \in \Omega^u} e_\rho^v \theta_\rho^u = 1, \quad \forall v \in V, \tag{2.16}$$

$$\sum_{u \in U} \sum_{\rho \in \Omega^u} f_\rho^w \theta_\rho^u = 1, \quad \forall w \in W, \tag{2.17}$$

$$\sum_{u \in U} \sum_{\rho \in \Omega^u} g_\rho^w \theta_\rho^u = 1, \quad \forall w \in W, \tag{2.18}$$

$$\sum_{u \in U} \sum_{\rho \in \Omega^u} q_\rho^h \theta_\rho^u \leq B, \quad \forall h \in H, \tag{2.19}$$

$$\theta_\rho^u \in \{0, 1\}, \quad \forall u \in U, \forall \rho \in \Omega^u. \tag{2.20}$$

La fonction objectif (2.15) minimise les coûts fixes de l'utilisation des autobus et les coûts

d'opération engendrés par les autobus et les chauffeurs. Les contraintes (2.16) indiquent que chaque d-trajet doit être couvert par un seul chauffeur. Les contraintes (2.17) garantissent qu'il y a un bus disponible au point de départ de chaque trajet, quant aux contraintes (2.18), elles assurent qu'il y a un bus qui quitte la destination finale du trajet. En d'autres termes, les contraintes (2.16)-(2.18) assurent la conservation du flot des bus en dehors du dépôt. Les contraintes (2.19) servent à compter le nombre de bus total nécessaire, en combinaison avec la fonction objectif qui cherche à minimiser le nombre de bus utilisés. Finalement, les contraintes (2.20) forcent les variables θ_ρ^u à prendre des valeurs binaires.
La relaxation linéaire de la formulation (2.15)-(2.20) peut être résolue par la méthode de GC, et le PMR est tout simplement cette relaxation mais limitée à un sous-ensemble de colonnes. Quant aux SPs, il y en a un pour chaque réseau $G^u, u \in U$ et ils correspondent à des SPPRCs puisque les réseaux sont acycliques. Soit $\boldsymbol{\alpha} = \{\alpha^v \mid v \in V\}, \boldsymbol{\beta} = \{\beta^w \mid w \in W\}, \boldsymbol{\gamma} = \{\gamma^w \mid w \in W\}, \boldsymbol{\delta} = \{\delta^h \mid h \in H\}$ les vecteurs des valeurs duales associés aux contraintes (2.16)-(2.19), respectivement. Le coût réduit de l'arc $(i, j) \in A^u, u \in U$ peut être calculé comme suit :

$$\bar{c}_{ij} = c_{ij} - e_{ij}^v \alpha^v - f_{ij}^w \beta^w - g_{ij}^w \gamma^w - q_{ij}^h \delta^h. \tag{2.21}$$

Comme pour le CVRP, un algorithme d'étiquetage est utilisé pour résoudre les SPs et plusieurs colonnes sont générées à chaque itération. Pour des informations complémentaires sur les composants du réseau et les règles de branchement, voir [31].

2.1.6 Techniques d'accélération

En plus des techniques employées pour minimiser l'effet de la dégénérescence (voir la section 2.1.3), plusieurs stratégies peuvent être utilisées pour accélérer la GC [32]. Étant donné que le temps de calcul de la GC est réparti entre la réoptimisation des PMR et la résolution du SP, certaines de ces techniques sont destinées pour le PMR, tandis que d'autres sont conçues pour le SP. Nous présentons ici quelques idées de base qui ont été utilisées dans nos expérimentations.

(1) À chaque itération de la GC, plusieurs colonnes de coût réduit négatif peuvent être ajoutées au PMR. Généralement, cette stratégie réduit le nombre total d'itérations. La possibilité de générer plusieurs colonnes à la fois dépend de l'algorithme utilisé pour résoudre le SP. S'il s'agit d'un SPPRC résolu avec un algorithme d'étiquetage, plusieurs colonnes sont générées à la fois sans besoin de calcul supplémentaire.

(2) Toujours concernant les colonnes, une autre stratégie consiste à en supprimer du PMR

lorsqu'il y en a un très grand nombre. Les colonnes supprimées peuvent toujours être générées à nouveau si nécessaire. Cette stratégie peut être implémentée en utilisant deux paramètres : un nombre minimum n^-_{cols} et un nombre maximum n^+_{cols} de colonnes à conserver dans le PMR. Une fois le nombre maximum atteint, les $n^+_{cols} - n^-_{cols}$ colonnes ayant les coûts réduits les plus positifs sont retirées du PMR.

(3) La génération d'un grand nombre de colonnes par itération peut ne pas être efficace si les colonnes ajoutées au PMR ne sont pas assez différentes les unes des autres. Pour les problèmes formulés comme un problème de partitionnement d'ensemble, chaque contrainte est liée à une tâche, tel que desservir un client, effectuer un trajet, etc. Pour assurer une certaine diversification, la procédure suivante peut être adoptée. Les colonnes sont tout d'abord triées par leur coût réduit, i.e., pour favoriser les colonnes avec les coûts réduits les plus négatifs, puis elles sont distribuées par ordre sur un nombre n_{blocs} de blocs. Chaque bloc doit contenir des colonnes dites disjointes, i.e., des colonnes qui ne couvrent pas les mêmes tâches. Initialement, la première colonne est ajoutée au bloc 1. Si la deuxième colonne est disjointe de la première, elle est ajoutée dans le bloc 1 aussi, sinon dans le bloc 2, et ainsi de suite. Le nombre de blocs n_{blocs} est un paramètre à préciser, et peut servir à limiter le nombre de colonnes ajouté à chaque itération.

(4) Une autre technique souvent utilisée, et qui est étroitement liée au point précédent (i.e., la diversification des colonnes), consiste à résoudre le sous-problème plusieurs fois dans la même itération. Pour les problèmes définis sur un réseau, à chaque fois qu'une colonne de coût réduit négatif est générée, les éléments du réseau associés aux tâches couvertes par la colonne sont retirés et une nouvelle itération est initiée. En conséquence, les sous-problèmes deviennent de plus en plus petits et donc plus rapides à résoudre. Chaque sous-problème résulte en des colonnes disjointes et le processus est répété tant que le coût réduit des colonnes générées est assez négatif, ou qu'un nombre maximum de colonnes à générer par itération est atteint.

(5) Lors de la résolution des SPs, il est possible d'utiliser des heuristiques afin de générer des colonnes de coût réduit négatif rapidement, et garder l'algorithme exact que pour les dernières itérations afin de prouver l'optimalité de la solution. Voici quelques stratégies [23, 33] qui peuvent être employées lorsque le SP est un SPPRC ou une de ses variantes :

- Il est possible de ne considérer qu'un sous-ensemble de ressources lors de l'application de la règle de dominance. En procédant ainsi, plusieurs étiquettes non dominées peuvent être rejetées, y compris celles menant à des chemins optimaux.
- Une autre heuristique consiste à garder un nombre limité n d'étiquettes à chaque noeud. Les étiquettes sont triées par coût réduit et n étiquettes sont préservées.

— Un autre moyen d'accélérer le SP consiste à réduire la taille du graphe en retirant des arcs [26]. Une possibilité consiste à limiter le nombre maximum d'arcs entrant et sortant à garder pour chaque noeud (autre que la source et la destination), en se basant sur les coûts réduits $\bar{c}_{ij}$. Comme pour la stratégie précédente, les arcs avec les coûts réduits les plus négatifs sont favorisés.

— D'autres heuristiques en dehors des algorithmes d'étiquetage peuvent être utilisées pour générer des chemins. Entre autres, une méthode tabou peut être employée [23]. Comme pour les stratégies précédentes, une méthode exacte est invoquée au moins une fois lors de la dernière itération pour s'assurer de l'optimalité de la solution trouvée.

2.2 Apprentissage machine

L'apprentissage machine (ML : *Machine learning*) est un sous domaine de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques afin de donner aux ordinateurs la capacité d'apprendre à partir d'observations, sans être explicitement programmés. En fournissant une quantité massive de données à la machine, cette dernière doit découvrir par elle-même comment résoudre le problème, en ajustant graduellement ses paramètres internes. Le but n'est pas seulement de mémoriser les données mais de pouvoir généraliser à de nouvelles observations. Dans cette section, nous présentons les principaux types d'apprentissage, les défis qui peuvent être rencontrés ainsi que les algorithmes d'apprentissage qui peuvent être employés selon le problème à résoudre. On invite les lecteurs à consulter les références [34–36] pour plus de détails sur le sujet.

2.2.1 Types d'apprentissage

L'apprentissage est bien entendu un domaine très vaste. Généralement, on distingue trois grandes catégories : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement.

Apprentissage supervisé

Dans le cas de l'apprentissage supervisé, on dispose d'un ensemble de données étiqueté $\mathcal{D} = \{\boldsymbol{x_i}, y_i\}_{i\in[1,n]}$, où $\boldsymbol{x_i} \in \mathbb{X}$ représente l'entrée du modèle, souvent représenté par un vecteur de caractéristiques dans $\mathbb{R}^m$, et $y_i \in \mathbb{Y}$ correspond à la sortie recherchée. L'objectif est d'entraîner un modèle capable de prédire la sortie y associée à une nouvelle entrée $\boldsymbol{x}$ non étiquetée. En d'autres termes, le but est de trouver une fonction $f : \mathbb{X} \longrightarrow \mathbb{Y}$ parmi les membres d'une certaine famille de fonctions $\mathcal{F}$, qui associe une sortie $f(\boldsymbol{x})$ à l'entrée $\boldsymbol{x} \in \mathbb{X}$. Une fonction de perte $\mathcal{L} : (\mathcal{F}, \mathcal{D}) \longrightarrow \mathbb{R}$ est utilisée afin d'évaluer la qualité des différentes fonctions $f \in \mathcal{F}$. On souhaite trouver la fonction qui minimise le risque empirique associé à $\mathcal{D}$:

$$f_{\mathcal{D}} \in argmin_{f \in \mathcal{F}} \ \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f, (\boldsymbol{x_i}, y_i)) \tag{2.22}$$

L'apprentissage supervisé est généralement utilisé soit pour de la *régression* ou de la *classification*. Dans le cas de régression, les sorties à prédire correspondent à des valeurs continues/réelles (e.g., prédiction de la consommation électrique d'une installation), et on parle de classification lorsque les sorties sont catégoriques, i.e., la sortie appartient à une valeur parmi un ensemble de possibilités. Par conséquent, la fonction de perte employée pour évaluer la performance du modèle dans (2.22) diffère selon la nature de la sortie y (i.e., régression ou classification).

Apprentissage non supervisé

Dans le cas de l'apprentissage non supervisé, on dispose d'un ensemble de données $\mathcal{D} = \{\boldsymbol{x}_i\}_{i \in [1,n]}$ non étiqueté, i.e., sans sorties y_i associées. Dans ce cas, le but est d'identifier des caractéristiques communes aux données d'entraînement. Dans cette famille d'algorithmes, on trouve généralement des algorithmes de partitionnement (*clustering* en anglais), où on cherche à classer les données dans des groupes homogènes selon les caractéristiques $\boldsymbol{x}$, une fonction de similarité est utilisée pour calculer la distance entre les paires d'exemples. Il est ensuite possible d'utiliser le modèle appris pour prédire l'appartenance d'un nouvel exemple à un ou plusieurs groupes. Dans cette famille, on trouve aussi les algorithmes de réduction de dimensionnalité pour lesquels l'objectif est de projeter les données dans un nouveau sous-espace de dimension inférieur à celui d'origine. Une nouvelle représentation des données est construite, ce qui permet également d'identifier les caractéristiques les plus importantes des données. Des cas d'utilisation possibles sont la compression des données, ou pour pallier au fléau de dimensionnalité qu'on discutera dans la prochaine section.

Apprentissage par renforcement

L'apprentissage par renforcement (RL : *Reinforcement learning*) est un paradigme d'apprentissage par expérience. Au lieu de donner à la machine les données sur lesquelles elle doit apprendre, les algorithmes RL explorent l'espace de décision en interagissant avec l'environnement afin d'atteindre un certain objectif. Il s'agit d'un fonctionnement itératif où à chaque itération, l'apprenant (i.e., l'agent) fait des observations et entreprend des actions dans son environnement, et en retour un score (i.e., une récompense) lui est attribué selon ses performances. Son objectif est d'apprendre à agir d'une manière qui maximisera ses récompenses au fil du temps.

2.2.2 Défis et limites

Dans cette section, nous présentons quelques défis qui peuvent être rencontrés dans un projet ML, avec une attention particulière à l'apprentissage supervisé, qui est le paradigme d'apprentissage que nous adoptons dans nos projets. Voici quelques exemples :

- **Données insuffisantes.** L'apprentissage nécessite un grand ensemble de données. En effet, même pour des problèmes très simples, des milliers d'exemples sont généralement nécessaires.
- **Données d'entraînement non représentatives.** Pour bien généraliser, il est crucial que les données d'entraînement soient représentatives des nouveaux exemples auxquels on souhaite généraliser. Dans le cas contraire, le modèle a moins de chances de faire des prédictions correctes.
- **Déséquilibre de classes.** Pour les problèmes de classification, il est fréquent d'avoir des données déséquilibrées, i.e., les classes ne sont pas représentées de façon égale dans l'échantillon. Ce problème augmente la difficulté d'apprentissage et peut induire en erreur avec des scores trop optimistes. Cependant, plusieurs techniques existent pour pallier ce problème, il est possible d'adapter les données en ajoutant des exemplaires de la classe minoritaire afin de rééquilibrer les données, ou bien d'adapter le modèle d'apprentissage en affectant un poids plus important aux données de la classe minoritaire. Ainsi, une erreur de classification d'une classe minoritaire est pénalisée plus fortement qu'une erreur de classification d'une classe majoritaire. Il est également recommandé d'utiliser différentes métriques pour mesurer la performance du modèle, e.g., sensibilité (*recall* en anglais), précision, etc.
- **Données de mauvaise qualité.** Le modèle tend a être moins performant si les données d'entraînement sont pleines d'erreurs et de valeurs manquantes ou aberrantes (i.e., outliers). Une phase de pré-traitement des données est nécessaire pour : éliminer l'information non pertinente, mettre à l'échelle les données, etc.
- **Surapprentissage.** Le surapprentissage (*overfitting* en anglais) se produit lorsqu'un modèle apprend les détails et le bruit des données d'apprentissage. Ceci peut avoir un impact négatif sur les performances du modèle lorsqu'il est utilisé sur des nouvelles données (i.e., mauvaise généralisation). Parmi les solutions possibles pour remédier à ce problème est l'utilisation d'une méthode de régularisation.
- **Fléau de dimensionnalité.** Il s'agit d'un problème rencontré sur les données de grande dimension. Généralement, lorsque la dimensionnalité augmente, le nombre d'exemples de données requis pour une bonne performance augmente de façon exponentielle. Afin de pallier à ce problème, il est possible d'augmenter le nombre de

données, ou de réduire la dimensionnalité en sélectionnant un sous-ensemble des caractéristiques (*feature selection*) ou encore d'utiliser une méthode de réduction de dimensionnalité.

— **Interprétabilité.** Ceci correspond à l'explicabilité des modèles. Plusieurs modèles agissent comme des boites noires et il est alors difficile de comprendre comment le système aboutit à une prédiction.

2.2.3 Méthodes d'apprentissage

Dans cette section, nous présentons quelques algorithmes d'apprentissage qui ont été utilisés dans le cadre de cette thèse, en se concentrant sur les méthodes d'apprentissage supervisé et plus précisément sur les algorithmes de classification. Nous avons choisi de répartir cette section en deux parties. Une première partie est dédiée aux méthodes classiques, alors que la deuxième partie couvre les méthodes d'apprentissage profond.

Méthodes classiques

Nous décrivons dans cette partie quelques algorithmes populaires d'apprentissage machine. Pour plus de détails, consulter [34, 35].

Régression logistique. La régression logistique est couramment utilisée pour estimer la probabilité qu'un exemple appartienne à une classe particulière. Il s'agit d'un modèle simple et linéaire qui calcule une somme pondérée des caractéristiques d'entrée (plus un terme de biais). Une fonction sigmoide (2.23) est utilisée en sortie afin de produire un nombre compris entre 0 et 1 correspondant à la probabilité d'appartenance à une classe donnée. Dans le cas de classification binaire (i.e., $y_i \in \{0, 1\}$), si la probabilité estimée est supérieure à 50%, l'exemple appartient à la classe 1, 0 sinon.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.23}$$

Machine à vecteurs de support. Une machine à vecteurs de support (SVM : *Support Vector Machine*) est un modèle d'apprentissage populaire et polyvalent. Il est capable d'effectuer de la classification linéaire en cherchant des hyperplans séparateurs entre les différentes classes, ou encore de la classification non linéaire en projetant les données vers un autre espace de dimension supérieur où elles deviennent linéairement séparables (astuce du noyau). De plus, la méthode fonctionne aussi pour le cas de régression.

Arbres de décision. Comme les SVM, les arbres de décision sont des algorithmes d'apprentissage machine polyvalents qui peuvent effectuer à la fois des tâches de classification et de régression. Ils sont considérés intuitifs et très faciles à interpréter puisqu'ils peuvent être visualisés sous un format d'arborescence (tant que la profondeur de l'arbre est d'une valeur raisonnable). Les arbres de décision cherchent à partitionner les données en se basant sur les caractéristiques F de l'ensemble d'apprentissage afin de déduire la sortie y recherchée. Initialement, l'algorithme commence à la racine de l'arbre et cherche à identifier la caractéristique qui servira à diviser les données d'apprentissage $\mathcal{D}$ en deux sous-ensembles : gauche $\mathcal{D}_l$ et droit $\mathcal{D}_r$, créant ainsi deux branches. Le choix de la caractéristique F_i est déterminé en minimisant une fonction de coût :

$$J(\mathcal{D}, F_i) = \frac{|\mathcal{D}_l|}{|\mathcal{D}|} G(\mathcal{D}_l) + \frac{|\mathcal{D}_r|}{|\mathcal{D}|} G(\mathcal{D}_r), \tag{2.24}$$

où $|\mathcal{D}|$, $|\mathcal{D}_l|$ et $|\mathcal{D}_r|$ correspondent aux tailles des ensembles de données du noeud père et des noeuds fils gauche et droit, respectivement. Dans le cas de classification, $G(\mathcal{D})$ correspond à l'indice de diversité de Gini exprimé par :

$$G(\mathcal{D}) = 1 - \sum_{c_i \in C} P(c_i)^2, \tag{2.25}$$

où C est l'ensemble des classes de sortie et $P(c_i)$ est le ratio obtenu en divisant le nombre d'exemples de la classe c_i par le nombre total des exemples de l'ensemble, i.e., $P(c_i) = \frac{|c_i|}{|\mathcal{D}|}$. À noter qu'il existe d'autres indices de diversité, tels que l'entropie et l'erreur de classification.

Suivant un processus itératif, la procédure est répétée à chaque nœud enfant jusqu'à ce qu'une limite de profondeur maximale est atteinte, ou bien jusqu'à l'obtention d'une feuille (i.e., les exemples d'apprentissage du nœud appartiennent tous à la même classe).

Forêts aléatoires. *Random Forest* (RF) en anglais, est une méthode d'apprentissage ensembliste où un groupe de classifieurs d'arbres de décision est utilisé. Les prédictions de chaque arbre sont regroupées, et la classe avec le plus de votes est choisie. Chaque arbre de décision est construit en utilisant un sous-ensemble aléatoire de données, ce qui augmente la généralisation (et limite le surapprentissage) et résulte souvent en des meilleurs résultats qu'avec un prédicteur individuel. En revanche, les forêts aléatoires sont considérées de type boite noire et donc pas aussi interprétables que les arbres de décision. L'algorithme général peut être résumé dans les étapes suivantes :

1. Sélectionner aléatoirement un sous-ensemble de données $\mathcal{D}_{sub}$ à partir de l'ensemble

d'entraînement $\mathcal{D}$

2. Construire un arbre de décision en utilisant $\mathcal{D}_{sub}$. À chaque noeud :
   (a) Sélectionner aléatoirement un sous-ensemble de caractéristiques $F_{sub} \subset F$
   (b) Diviser le noeud en suivant la procédure décrite plus-haut, i.e., choisir la caractéristique qui minimise la fonction de coût (2.24), cependant, en considérant uniquement le sous-ensemble F_{sub}
3. Répéter les étapes 1. et 2. pour construire k arbres de décisions, où k est un hyperparamètre
4. Regrouper les prédictions obtenues par les k arbres de décision et retourner le résultat ayant le plus de votes

Méthodes d'apprentissage profond

L'apprentissage profond [36] est un sous-domaine de l'apprentissage machine qui regroupe des méthodes d'apprentissage basées sur les réseaux de neurones artificiels (ANN : *Artificial Neural Network*). Les ANNs sont inspirés des réseaux de neurones biologiques présents dans le cerveau humain. Ils sont polyvalents, puissants et évolutifs, ce qui les rend idéaux pour s'attaquer à des tâches d'apprentissage vastes et très complexes telles que la classification d'images, la reconnaissance vocale, la vision par ordinateur, etc. Étonnamment, les ANNs existent depuis assez longtemps, ils ont été introduits pour la première fois en 1943 [37], mais ce n'est que depuis quelques années que nous avons assisté à une grande vague d'intérêt pour l'apprentissage profond et les ANNs. Dans cette section, nous nous limiterons à décrire brièvement les ANNs, ainsi que leur utilisation pour des tâches de classification sur des données représentées par des graphes.

Réseaux de neurones artificiels. Les réseaux de neurones sont des modèles complexes qui peuvent être utilisés pour la régression et la classification. Un ANN comporte trois couches ou plus de neurones artificiels qui sont interconnectées. La première couche d'un ANN est constituée de neurones d'entrée dont le rôle est de transmettre les données d'entrée aux couches intermédiaires. Généralement, il y a autant de neurones dans la couche d'entrée que de caractéristiques x_i. Les couches intermédiaires, aussi appelées *couches cachées*, sont constituées de neurones qui effectuent des transformations sur la base des entrées reçues de la couche précédente. L'entrée de chaque neurone est associée à un poids w, qui est un paramètre ajustable par le modèle, représentant la force de la connexion. Une somme pondérée des entrées est donc calculée à laquelle une transformation est appliquée en utilisant une fonction d'activation (e.g., ReLU, tanh, etc.). Il est possible d'empiler plusieurs couches cachées, ce qui

permet d'obtenir des modèles plus complexes qui ont le potentiel de surpasser les méthodes classiques sur certaines tâches. Finalement, la couche de sortie est la couche finale du réseau où les prédictions sont obtenues. Elle rassemble les sorties de la dernière couche cachée. Les neurones de cette couche possèdent leurs propres poids qui sont appliqués avant la sortie finale. Une fonction sigmoïde (i.e., voir (2.23)) ou softmax est souvent utilisée à la sortie lorsqu'il s'agit d'un problème de classification binaire ou multi-classes, respectivement.
Pendant de nombreuses années, les chercheurs ont lutté pour trouver un moyen d'entraîner les ANNs mais sans succès. Mais en 1986, Rumelhart et *al.* [38] ont publié un article révolutionnaire qui a introduit l'algorithme de rétropropagation, qui est un algorithme d'optimisation permettant d'ajuster les paramètres (i.e., les poids w) d'un réseau de neurones. La rétropropagation calcule le gradient de la fonction de perte $\mathcal{L}$ par rapport aux poids du réseau. Ceci a également rendu possible l'utilisation des méthodes de *descente du gradient* pour l'entraînement des ANNs.

Réseaux de neurones en graphes. Les réseaux de neurones en graphes (GNN : *Graph Neural Network*) constituent un outil efficace pour apprendre avec des données structurées en graphes. Ils peuvent être utilisés pour des tâches telles que la classification des nœuds, la prédiction de liens, la classification de documents et même l'optimisation combinatoire. Ils ont été initialement introduits par Gori et *al.* [39] et développés par Scarselli et *al.* [40]. Un GNN vise à apprendre une représentation d'un nœud en agrégeant les informations des nœuds voisins. Ces représentations sont ensuite utilisées pour obtenir les prédictions. Dans le cadre de cette thèse, nous nous intéressons uniquement au problème de classification des noeuds.
Étant donné un graphe $G = (V, E)$, où V est l'ensemble des nœuds et E l'ensemble des arêtes, chaque nœud $v \in V$ est caractérisé par son vecteur de caractéristiques $\boldsymbol{x}_v$. La représentation du noeud v à l'itération $k = 0, 1, \ldots, K$ est dénotée par le vecteur $\boldsymbol{h}_v^{(k)}$ avec $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$. Soit $\mathcal{N}(v)$ l'ensemble des nœuds voisins (adjacents) de $v \in V$. A chaque itération $k > 0$ et pour chaque noeud $v \in V$, une fonction d'agrégation $aggr$ est appliquée pour calculer un nouveau vecteur $\boldsymbol{a}_v^{(k)}$ contenant l'information agrégée des nœuds voisins :

$$\boldsymbol{a}_v^{(k)} = aggr(\{\phi^{(k)}(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{h}_v^{(k-1)}) : u \in \mathcal{N}(v)\}), \qquad \forall v \in V,$$

avec $\phi^{(k)}$ une fonction correspondant à un ANN, dont le rôle est d'identifier et d'extraire l'information pertinente contenue dans les noeuds voisins. La fonction $aggr$ doit être invariante aux permutations, elle est souvent implémentée en utilisant des fonctions telles que la fonction de somme, de moyenne, de min/max, etc. L'étape suivante consiste à mettre à jour les représentations des noeuds en prenant en compte le vecteur $\boldsymbol{a}_v^{(k)}$:

$$\boldsymbol{h}_v^{(k)} = \psi^{(k)}([\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}]), \qquad \forall v \in V,$$

où $\psi^{(k)}$ est un autre ANN et [.] est l'opération de concaténation.
Au fur et à mesure des itérations, les nœuds collectent des informations en provenance des nœuds plus éloignés. A la dernière itération K, la représentation $\boldsymbol{h}_v^{(K)}$ d'un noeud $v \in V$ peut être utilisée pour prédire son étiquette, dénotée par $\hat{y}_v$:

$$\hat{y}_v = out(\boldsymbol{h}_v^{(K)}), \qquad \forall v \in V.$$

La fonction *out* correspond à un ANN qui produit la sortie finale. Trois modèles d'ANN sont donc utilisés.
Dans la littérature, plusieurs architectures ont été proposées. Il est possible, par exemple, d'inclure un vecteur de caractéristiques des arêtes, d'avoir un graphe orienté, de classifier les arêtes au lieu des noeuds, etc. Un bon aperçu des architectures qui ont été proposées dans la littérature peut être trouvé dans [41, 42].

2.3 Apprentissage machine et optimisation combinatoire

Au cours des dernières années, les chercheurs se sont intéressés de plus en plus à l'utilisation des algorithmes ML pour résoudre des problèmes combinatoires, ou pour accélérer les méthodes existantes [43, 44].

Dans le contexte de branch-and-price, Václavík et *al.* [45] ont proposé une méthode basée sur le ML pour estimer une borne supérieure de la valeur optimale du SP. À chaque itération de la GC, les valeurs duales sont transformées en un vecteur de caractéristiques qui est ensuite passé en entrée à un modèle de régression. La borne supérieure estimée par le modèle est ensuite utilisée comme contrainte dans le SP. Ceci a comme effet de réduire l'espace de recherche et d'accélérer le temps de calcul pris par les SPs.

Dans le domaine aérien, Yaakoubi et al. [46] ont proposé un modèle d'apprentissage profond visant à accélérer l'optimisation d'un problème de rotations d'équipages. Le but est de prédire les probabilités de connexions entre les vols en exploitant les solutions historiques, puis d'identifier des groupes de vols qui peuvent être utilisés pour construire une solution initiale à fournir pour la méthode ADC [11]. Dans le même contexte, mais cette fois-ci en utilisant I^{2}CG, Tahir et *al.* [47] ont exploité les prédictions de connexion entre les vols dans les SPs afin de réduire l'espace de recherche et de favoriser la génération de colonnes qui ont de grandes chances d'appartenir à des solutions optimales ou quasi-optimales. Toujours dans le domaine aérien, Quesnel et al. [48] ont proposé un algorithme B&P exploitant l'AM pour

la résolution du problème d'horaires personnalisés. Un modèle d'apprentissage profond est employé afin de prédire les rotations qui ont de grandes probabilités d'être assignées à chaque membre d'équipage. Les prédictions obtenues sont ensuite utilisées afin de réduire la taille des réseaux des SPs. Les résultats obtenus montrent des réductions importantes en temps de calcul tout en produisant des solutions quasi-optimales.

Dans le contexte de B&B, plusieurs chercheurs ont exploré l'utilisation de l'apprentissage machine pour la sélection de variables sur lesquelles brancher [49–51]. La plupart de ces méthodes cherchent à apprendre une stratégie de branchement imitant le branchement fort (*Strong branching* en anglais), qui est une stratégie qui réduit considérablement le nombre de nœuds explorés, mais qui est coûteuse en terme de calcul. D'autres travaux se sont intéressés à apprendre de nouvelles stratégies d'exploration pour déterminer le prochain noeud à explorer [52]. Le but est de construire des modèles capables de guider la recherche vers les zones les plus prometteuses de l'arbre de branchement. Par ailleurs, dans le contexte de *branch-and-cut* (i.e., combinaison de B&B et une méthode de plans coupants), certains travaux ont montré des résultats prometteurs qui exploitent de l'apprentissage machine pour la sélection des plans coupants à ajouter à chaque noeud [53].

D'autres travaux de recherche ont été consacrés au problème de voyageur de commerce (TSP : *Traveling Salesman Problem*), qui est un problème d'optimisation combinatoire classique où l'objectif est de visiter un ensemble de nœuds exactement une fois tout en minimisant la distance parcourue. Joshi et *al.* [54] ont utilisé un modèle de GNN convolutionnel qui prend en entrée le graphe entier et produit en sortie une matrice d'adjacence avec les probabilités des arêtes associées. Les probabilités estimées par le modèle sont ensuite utilisées pour construire une tournée valide à l'aide d'un algorithme de recherche en faisceau (*beam search* en anglais). D'autres chercheurs ont choisi d'employer des méthodes d'apprentissage par renforcement [55, 56] en exploitant des modèles encodeur-décodeur. Le rôle de l'encodeur est d'apprendre de nouvelles représentations des nœuds, alors que le décodeur construit une route valide de manière itérative, en se basant sur les représentations des nœuds et en masquant les nœuds déjà visités. Suivant la même approche, d'autres travaux de recherche [57] se sont penchés sur la résolution du CVRP, qui est considéré une généralisation du TSP pour le cas avec plusieurs véhicules. En plus des méthodes d'apprentissage supervisé et par renforcement, des méthodes hybrides ont été proposées, combinant le ML et d'autres méthodes variées (e.g., programmation dynamique, recherche locale, programmation par contraintes, etc.).

Les méthodes basées sur l'apprentissage machine se sont révélées assez efficaces pour des problèmes qui sont résolus de façon répétitive, surtout lorsque les données du problème ne changent que légèrement. À titre d'exemple, on trouve des applications au problème d'enga-

gement d'unités (*unit commitment problem*) [58] et le problème d'emplacement d'installations (*facility location problem*) [59].

CHAPITRE 3 ORGANISATION DU TRAVAIL

L'objectif général de cette thèse est d'accélérer la méthode de GC en utilisant les techniques de l'apprentissage machine. Comme il a été susmentionné, la GC est considérée la méthode exacte de pointe pour la résolution de plusieurs problèmes industriels. Cependant, malgré son efficacité, la méthode peut prendre énormément de temps sur les instances de grande taille. Ceci nous incite à chercher de nouvelles approches permettant de réduire les temps de calcul, soit du côté du PMR ou bien du SP.

Dans le premier sujet de cette thèse, nous nous concentrons sur le côté du PMR et nous proposons une méthode ML pour sélectionner les colonnes à ajouter à chaque itération. En effet, il a été démontré à plusieurs reprises que l'ajout de plusieurs colonnes par itération améliore la convergence de la GC et réduit le nombre total des itérations. Lorsqu'un grand nombre de colonnes est généré, il est courant de les trier dans l'ordre croissant de leurs coûts réduits, et d'en ajouter un nombre maximal au PMR à chaque itération. Cependant, cette stratégie de sélection qui n'est basée que sur le coût réduit n'est pas nécessairement la plus efficace. Dans ce premier sujet, nous cherchons à explorer une meilleure stratégie de sélection qui tire profit de l'historique des exécutions, et qui favorise l'ajout des colonnes qui ont une grande probabilité de faire partie de la solution optimale du PM.

Dans le deuxième sujet, nous nous intéressons aux problèmes résolus avec la GC où le SP correspond à un SPPRC ou une de ses variantes. Dans ce cas, une façon d'accélérer la résolution du SP consiste à réduire la taille du réseau sous-jacent. Il est possible de construire un réseau réduit qui est dérivé du réseau original, où l'on ne garde que les arcs qui ont une forte probabilité de faire partie des solutions des PMRs. Ceci a pour effet de diminuer considérablement le nombre d'étiquettes à propager dans le réseau, réduisant ainsi le temps de calcul total consacré aux SPs. En effet, en analysant les données recueillies lors de la résolution d'une instance du VRPTW, nous avons constaté que les arcs de la solution optimale du PM ne représentent que 1% à 1,5% du nombre total des arcs. Notre but n'est pas de prédire exactement la solution optimale (i.e., les $\sim$ 1% des arcs), tâche qui s'avère ardue, mais seulement de réduire au maximum la taille du réseau en utilisant un modèle de prédiction.

Dans le troisième sujet, nous développons une heuristique pour la résolution d'un problème d'optimisation après un changement mineur de ses paramètres. L'idée découle de l'observation suivante : étant donné une instance et sa solution $\mathcal{S}_1$, si l'instance est réoptimisée après un changement mineur de ses données, la nouvelle solution $\mathcal{S}_2$ sera très similaire à $\mathcal{S}_1$. Par conséquent, au lieu de résoudre le problème à partir de zéro, l'objectif est de prédire les

parties de la solution qui ont une forte probabilité de rester inchangées. Nous traitons ce problème en appliquant le paradigme 'ML $\longrightarrow$ GC', i.e., le modèle de prédiction est utilisé pour obtenir les prédictions sur les parties de la solution à fixer, cette information est ensuite passée comme entrée au solveur pour compléter la solution.

CHAPITRE 4 ARTICLE 1: MACHINE-LEARNING-BASED COLUMN SELECTION FOR COLUMN GENERATION

Cet article a été publié dans la revue *Transportation Science* le 30 Juin 2021.

Référence : M. Morabit, G. Desaulniers, A. Lodi (2021), Machine-Learning–Based Column Selection for Column Generation. *Transportation Science* 55(4) :815-831.

Abstract

Column generation (CG) is widely used for solving large-scale optimization problems. This article presents a new approach based on a machine learning (ML) technique to accelerate CG. This approach, called *column selection*, applies a learned model to select a subset of the variables (columns) generated at each iteration of CG. The goal is to reduce the computing time spent reoptimizing the restricted master problem at each iteration by selecting the most promising columns. The effectiveness of the approach is demonstrated on two problems : the vehicle and crew scheduling problem, and the vehicle routing problem with time windows. The ML model was able to generalize to instances of different sizes, yielding a gain in computing time of up to 30%.

Keywords : Column generation, machine learning, column selection.

4.1 Introduction

Column generation (CG) is an iterative method used to solve the linear relaxation of large-scale optimization models which involve a very large number of variables (columns) that cannot be considered at once. The method exploits the fact that the majority of these columns will not be part of an optimal solution. It starts with a subset of variables and generate new ones that have the potential to improve the current solution, i.e., variables with a negative reduced cost (assuming a minimization problem), until an optimal solution is obtained and proved optimal. At each iteration, it solves a restricted master problem (RMP) and a pricing problem (PP). The RMP is the original linear program (LP) restricted to a subset of its variables and is solved by an LP solver such as the primal simplex algorithm. The PP is application-specific and aims at finding negative reduced cost columns with respect to the dual solution of the current RMP. When such columns are found, they are added to the RMP to start a new iteration. Otherwise, the CG process stops, certifying the optimality of the current RMP solution for the whole LP.

When solved by CG, several large-scale models, such as the set-partitioning model, are prone

to high degeneracy, which results in slow convergence. For these problems, many columns can be generated at each iteration and added to the RMP. However, doing so just increases the difficulty of dealing with degeneracy and, thus, becomes counterproductive. In this context, we propose in this paper a new approach to accelerate CG that relies on machine learning (ML) techniques and focuses on “column selection". More precisely, at each CG iteration, a classification algorithm is used to select promising columns when a large set of columns is generated. The learning algorithm is trained on data collected from previous executions. Although the ideas presented here can be applied to various problems, the effectiveness of the approach will be demonstrated on two specific problems : the vehicle and crew scheduling problem (VCSP) and the vehicle routing problem with time windows (VRPTW).

4.1.1 Literature review

CG has been used to solve many combinatorial optimization problems, a classic example being the cutting stock problem [2]. The authors proposed to solve the linear relaxation of the original model by considering only a subset of the variables representing feasible cutting patterns and generating new columns as needed by solving a knapsack problem. The method has since been successfully applied to other problems, including vehicle routing problems, crew scheduling problems, and many other problems. To derive integer solutions, the method is embedded in a branch-and-bound framework, i.e., CG is performed at every node of the search tree, yielding a branch-and-price algorithm [1, 4].

The CG algorithm is well known to have convergence issues due to high degeneracy and dual variable instability, especially on problems formulated as set partitioning problems. In the literature, several strategies have been developed to address these difficulties [5, 6], such as removing redundant constraints from the problem or perturbing them by adding penalized slack and surplus variables [7]. Other stabilization techniques can be used to reduce the oscillation of the dual values from one solution to another. These techniques can, for example, force the dual variables to remain in relatively small intervals around what is called a stability center, which corresponds to the best estimate of the optimal dual values. A stabilization function (see, e.g., [8]) is used to penalize dual solutions that are far from the stability center. The penalties and intervals can be adjusted dynamically throughout the execution of CG. [9] propose, among others, a smoothing technique that considers a dual vector combining the current dual solution and those form previous iterations. The use of an interior point or primal-dual method (e.g., [10,60]) for solving the RMP produces central dual solutions which can also help to reduce the number of CG iterations and speed up the solution process.

Other methods more specific to problems involving set partitioning constraints and suffering

from high degeneracy were explored. Elhallaoui et *al.* [11] introduced a dynamic constraint aggregation method (DCA) that consists in reducing the number of set partitioning constraints in the RMP by aggregating some of them and revising this aggregation as needed to guarantee optimality. Further improvements of this method were introduced in [12,61] with the aim of maintaining a higher level of aggregation and further reducing degeneracy. DCA has fostered the development of the improved primal simplex algorithm (IPS, [13]) that can tackle general linear programs without CG while reducing degeneracy. The method decomposes the problem in two subproblems : a reduced problem (RP) and a complementary problem (CP). The RP is a linear program that contains a subset of the constraints and of the variables which are called the compatible variables, i.e., those whose coefficient columns can be written as linear combinations of the columns that are part of the current solution. The CP is also a linear program that is solved to prove the optimality of the RP solution for the original problem or, otherwise, to identify incompatible variables for building a new RP problem. Exploiting the IPS theoretical results, Zaghrouti et *al.* [14,15] developed variants of the integral simplex using decomposition algorithm for solving set-partitioning problems. This algorithm solves alternately a RP and a CP to find a sequence of improving integer solutions. More recently, to solve set partitioning problems, Tahir et *al.* [16] introduced a combination of ISUD and CG that is called integral column generation (ICG). At each CG iteration, ISUD solves the RMP to yield an integer solution as opposed to the standard CG that produces fractional solutions most of the times. Using a dual solution obtained by solving a linear system of equations involving the dual values of the last CP solved, the CG PP is solved to generate new columns to be handled by ISUD in the next RMP.

To the best of our knowledge, there has been no work dedicated to column selection in the CG literature. However, it has been shown many times that generating more than one column per iteration improves CG convergence. On the other hand, when many columns are generated at a given iteration, it is common to impose a maximum number of columns to add to the RMP and to select the columns to add in increasing order of their reduced cost.

In the last few years, researchers have become increasingly interested in the use of ML algorithms to solve combinatorial optimization problems, or to accelerate existing optimization methods. A good overview on ML techniques for combinatorial optimization is given by Bengio et *al.* [43]. The techniques can fall into one of two categories. The first category comprises the *imitation learning* methods, where we know in advance what (not necessarily optimal) behavior is expected for the method, and we want to replace expensive calculations with a quick approximation. The second category includes methods based mainly on reinforcement learning, where current solutions are not satisfactory and we look for better ways to solve

the problem.

Little work has been done so far in the CG or branch-and-price context. In [45], the authors proposed a regression model to estimate at each CG iteration an upper bound on the optimal value of the PP. This upper bound is then used to reduce the PP search space and accelerate its computational time. Other works have explored the use of ML in branch-and-bound algorithms. [49] and [50] studied variable selection for branching. Their goal is to imitate the *strong branching* strategy, a strategy that significantly reduces the number of nodes to explore, but is time consuming. Other branch-and-bound methods using learning algorithms are described in the survey of Lodi and Zarpellon [62].

4.1.2 Paper contribution and structure

The main contribution of this paper is the design of a new CG algorithm that incorporates column selection by ML to reduce degeneracy and computational time. Column selection is applied at every CG iteration to select promising columns to add to the RMP. The selection is performed in a very short computational time by a ML model trained on data collected from previously solved instances. The algorithm is tested on two well-known problems from the literature (the VCSP and the VRPTW), and can also be applied to other problems that are solved using CG. Note that we focus on solving the initial linear relaxation only, i.e., at the root node of a search tree, but the proposed column selection strategy is also applicable at other nodes.

The paper is organized as follows. In Section 2, we present a generic linear relaxation model and describe a standard CG algorithm to solve it. In Section 3, we focus on the CG/ML framework that we propose by describing in details the different steps, from data generation, to training and prediction. Sections 4 and 5 will be devoted to our two case studies, namely, the VCSP and the VRPTW, respectively. In section 6, we report and discuss the results of our computational experiments. Finally, conclusions are drawn in Section 7.

4.2 Basic model and column generation

Let us consider the following generic linear program, called the master problem (MP) in a CG context :

$$\min_{\theta} \quad \sum_{p \in \mathcal{P}} c_p \theta_p \tag{4.1}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}} \mathbf{a_p} \theta_p = \mathbf{b}, \tag{4.2}$$

$$\theta_p \geq 0, \quad \forall p \in \mathcal{P}, \tag{4.3}$$

where $\mathcal{P}$ is the index set of the variables θ_p ; $c_p \in \mathbb{R}$ and $\mathbf{a_p} \in \mathbb{R}^m$ are the cost coefficient and constraint coefficient vector of θ_p, respectively ; and $\mathbf{b} \in \mathbb{R}^m$ is the right-hand side vector of the constraints (4.2). Note that some or all these constraints could also be inequalities. We assume that the number of variables θ_p is very large and the set of objects (e.g., vehicle schedules or cutting patterns) associated with these variables can be implicitly modeled as solutions of an optimization problem (e.g., a shortest path problem or a knapsack problem). To solve this MP, CG proceeds as follows (see Figure 4.1). At each iteration, it solves a RMP that considers a subset $\Omega \subseteq \mathcal{P}$ of the columns. It yields a primal solution x (assuming that $\theta_p = 0$, $\forall p \in \mathcal{P} \setminus \Omega$) and a dual solution given by the dual values $\pi \in \mathbb{R}^m$ associated with constraints (4.2). This dual solution is then used to find new negative reduced cost variables θ_p, by solving the following PP :

$$\bar{c} = \min_{p \in \mathcal{P}} \{c_p - \pi^T \mathbf{a_p}\} \tag{4.4}$$

If negative reduced cost columns are found, they are added to the RMP, which is then re-optimized to obtain new dual values. Otherwise, the current RMP solution is optimal for the MP and CG stops. Note that, for some problems, the search for negative reduced cost columns can be distributed across several PPs.

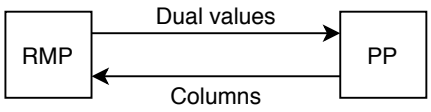


FIGURE 4.1 Standard CG process

Several strategies can be applied to accelerate column generation (see, [32]). We present here a few basic ideas that were used in our computational experiments.

- At each CG iteration, several negative reduced cost columns can be added to the RMP. Typically, this strategy reduces the total number of iterations and the overall time spent solving the PP. The possibility to generate several columns at once depends on the algorithm used to solve the PP. In our two case studies, the PP is formulated as a shortest path problem with resource constraints (SPPRC, [18]) and solved by dynamic programming (more precisely, a labeling algorithm) which offers this possibility at no extra computational effort.
- Generating a large number of columns in a given iteration may not be effective if the columns added to the RMP are not sufficiently different from each other. To favor column diversity, we apply a simple pre-selection procedure which has proven to be

effective for set-partitioning models (possibly with side constraints) such as the ones used for the VCSP and the VRPTW. In these models, the set-partitioning constraints enforce the covering of a task (a bus trip in the VCSP or a customer service in the VRPTW) exactly once. Diversified columns should not cover the same tasks as much as possible. Two columns that do not cover the same tasks are said to be disjoint. The pre-selection procedure proceeds as follows. First, the columns are sorted in increasing order of their reduced cost. Second, following this order, the columns are distributed in blocks (numbered 1, 2, 3, ...) of disjoint columns using the following rule : a column is put in the block numbered b if this column is not disjoint from at least one column in each block 1 to $b-1$, and disjoint from all columns already in block b. Consequently, the first column is put in block 1, the second in block 1 if it is disjoint from the first column or in block 2 otherwise, and so on. Then, only the columns in the first n_{blks}^{max} blocks of disjoint columns are kept, where n_{blks}^{max} is a predefined parameter value.

- Another strategy consists in removing columns from the RMP when it becomes large. The removed columns can still be generated again if necessary. In our tests, this strategy is implemented using two parameters : a minimum (n_{cols}^{-}) and a maximum (n_{cols}^{+}) number of columns to keep in the RMP. Once the maximum number is reached, the $n_{cols}^{+} - n_{cols}^{-}$ columns with the largest reduced costs are removed from the RMP.
- Heuristics can be used to speed-up the PP. For our case, we try to generate negative reduced cost columns using a heuristic labeling algorithm that does not consider all resource constraints in the dominance rule, nor all arcs in the PP network. When no negative reduced cost column can be found heuristically, an exact algorithm is invoked to ensure the exactness of the algorithm.

4.3 Methodology

At each CG iteration, after reoptimizing the RMP and solving the PP, a set $\mathcal{G}$ of columns is obtained. The goal is to select a subset of columns $\mathcal{G}^S \subseteq \mathcal{G}$ to be added to the RMP. This selection of columns can be considered as a binary classification problem (supervised learning problem), where we try to classify each generated column into one of two classes $y \in \{0, 1\}$, i.e., 1 if the column is to be selected and 0 otherwise. The next sections describe the different steps required to build the data set, as well as details on the algorithm that will be used for the training.

4.3.1 Data collection

As with all supervised learning problems, a data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ is required, where n is the size of the dataset, $\boldsymbol{x}_i$ is the vector representing the features of

column i, and $y_i \in \{0, 1\}$ the column label (our target/output). Before talking about the features to be extracted from the generated columns, we start by describing how the labels are assigned, which is equivalent to answering the following question : which columns are considered as “promising" and should be selected ?

Labels

The idea is to add a small subset of the generated columns to the RMP so that it remains fast to reoptimize. On the other hand, we also want a maximum reduction of the objective value at each CG iteration. For these reasons, a good option is to add only the generated columns that will take a positive value after the RMP reoptimization, i.e., we are looking for a set of columns that will ensure a maximal decrease of the objective value and that is of minimal size. This column selection problem is, thus, a hierarchical bi-objective problem. Using a weighted sum of the objective functions, it can be formulated as the following mixed integer linear program (MILP), which is a copy of the RMP with additional constraints and integer variables.

Let ℓ be the CG iteration number, Ω_ℓ the set of columns present in the RMP at the start of iteration ℓ, and $\mathcal{G}_\ell$ the generated columns at this iteration. For each column $p \in \mathcal{G}_\ell$, we define a decision variable y_p that takes value 1 if column p is selected and 0 otherwise. To minimize the number of selected columns, a sufficiently small penalty ϵ is incurred for each selected column.

The proposed MILP is as follows :

$$\min_{\theta, y} \quad \sum_{p \in \Omega_\ell \cup \mathcal{G}_\ell} c_p \theta_p + \sum_{p \in \mathcal{G}_\ell} \epsilon y_p \tag{4.5}$$

$$\text{s.t.} \quad \sum_{p \in \Omega_\ell \cup \mathcal{G}_\ell} \mathbf{a_p} \theta_p = \mathbf{b}, \tag{4.6}$$

$$\theta_p \geq 0, \quad \forall p \in \Omega_\ell \cup \mathcal{G}_\ell \tag{4.7}$$

$$\theta_p \leq y_p, \quad \forall p \in \mathcal{G}_\ell \tag{4.8}$$

$$y_p \in \{0, 1\}, \quad \forall p \in \mathcal{G}_\ell. \tag{4.9}$$

Without the y_p variables and constraints (4.8) and (4.9), model (4.5)–(4.9) would exactly be the RMP at iteration $\ell + 1$. Assuming that ϵ is sufficiently small, these variables and constraints are only considered to minimize the set of selected columns. These columns correspond to those for which $y_p = 1$, i.e., the columns y_p associated with positive-valued θ_p variables (due to constraints (4.8)). The subset of columns to be added in the RMP at iteration ℓ is therefore

$$\mathcal{G}_\ell^S = \{p \in \mathcal{G}_\ell : y_p = 1\}. \tag{4.10}$$

The overall procedure is illustrated in Figure 2.

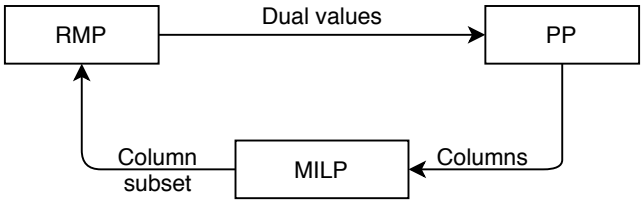


FIGURE 4.2 CG algorithm with MILP column selection

In other words, we are using the above MILP as the “expert" that we want to imitate and learn from.

Column features

In CG, each column represents something different depending on the PP generating the columns. For example, if the PP is a shortest path problem with resource constraints and each column is associated with a path representing, e.g., a route or a schedule, features such as resource values, number of nodes in the path, or distances between nodes can be considered. This is in addition to general features that can be used for any problem such as the cost, reduced cost, dual values of the constraints that the column contributes to. A complete list of the features is given in Sections 4.4 and 4.5 for our two case studies.

4.3.2 Algorithmic requirements

In ML, many classification algorithms exist. In our case, the choice of the appropriate algorithm should take into consideration the following points :

- The selection of a column depends on the presence of other columns in the problem, i.e., $\Omega_\ell \cup \mathcal{G}_\ell$. It is preferable that each column has information about the other columns.
- Based on the previous point, if we consider all the columns at once (taking $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ all at once as input to the ML model), it should be noted that the number of columns n will be different from one iteration to another, and that the result of the prediction should be the same regardless of the columns order (*order invariance*).
- Dual values contain rich information that should be used to determine whether or not a column should be selected. However, the number of dual values depends on the number of constraints in the problem, and the number of constraints to which a column contributes is different from one column to another.
- The ML model should be able to generalize to new instances of a given problem, keeping in mind that the instances will not be necessarily of the same size.

– The purpose of the ML model is to replace the MILP presented above, which is computationally expensive. A fast prediction is desirable.

4.3.3 Graph neural networks for column classification

Graph neural networks (GNNs) form an effective framework for learning with graph structured data. They can be used for tasks like node classification, link prediction, graph prediction, document classification and even combinatiorial optimization. They were initially introduced by Gori et *al.* [39] and further developed by Scarselli et *al.* [40]. A GNN aims at learning a node representation by aggregating information from the neighbor nodes in an iterative manner. This is also known as a message-passing method. The node representations can then be used to produce an output label for node classification tasks.
Due to the effectiveness of these methods in capturing dependencies between nodes, various studies tried to extend the learning approaches on graphs by adopting ideas from deep learning paradigms, and gave birth to several methods based on convolutional neural networks, known as graph convolutional networks (GCN), where the convolution operation has been redefined for graph structured data. GCN methods are divided into two groups : the first group contains the spectral-based methods which are based on spectral graph theory and graph signal processing, and the second one includes the spatial-based methods that agreggate information directly from the neighbor nodes. These approaches are summarized in a recent survey by Wu et *al.* [42]

GNN overview

Given a graph $G = (V, E)$, where V is the set of nodes and E the set of edges, each node $v \in V$ is characterized by its feature vector $\boldsymbol{x}_v$. The goal is to iteratively update the representation of a node (also called *state*) by aggregating information from the neighbor nodes. Let $\boldsymbol{h}_v^{(k)}$ be the representation vector of node $v \in V$ at iteration $k = 0, 1, \ldots, K$. Let $\mathcal{N}(v)$ be the set of neighbor (adjacent) nodes of $v \in V$. As shown in Figure 4.3, the representation vectors are iteratively updated by aggregating the representations of the neighbor nodes as follows. At an iteration $k > 0$, an aggregated function, denoted $aggr$, is first applied to compute an aggregated information vector $\boldsymbol{a}_v^{(k)}$ for each node $v \in V$:

$$\boldsymbol{a}_v^{(k)} = aggr(\{\phi^{(k)}(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{h}_v^{(k-1)}) : u \in \mathcal{N}(v)\}), \qquad \forall v \in V,$$

with $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$ and $\phi^{(k)}$ is a learned function. Function $aggr$ should be invariant to permutations of the node order, such as the sum, mean, and min/max functions. Then, using a function denoted *comb* (which is often a concatenation function), we combine the aggre-

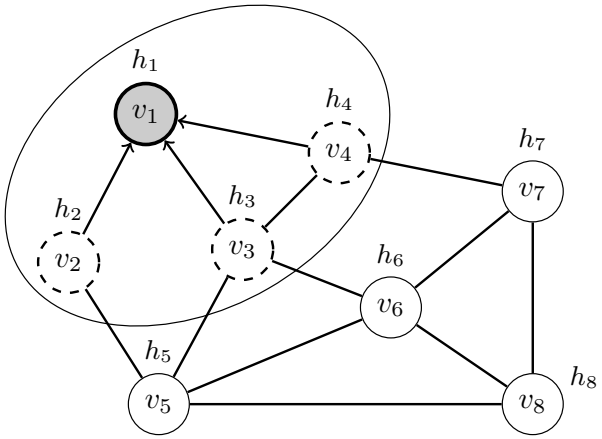


FIGURE 4.3 Representation vector h_1 of node v_1 is updated by aggregating the representations h_2, h_3, h_4 of its neighbor nodes

gated information with the current state of the given nodes to obtain the updated node representation vectors :

$$\boldsymbol{h}_v^{(k)} = \psi^{(k)}(comb(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)})), \qquad \forall v \in V,$$

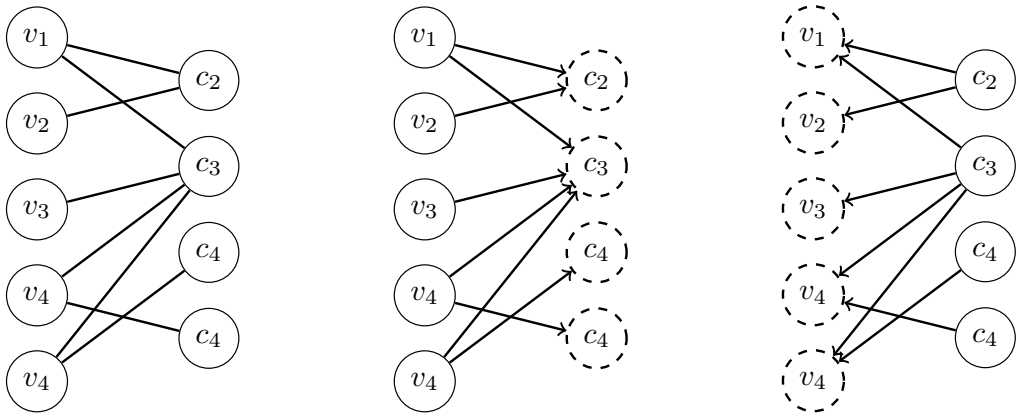
where $\psi^{(k)}$ is another learned function.
As the iterations progress, the nodes collect information from farther away neighbor nodes. At the final iteration K, the representation $\boldsymbol{h}_v^{(K)}$ of a node $v \in V$ can be used to predict its label, denoted l_v, by applying a final learned transformation, denoted *out*, i.e.,

$$l_v = out(\boldsymbol{h}_v^{(K)}), \qquad \forall v \in V.$$

The learned functions $\phi^{(k)}$, $\psi_k^{(k)}$ and *out* are implemented by using multilayer perceptron feedforward neural networks [36].
A variety of architectures can be implemented within a GNN. For example, we can have edge features, a directed graph instead of an undirected one, an edge or graph classification problem, etc. A good summary of the architectures that have been proposed in the literature can be found in [41], where a configurable framework for building complex architectures is also proposed.

Our model

The model presented in the previous section can be used for our column classification task. One obvious architecture consists in representing each column by a node and linking two



(a) Bipartite graph with two node types V and C

(b) Phase 1 : constraint representations update

(c) Phase 2 : column representations update

FIGURE 4.4 Our model represented by a bipartite graph for column classification/selection.

nodes with an edge if their corresponding columns contribute to at least one constraint in common. However, in this case, the number of edges would be very large and the dual value information would be difficult to include in the model. A different architecture uses a bipartite graph with two node types : column nodes V and constraint nodes C. An edge (v, c) exists between a node $v \in V$ and a node $c \in C$ if column v contributes to constraint c (see Figure 4.4a). The advantage of this representation is that we can have feature vectors attached to the constraints nodes, which will allow us to include dual information easily.

Since there are two node types, each iteration proceeds in two phases : a first phase is performed to update the constraint representations (Figure 4.4b), followed by a second phase that updates the column representations (Figure 4.4c). Note that, after a single iteration, every column node contains some information about the other columns that contribute to the same constraints. However, better information may be transmitted by performing additional iterations. The column representations $\boldsymbol{h}_v^{(K)}$, $v \in V$, are then used for predicting labels $y_v \in \{0, 1\}$. The steps are described in Algorithm 1.

As described in the previous section, we start by initializing the representation vectors of both the column and constraint nodes by the feature vectors $\boldsymbol{x}_v$ and $\boldsymbol{x}_c$, respectively (Steps 1 and 2). For each iteration k we perform the two phases : updating the constraint representations (Steps 4 and 5), then the column ones (Steps 6 and 7). The sum function is used for the *aggr* function and the vector concatenation for the *comb* function. The functions $\phi_C^{(k)}$, $\psi_C^{(k)}$, $\phi_V^{(k)}$, and $\psi_V^{(k)}$ are 2-layer feedforward neural networks with ReLU activation functions and *out* is a 3-layer feedforward neural network with a sigmoid function for producing the final probabilities

Algorithm 1 GNN applied to the bipartite graph model

1: $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v, \quad \forall v \in V$
2: $\boldsymbol{h}_c^{(0)} = \boldsymbol{x}_c, \quad \forall c \in C$
3: **for** $k = 1, ..., K$ **do**
4: $\quad \boldsymbol{a}_c^{(k)} = \sum_{u \in \mathcal{N}(c)} \phi_C^{(k)}(\boldsymbol{h}_c^{(k-1)}, \boldsymbol{h}_u^{(k-1)}), \quad \forall c \in C$
5: $\quad \boldsymbol{h}_c^{(k)} = \psi_C^{(k)}([\boldsymbol{h}_c^{(k-1)}, \boldsymbol{a}_c^{(k)}]), \quad \forall c \in C$
6: $\quad \boldsymbol{a}_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \phi_V^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{h}_u^{(k-1)}), \quad \forall v \in V$
7: $\quad \boldsymbol{h}_v^{(k)} = \psi_V^{(k)}([\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}]), \quad \forall v \in V$
8: **end for**
9: $y_v = out(\boldsymbol{h}_v^{(K)}), \quad \forall v \in V$

(Step 9). A weighted binary cross entropy loss is used to evaluate the performance of the model, where the weights are used to deal with the imbalance between the two classes. Indeed, about 90% of the columns belong to the unselected class, i.e., their label $y_v = 0$.

Column set V includes the newly generated columns (i.e., those in $\mathcal{G}_\ell$) and also the columns in the current basis of the RMP. The latter columns can give an indication of which columns to select and they have proven to be valuable. Taking all the columns of the RMP (i.e., the non-basic columns as well) can result in a slightly better accuracy of the model. However, the graphs would be much larger each time that columns are added to the RMP, which would require more computational effort. This would slow down the training and the prediction, and also increase memory usage.

The data is collected by solving multiple problem instances using the MILP (10)-(15) to select the columns and assign the labels. At each CG iteration, a bipartite graph is built and the following data is stored :

— The sets of column and constraint nodes ;
— A sparse matrix $E \in \mathbb{R}^{n \times m}$ storing the edges ;
— A column feature matrix $X^V \in \mathbb{R}^{n \times d}$, where n is the number of columns and d the number of column features ;
— A constraint feature matrix $X^C \in \mathbb{R}^{m \times p}$, where m is the number of constraints and p the number of constraint features ;
— The label vector $\boldsymbol{y}$ of the newly generated columns in $\mathcal{G}_\ell$.

Note that, in a CG algorithm, the number of generated columns n is not constant throughout the iterations. Furthermore, if CG is used to solve various instances of different sizes, the number of constraints m may vary from one instance to another. Nonetheless, the same GNN model can handle graphs with different numbers of nodes and arcs, as long as the numbers of column and constraint features (d and p) are identical.

4.4 Case study I : Vehicle and crew scheduling problem

In this first case study, we focus on a problem arising in urban transit systems called the VCSP. Traditionally, this problem is split into two problems that are solved in a sequential fashion, where the vehicle schedules are first determined (vehicle scheduling problem - VSP) before the crew schedules (crew scheduling problem - CSP). This approach has been strongly criticized [63], because in most cases driver costs dominate the vehicle costs. In our case, we consider the VCSP, which is a complete integration of the CSP and the VSP. The objective is to determine vehicle schedules and driver schedules simultaneously while minimizing the total cost. The first formulation that integrates both problems is that of Freling et *al.* [64], where two approaches for solving the problem, the first using CG and the second using Lagrangian relaxation, are proposed.

In our work, we consider the model of Haase et *al.* [31], which is a set partitioning model with additional constraints involving only the crew scheduling variables. The additional constraints ensure that vehicle schedules are obtained in polynomial time. Moreover, taking into account the vehicle costs in the objective function ensures that an optimal solution is obtained. The authors propose a branch-and-price method to solve the proposed formulation. Other formulations based on CG and set partitioning models were then proposed by Freling et *al.* [65]. An extension for the case of multiple depots (MD-VCSP) using Lagrangian relaxation in combination with CG is proposed by the same authors in [66]. A similar formulation but with less decision variables and fewer constraints is proposed by Mesquita and Paias [67]. More work was done in this direction by de Groot and Huisman [68], where different splitting strategies for tackling large real-world MD-VCSP instances are devised.

4.4.1 Problem description and basic solution approach

We focus on the case of a bus company with a single depot and a homogeneous fleet of vehicles as described in [31]. Each bus line is defined by : a departure point, a destination point, several intermediate stops where passengers can get on and off the bus. Some of these stops are called relief points where driver exchanges can occur. For each line, there is a predefined timetable defining a set of trips to be covered during the day. The objective is to assign the buses to the different trips, as well as assigning the drivers to the buses, while minimizing total costs, which include driver and bus operational costs, and may also include fixed costs for vehicles and drivers.

As mentioned above, our basic solution approach is that of Haase et *al.* [31] who developed a branch-and-price algorithm, where the master problem is a set partitioning model with additional constraints, involving driver schedule variables only. The PP is a shortest path problem with resource constraints solved with a labeling algorithm [18] that allows the gene-

ration of driver schedules, also called duties. Note that the number of columns generated by this algorithm can be adjusted by modifying the dominance rule or by solving the same PP multiple times after removing some nodes/arcs from the network after each resolution. The detailed network structure and formulation are described in [31].

4.4.2 Features considered

As mentioned in Section 4.3.1, the features collected depend on the problem at hand. For the VCSP case, each column represents a path with resource constraints, and the following features are used :

- **Columns :** cost, reduced cost, total number of constraints that the column contributes to, number of constraints per constraint group that the column contributes to (the set partitioning model of Haase et *al.* [31] has four groups of constraints), resource values (i.e., work time, duty duration), duty type (there are two possible duty types), columnIsNew (boolean value indicating if the column is newly generated or in the basis), column value (if the column is in the basis, 0 otherwise), column incompatibility degree (as defined in [12]).
- **Constraints :** dual value, number of columns (newly generated or in the RMP basis) contributing to the constraint (equal to the degree of the node in the bipartite graph).

4.4.3 VCSP instances

Instances of the VCSP were randomly generated as in [31]. The trips to cover during a day are uniformly distributed over the time horizon, with more trips during peak hours. A total of 100 instances of 400 trips were generated for the training phase. These instances were solved using the MILP for the column selection at each iteration and the labels were assigned accordingly. The data of the 100 instances were collected resulting in more than 7000 iteration datasets (i.e., 7000 bipartite graphs). Once the training was completed, new instances of different sizes varying between 300 and 800 trips were solved using the learned model for selecting the columns. The experimental results of both ML and optimization phases are reported in the next section.

4.4.4 Computational results

We start our computational experiments by evaluating the performance of the MILP proposed in Section 4.3.1 because we are trying to imitate it. Once the performance of the MILP is confirmed, we can proceed to the evaluation of the training phase and, finally, to the integration of the learned model in the CG algorithm.

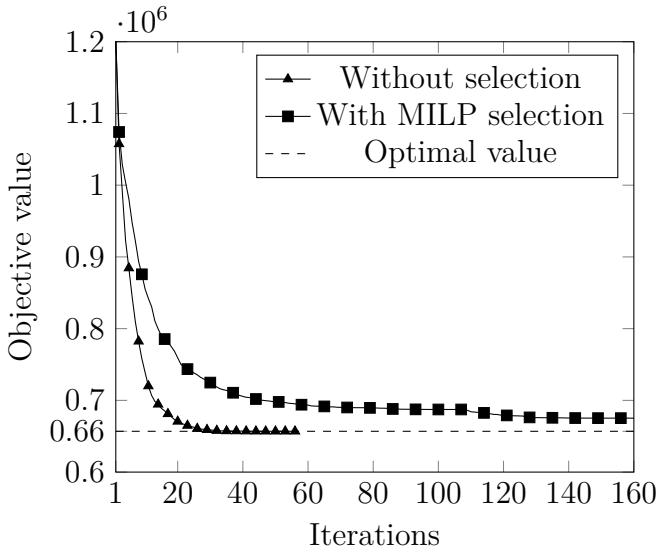


FIGURE 4.5 Comparison between the execution without selection and with MILP selection.

MILP performance

Column selection by the MILP was performed on several VCSP test instances. For a typical instance, Figure 4.5 compares the normal execution, where all generated columns are added (curve with triangles), and the execution with the MILP selection (curve with squares). By adding all the columns, the normal execution reaches the optimal solution in 56 iterations for this instance, unlike the execution with the MILP selection, which exhibits a major convergence issue. This is mainly due to the rejected columns that remain with a negative reduced cost after the RMP reoptimization and keep on being generated in subsequent iterations, even though they do not improve the objective value (degeneracy). Somehow, the columns selected by the MILP are not sufficient and do not allow the generation of significantly better columns in the subsequent iterations. A possible solution would be to add more columns, and the best candidates are the columns that remain with a negative reduced cost after the reoptimization of the RMP (see Figure 4.6). Given the large number of those columns, it is not necessary to add them all. One option consists of sorting them by increasing reduced cost and adding a predetermined percentage of them.

TABLE 4.1 Results with the MILP selection.

Instance	# Constraints	Without Selection				With Selection				Time Reduction
		Time	Time RMP	# Itr	# Col	Time	Time RMP	# Itr	# Col	
VCS_400_1	1740	310	244	104	20,195	174	132	44	15,624	44%
VCS_400_2	1757	311	262	46	17,942	178	140	41	14,754	53%
VCS_400_3	1752	464	405	53	20,140	298	243	57	17,101	36%
VCS_400_4	1757	363	308	50	19,453	284	233	51	16,683	22%
VCS_400_5	1759	319	270	46	18,162	218	175	43	14,352	32%
Average	**1753**	**353**	**297**	**59**	**19,178**	**230**	**184**	**47**	**15,703**	**34%**

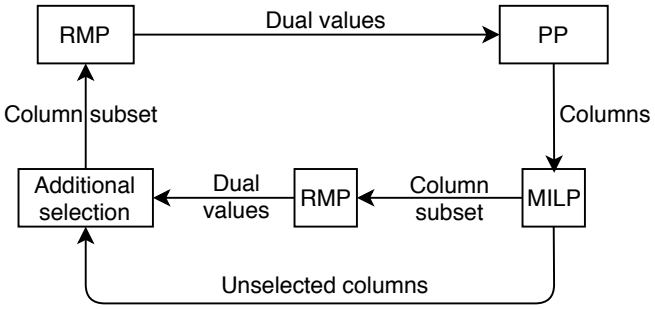


FIGURE 4.6 CG algorithm with MILP column selection and additional columns

For five VCSP instances with 400 trips, Table 4.1 shows the results obtained by the CG algorithm without column selection and the one with column selection by the MILP and additional columns. For each instance, we report : the total computational time (in seconds) required to solve the linear relaxation, the time taken to solve the RMPs only (in seconds), the number of CG iterations, the number of columns in the RMP at the end of the optimization, and the percentage of the computational time saved by using column selection. Preliminary tests were run to configure both algorithms. In particular, the CG algorithm without column selection turns out to be better when the number of columns generated is approximately three times less than the number of columns generated by the algorithm with column selection. For the latter algorithm, we select in addition of the column selected by the MILP 50% of the unselected columns that have a negative reduced cost of the RMP reoptimization. Note that, for the results presented in this section, no columns are removed from the RMP (i.e., $n^{+}_{cols} = \infty$) and the computational time of the algorithm with column selection does not include the time spent solving the MILP at every iteration since we are interested only on assessing the effectiveness of the selection. The MILP will be replaced afterwards with a fast learned model.

From these results, we observe that column selection allows a reduction of the average com-

putational time of 34%, showing the potential of column selection. Given that the average number of iterations is quite similar for the two algorithms (except for the first instance which highly suffered from degeneracy without column selection), we can deduce that this gain is due to a reduction in the time spent solving the RMPs. Indeed, the saving on the average RMP time is around 38%, which results from a smaller average number of columns to handle (around 20% less columns).

Machine learning phase

As always in ML, several hyperparameters have to be adjusted. The best model that we obtained uses the values presented in Table 4.2. The accuracy can be slightly improved in favor of a longer prediction or training time, which is not worth it based on the experiments we conducted.

TABLE 4.2 Parameters used for training the VCSP GNN model.

Parameters	**Value**
Number of iterations	1
Learning rate	10^{-3}
Maximum number of epochs	1000
Epoch size	32
Batch size	16
Intermediate NN architecture	32×32
Output NN architecture	$32 \times 32 \times 1$
Activation function	ReLU
Optimizer	Adam
Sigmoid class weights	10 :1

The data set is split into two sets : training and test sets (75% for training and 25% for test). The training is performed in 1000 epochs, where each epoch includes 32 batches, and each batch includes data from 16 iterations. The number of iterations corresponds to the number of updates of the constraint and column nodes in the graph (parameter K in Algorithm 1). This parameter is set to 1 as a higher value of K resulted in a slight improvement of 1%, but the training and the prediction took at least twice as much time, which is not desirable. The intermediate NN architecture corresponds to a 2-layer neural network with 32 neurons on each layer and ReLU activations. This architecture implements the functions $\phi_C^{(k)}$, $\psi_C^{(k)}$, $\phi_V^{(k)}$, $\psi_V^{(k)}$ in Algorithm 1, whereas the Output NN architecture implements the output function

out. To deal with the imbalanced data, a weighted sigmoid function is used, assigning a weight of 10 and 1 to the columns with label $y_v = 1$ and $y_v = 0$, respectively (i.e., misclassifying one column with label $y_v = 1$ is equivalent to misclassifying 10 columns with label $y_v = 0$). The weights allow also to give more importance to the columns to be selected.

To evaluate the performance of the ML model obtained, we consider different metrics that are listed in Table 4.3 and computed as follows. Let tp, tn, fp, and fn be the number of true positives (columns with label $y_v = 1$ that are predicted correctly), true negatives (columns with label $y_v = 0$ that are predicted correctly), false positives (selected columns that should have been rejected), and false negatives (rejected columns that should have been selected), respectively. Then,

$$\begin{aligned} \text{Recall} &= \frac{tp}{tp + fn} \\ TNR &= \frac{tn}{tn + fp} \\ \text{Precision} &= \frac{tp}{tp + fp} \\ \text{Balanced Accuracy} &= \frac{TPR + TNR}{2}, \end{aligned}$$

where Recall is also called TPR for True Positive Rate and TNR stands for True Negative Rate.

The results reported in Table 4.3 were obtained on a test set consisting of 400-trip instances. From these results, one can see that the Recall value is high, indicating that the columns to select are predicted correctly with an 86.2% accuracy. However the TNR value is only 66.6%, which means that among all the columns that should not be selected, only 66.6% were predicted correctly and the remaining ones are added when they should not. In our case, we are more interested by selecting accurately the columns with label $y_v = 1$ because selecting additional columns is not an issue and may be desirable to avoid convergence issues as encountered with the initial MILP selection. Because of the label imbalance, the 33.4% columns that are misclassified and added by the model largely exceed in numbers the 86.2% correctly predicted ones, which is reflected by the Precision value of 23.7%. In other words, out of all the columns selected by the model, only 23.7% should actually be selected and the remaining columns are supplementary. Therefore, no additional columns are required when using this model in the CG algorithm, unlike the MILP where additional columns were needed. Next, note that the use of the normal accuracy ($accuracy = \frac{tp+tn}{tp+tn+fp+fn}$) can be ambiguous when dealing with imbalanced data. Consequently, it is more interesting to consider the Balanced accuracy that normalizes both classes. A Balanced accuracy of 50% is

TABLE 4.3 Metrics of the best GNN model obtained for the VCSP.

Metric	**Value**
Recall	86.2%
TNR	66.6%
Precision	23.7%
Balanced Accuracy	76.5%

obtained when all the columns are given the same label 0 or 1, so our 76.5% Balanced accuracy is a satisfactory result. Finally, to conclude this section, we highlight that the training phase can require a few hours of computing time or even more depending on the performance of the computer used and the usage of GPUs or not.

Optimization phase

Once the training is done, the learned model is integrated into the CG algorithm for selecting columns at each iteration (it replaces the MILP in Figure 4.2). To test the efficiency of the learned model, we compare the following five algorithms that all use the acceleration strategies described in Section 4.2 : i) disjoint blocks to diversify the columns to be added or selected, ii) column removal when the RMP becomes too large, and iii) heuristic labeling for solving the PP before calling the exact labeling algorithm if needed.

No selection (NO-S) : This is the standard CG algorithm with no selection involved, with the use of the acceleration strategies described in Section 4.2.

MILP selection (MILP-S) : The MILP is used to select the columns at each iteration, with 50% additional columns to avoid convergence issues. Because the MILP is considered to be the expert we want to learn from and we are looking to replace it with a fast approximation, the total computational time does not include the time spent solving the MILP.

GNN selection (GNN-S) : The learned model is used to select the columns. At every CG iteration, the column features are extracted, the predictions are obtained, and the selected columns are added to the RMP.

Sorting selection (Sort-S) : The generated columns are sorted by reduced cost in ascending order, and a subset of the columns with the lowest reduced cost are selected. The number of columns selected is on average the same as with the GNN selection.

Random selection (Rand-S) : A subset of the columns is selected randomly. The number of columns selected is on average the same as with the GNN selection.

TABLE 4.4 Parameter values used by the five algorithms.

Algorithm	n^-_{cols}	n^+_{cols}	n^{max}_{blks}	min_{sel}	ϵ (Penalty)	Additional columns
NO-S	$3 \times n_{cons}$	$6 \times n_{cons}$	4	-	-	-
MILP-S			14	100	0.1	50%
GNN-S					-	0%
Sort-S						
Rand-S						

Although the reduced cost of a column is not a reliable indicator of whether a column is interesting or not, comparing the model with the last two selection strategies will help us to determine if the model was really able to identify some hidden patterns and to learn effectively from the data.

The parameter values used by the five algorithms are summarized in Table 4.4. As defined in Section 4.2, the parameters n^-_{cols} and n^+_{cols} correspond to the minimum and maximum number of columns to keep in the RMP, respectively, where n_{cons} is the number of constraints in the RMP. To ensure column diversity, a number n^{max}_{blks} of disjoint blocks is used. For algorithms including a selection strategy, a higher value is used to select from a larger pool of columns. For the MILP, a low penalty $\epsilon = 0.1$ is used for every selected column since the selection with the MILP alone suffers from convergence issues and a higher penalty implies less selected columns. In addition to the low penalty, 50% of the remaining columns with the most negative reduced cost (after the RMP reoptimization) are added to the RMP. The parameter min_{sel} is equal to the minimum number of columns that need to be generated to activate column selection at a CG iteration. In fact, in preliminary tests, we observed that it is not worthwhile selecting columns from a small set of generated columns. As shown in the ML results, the GNN model selects more columns than the MILP (Recall of 23.7%) and therefore no additional columns are needed. Note that some parameters were tuned by group of algorithms to obtain the best results for each algorithm and to provide a fair comparison between them.

The results obtained for 15 VCSP instances involving 300 to 500 trips are reported in Table 4.5. For each algorithm, the total time in seconds, the time spent solving the RMP and the PPs, as well as the number of CG iterations are reported. The time reduction column compares the GNN-S to the NO-S algorithm. First, let us mention that the time taken by the GNN model to make the prediction in GNN-S is negligible : it varies between $40ms$ and $70ms$ per iteration, depending on the size of the instance and the number of generated columns. Therefore, given the reduced number of iterations performed, the total prediction time sums

up to a few seconds only. By excluding the time taken to solve the MILPs in the algorithm MILP-S, we are assuming that the MILP requires a negligible time just like the GNN prediction time, which allows us to fairly compare the two selection techniques and to check how close GNN is to the MILP selection. Obviously, solving the MILP is too time-consuming to be used in practice. In fact, its solution time is larger than the time required to solve the RMP.

The computational times reported for both GNN-S and MILP-S (the latter excluding the MILP solution time) are very close to each other for most instances, showing that the GNN does a good job at imitating the MILP selection. Still, there is a difference between the two, namely, the GNN model tends to select more columns than the MILP. Therefore, slightly different results are obtained, sometimes better and sometimes worse but in average very close. The times of the last two algorithms based on the other two selection strategies (Sort-S and Rand-S) are either close or worse than those obtained by NO-S ; they are far from the results given by GNN-S or MILP-S. Their relatively bad performance is also reflected by the larger number of iterations performed. The selection with the GNN model gives good results on all the instances even if the training has only been done on instances with 400 trips, which shows the ability of the model to generalize to instances of different size. Additionally, the total number of columns added to the RMP before reaching the optimal solution is generally lower than the case without selection. Compared to NO-S, GNN-S gives on average a reduction ranging from 25% to 30% as shown in the last column where most of the reductions are obtained on the RMP side.

Additional experiments have been conducted on larger instances with 600 to 800 trips, comparing only the first three algorithms. The detailed results are shown in Table 4.6. They indicate that column selection using the GNN model is still practical and gives good results, namely, average time reductions of 21 to 23%. Note that, for these tests, we used the same prediction model as for the smaller instances, which was trained on instances with 400 trips.

As mentioned in Section 4.1.1, the CG algorithm is well known to have convergence issues due to the degeneracy and dual variable instability. To overcome these difficulties, various stabilization techniques have been devised and proven to be very effective on different problems [9]. To test whether the improvements obtained by our column selection strategy are preserved for a stabilized CG algorithm, we implemented a variant of the stabilized algorithm of Pessoa et *al.* [9] that is based on the Wentges rule

$$\tilde{\pi}^t = \alpha\hat{\pi} + (1-\alpha)\pi^t, \tag{4.11}$$

where α is a parameter in $(0, 1]$ (set to 0.6 for our tests), $\tilde{\pi}^t$ is the vector of dual variables

TABLE 4.5 Results for VCSP instances with 300 to 500 trips.

Instance	NO-S				MILP-S				GNN-S				Sort-S				Rand-S				Time reduction
	Time (s)			#Itr	Time (s)			#Itr	Time (s)			#Itr	Time (s)			#Itr	Time (s)			#Itr	
	Total	RMP	PP		Total	RMP	PP		Total	RMP	PP		Total	RMP	PP		Total	RMP	PP		
VCS_300_1	105	66	39	63	73	41	32	62	69	41	28	54	93	52	41	74	110	63	47	81	**34%**
VCS_300_2	72	48	24	41	49	31	18	40	49	33	16	37	71	44	27	55	73	46	27	52	**32%**
VCS_300_3	131	80	51	83	115	59	56	94	95	57	38	76	156	78	78	151	150	83	67	118	**27%**
VCS_300_4	158	104	54	73	107	55	52	76	110	57	53	75	151	84	67	118	175	97	78	123	**30%**
VCS_300_5	76	47	29	58	54	31	23	49	55	34	21	45	79	48	31	61	86	54	32	60	**28%**
Average	**108**	**69**	**39**	**63**	**79**	**43**	**36**	**64**	**75**	**44**	**31**	**57**	**110**	**61**	**49**	**91**	**118**	**68**	**50**	**86**	**30%**
VCS_400_1	265	178	87	74	170	113	57	56	180	96	84	75	287	174	113	108	270	170	100	95	**32%**
VCS_400_2	259	184	75	72	187	129	58	58	158	106	52	55	305	192	113	112	227	154	73	71	**39%**
VCS_400_3	365	260	105	93	277	191	86	89	278	171	107	96	435	256	179	186	427	274	153	140	**24%**
VCS_400_4	315	222	93	77	254	173	81	73	221	138	83	70	391	246	145	137	370	243	127	109	**30%**
VCS_400_5	234	168	66	59	199	137	62	60	207	138	69	61	319	207	112	110	277	188	89	79	**12%**
Average	**287**	**202**	**85**	**75**	**217**	**148**	**68**	**67**	**208**	**129**	**79**	**71**	**347**	**215**	**132**	**130**	**314**	**205**	**108**	**98**	**25%**
VCS_500_1	721	551	170	79	449	281	168	75	464	295	169	78	781	508	273	150	781	561	220	108	**36%**
VCS_500_2	769	596	173	79	490	321	169	79	455	298	157	72	766	522	244	144	779	574	205	108	**41%**
VCS_500_3	910	679	231	113	510	332	178	88	490	326	164	83	1105	707	398	229	881	622	259	131	**46%**
VCS_500_4	572	416	156	77	485	327	158	87	496	328	168	90	616	430	186	102	583	390	193	105	**13%**
VCS_500_5	818	641	177	92	645	474	171	94	703	512	191	97	794	609	185	107	926	660	266	150	**14%**
Average	**758**	**576**	**181**	**88**	**515**	**347**	**168**	**84**	**521**	**351**	**170**	**84**	**812**	**555**	**257**	**146**	**790**	**561**	**228**	**120**	**26%**

TABLE 4.6 Results for VCSP instances with 600 to 800 trips.

Instance	NO-S				MILP-S				GNN-S				Time reduction
	Time (s)			#Itr	Time (s)			#Itr	Time (s)			#Itr	
	Total	RMP	PP		Total	RMP	PP		Total	RMP	PP		
VCS_600_1	1319	1012	307	108	1049	751	298	100	998	713	285	96	**24%**
VCS_600_2	1445	1098	347	112	1005	699	306	92	1031	709	322	99	**29%**
VCS_600_3	983	740	243	89	659	466	193	75	805	555	250	93	**18%**
VCS_600_4	1930	1434	496	159	1659	1172	487	156	1570	1054	516	165	**19%**
VCS_600_5	1379	977	402	137	1028	708	320	108	1025	674	351	115	**26%**
Average	**1411**	**1052**	**359**	**121**	**1080**	**759**	**320**	**106**	**1085**	**741**	**344**	**115**	**23%**
VCS_700_1	2396	1944	452	101	1977	1513	464	107	1882	1444	438	97	**21%**
VCS_700_2	1613	1285	328	80	1118	836	282	73	1212	901	311	78	**25%**
VCS_700_3	2074	1665	409	106	1470	1090	380	89	1546	1151	395	97	**25%**
VCS_700_4	2465	1982	483	102	1854	1377	477	104	2061	1563	498	109	**16%**
VCS_700_5	2101	1730	371	85	1487	1118	369	87	1672	1278	394	101	**20%**
Average	**2129**	**1721**	**408**	**94**	**1581**	**1186**	**395**	**92**	**1674**	**1267**	**407**	**96**	**21%**
VCS_800_1	4282	3525	757	126	3107	2523	584	109	3614	2848	766	135	**16%**
VCS_800_2	4655	3755	900	145	3205	2550	655	114	3461	2685	776	133	**26%**
VCS_800_3	4998	4036	962	163	3065	2484	581	106	3929	3046	883	155	**21%**
VCS_800_4	8561	6262	2299	382	7333	5809	1524	274	6983	4994	1989	343	**18%**
VCS_800_5	10156	6944	3212	549	5480	4329	1151	195	6676	4731	1945	330	**34%**
Average	**6530**	**4904**	**1626**	**273**	**4438**	**3539**	**800**	**159**	**4932**	**3660**	**1272**	**219**	**23%**

TABLE 4.7 Stabilized algorithms' results for VCSP instances with 300 to 500 trips.

Instance	Stab-NO-S				Stab-GNN-S				Time reduction
	Time (s)			#Itr	Time (s)			#Itr	
	Total	RMP	PP		Total	RMP	PP		
VCS_300_1	65	37	28	47	43	26	17	35	**33%**
VCS_300_2	50	29	21	38	37	22	15	32	**26%**
VCS_300_3	70	40	30	52	58	33	25	48	**17%**
VCS_300_4	94	56	38	56	74	45	29	50	**21%**
VCS_300_5	55	33	22	40	38	22	16	34	**30%**
Average	**67**	**39**	**28**	**46**	**50**	**30**	**20**	**40**	**25%**
VCS_400_1	156	100	56	51	131	84	47	49	**16%**
VCS_400_2	161	106	55	50	107	69	38	41	**33%**
VCS_400_3	204	140	64	54	144	95	49	50	**29%**
VCS_400_4	211	142	69	54	167	111	56	52	**20%**
VCS_400_5	189	132	57	49	142	96	46	42	**24%**
Average	**184**	**124**	**60**	**51**	**138**	**91**	**47**	**47**	**24%**
VCS_500_1	381	274	107	56	330	224	106	57	**13%**
VCS_500_2	471	354	117	56	320	212	108	53	**32%**
VCS_500_3	493	364	129	66	317	221	96	53	**35%**
VCS_500_4	477	337	140	63	333	208	125	59	**30%**
VCS_500_5	571	453	118	58	379	275	104	56	**33%**
Average	**478**	**356**	**122**	**60**	**335**	**228**	**107**	**55**	**27%**

used by the PPs at iteration t, π^t is the dual solution returned by the RMP at this iteration, and $\hat{\pi}$ is the dual solution that allowed to find, in the last five iterations, the largest least reduced cost obtained in an iteration. This algorithm differs from that of [9] only by how $\hat{\pi}$ is computed. In their algorithm, $\hat{\pi}$ corresponds to the dual solution yielding, over all previous iterations, the largest least reduced cost in an iteration, i.e., the best Lagrangian bound found so far. Because columns are mostly generated by heuristic labeling, we do not have access to a Lagrangian bound in most iterations, making it difficult to apply the exact same technique. We integrated this stabilization procedure to the NO-S and GNN-S algorithms, to yield two stabilized algorithms denoted Stab-NO-S and Stab-GNN-S. The results obtained on the same 300- to 500-trip VCSP instances as above are reported in Table 4.7, where for both algorithms, we indicate the same information as in the previous tables. The results show that stabilization can improve the performance of the non-stabilized algorithms by approximately 37% for NO-S and 35% for GNN-S. Therefore, integrating column selection into a stabilized CG algorithm still yields average computational time reductions ranging between 24% and 27%, while these reductions range between 25% and 30% without stabilization (see Table 4.5). As above, most of the reductions come from the RMP side.

4.5 Case study II : Vehicle routing problem with time windows

The vehicle route problem (VRP) has been the subject of intensive research for over fifty years because of its importance in transportation planning and its great difficulty to solve. The VRP consists in planning least-cost delivery routes in order to serve a geographically dispersed set of customers, while respecting the capacity of the vehicles used. In this paper, we focus on the VRPTW, which is a generalization of the VRP where, for each customer, a time window in which the delivery must be made is imposed. To date, the best known algorithms for solving the problem are based on CG where the PP generates vehicle routes. This PP is modeled as an elementary shortest path problem with resource constraints (ESPPRC), where the resource constraints enforce the respect of the time windows, the vehicle capacity, and elementarity requirements. The ESPPRC is usually solved by a labeling algorithm (see, e.g., [18]).

The first CG algorithm for the VRPTW was introduced by Desrochers et *al.* [69]. They obtained their best computational results by considering a relaxation of the PP that allows multiple visits to a customer in a route but avoids 2-cycles. This relaxation however yields weak lower bounds. Later, Feillet et *al.* [21] developed a labeling algorithm for solving the ESPPRC and a CG algorithm where only elementary routes are generated, yielding strong lower bounds (see, [26]). Because the ESSPRC is NP-hard, it is highly difficult to solve it as a PP for instances where a large number of customers can be visited in a route. Therefore, other PP relaxations have been investigated. In particular, Baldacci et *al.* [24] proposed the ng-route relaxation which allows the generation of routes with cycles but prohibits certain cycles by defining a neighborhood for each customer. The neighborhood size can be adjusted to define a compromise between the complexity of solving the PP and the quality of the lower bound achieved. In a different research stream, several valid inequalities specific to the VRPTW have been proposed to tighten the lower bound like the 2-path cuts [70], and the subset-row cuts [71]. Further enhancements were presented in [29]. For a broader view of the various techniques devised for the VRP and its variants, the readers can consult the surveys of Desaulniers et *al.* [72] and Costa et *al.* [27].

4.5.1 Problem description and basic solution approach

Given a fleet of vehicles assigned to a single depot, the VRPTW consists in determining a set of possible routes (one route per vehicle used) to deliver goods to a geographically dispersed set of customers while minimizing the total cost, which is most often proportional to the distance traveled. Each customer must be visited exactly once by a vehicle during a specific time window. If the vehicle visits the customer earlier, the vehicle has to wait until the customer is available in order to make the delivery. A route starts from the depot and visits

a sequence of customers before returning to the depot and it is feasible if the total amount of goods delivered does not exceed the capacity of the vehicle and if the time windows of the visited customers are respected.
In this work, we model the VRPTW as a set-partitioning problem like in most recent papers (see, e.g., [29]) and solve its linear relaxation by CG where the PP is a shortest path problem with resource constraints and 2-cycle elimination as in [69]. The PP is solved by a labeling algorithm that can generate multiple columns as for the VCSP.

4.5.2 Features Considered

The features considered for the VRPTW are quite similar to those of the VCSP since both problems are formulated as set-partitioning problems where every column represents a path with resource constraints. The features are :

- **Columns :** cost, reduced cost, number of constraints that the column contributes to (i.e, number of customers visited by the route), resource values (i.e., time and load), columnIsNew (boolean value indicating if the column is newly generated or in the basis), column value (if the column is in the basis or 0 otherwise).
- **Constraints :** dual value, number of columns (newly generated or in the RMP basis) contributing to the constraint (i.e, number of routes that can visit the customer).

4.5.3 VRPTW instances

The column selection strategy proposed in this paper aims at reducing degeneracy in the MP and, thus, reducing the time spent solving the RMPs. Thus, it can have a significant impact on the overall computational time if the RMPs take a relatively large proportion of it. Because the known VRPTW benchmark instances typically require more time for solving the PP than the RMP, we have decided to modify some of the Gehring & Homberger instances [73] with the aim of creating instances where the PP becomes relatively easy to solve. To do so, the widths of the largest time windows, namely, those with a width exceeding a given threshold, were reduced so that they became shorter than this threshold. The chosen threshold varied from one instance to another between 90 and 120. For our tests, we retained only the instances that spent between 70 and 90% of the total time solving the RMPs. These instances are all derived from the R2 class (the customers are randomly located in a square) and involve 400, 600 or 800 customers.
To perform training, data was collected from only 20 VRPTW instances with 600 customers. This was sufficient given that much more CG iterations (a total of around 10,000 for these 20 instances) were needed to find an optimal solution than for the VCSP. The MILP was again used for column selection and label assignment. To avoid convergence issues, all the

remaining columns with negative reduced cost were added to the RMP since their number is not as high as in the VCSP where only 50% of them were added. Additionally, the iterations with less than 150 generated columns were ignored. Once the data generation was completed, the training was performed by splitting the datasets into training and test sets.

4.5.4 Computational results

As for the VCSP, this section is divided in two : a ML subsection to present the training results, followed by the results of the integration of the learned model in the CG algorithm.

Machine learning phase

The hyperparameter values are presented in Table 4.8. The values are quite similar to the ones used for the VCSP. The only differences are : i) a larger batch size because the bipartite graphs are smaller due to the reduced number of constraints, ii) a larger weight assigned to the columns to select (15 versus 10) because the columns with label $y_v = 1$ represent only 7% of all the columns.

Table 4.8 Parameters used for training the VRPTW GNN model.

Parameters	**Value**
Number of iterations	1
Learning rate	10^{-3}
Maximum number of epochs	1000
Epoch size	32
Batch size	24
Intermediate NN architecture	32×32
Output NN architecture	$32 \times 32 \times 1$
Activation function	ReLU
Optimizer	Adam
Sigmoid class weights	15 :1

We use the same metrics as in Section 4.4.4 to evaluate the performance of the GNN model. From the results reported in Table 4.9, we can see that the columns with label $y_v = 1$ are correctly predicted with 79.3% accuracy. However, the accuracy of the columns with label 0 is only 68.2%, and given their large number due to the imbalance between the two classes, the columns that actually should be selected ($y_v = 1$) represent only 21.8% of the total columns

TABLE 4.9 Metrics of the best GNN model obtained for the VRPTW.

Metric	**Value**
Recall	79.3%
TNR	68.2%
Precision	21.8%
Balanced Accuracy	73.8%

selected by the model. Compared to the results obtained for the VCSP, these results are only slightly different. Thus, with a minimum of tuning, it was possible to obtain a learned model with a very close performance, even though the VRPTW and the VCSP are two completely different problems from a combinatorial optimization perspective.

Optimization phase

To evaluate the performance of the GNN model, VRPTW instances with 400, 600 and 800 customers were generated. The results on instances with 400 and 800 customers allow us to test the model's ability to generalize to instances of different sizes. To do so, we compare the following three algorithms that apply the acceleration strategies described in Section 4.2 :

No selection (NO-S) : This is the standard CG algorithm with no column selection involved. All the generated columns are added to the RMP.

GNN selection (GNN-S) : The learned model is used to select the columns to add to the RMP. At each iteration, we generate more columns compared to NO-S from which we select the columns to add to the RMP. The number of columns considered is controlled by the number of disjoint blocks.

Random selection (Rand-S) : A subset of the generated columns is randomly selected. The number of columns selected is on average the same as with GNN-S.

The values of the main parameters used by these algorithms are provided in Table 4.10. Recall that n^{-}_{cols} and n^{+}_{cols} are the minimum and maximum numbers of columns to keep in the RMP, respectively, n_{cons} is the number of constraints in the RMP, n^{max}_{blks} is the number of disjoint blocks used to ensure column diversity, and min_{sel} is the minimum number of generated columns to activate column selection.

TABLE 4.10 Parameter values used by the three algorithms.

Algorithm	n^{-}_{vols}	n^{+}_{cols}	n^{max}_{blks}	min_{sel}
NO-S	$20 \times n_{cons}$	$40 \times n_{cons}$	35	-
GNN-S	$10 \times n_{cons}$	$20 \times n_{cons}$	50	150
Rand-S				

For each tested instance, Table 4.11 reports the total time in seconds, the time spent solving the RMP and the PP, as well as the number of iterations required by the three algorithms for solving the linear relaxation. The last column corresponds to the time reduction when comparing GNN-S with NO-S. One can see that the column selection with the GNN model gives positive results, yielding average reductions ranging from 20% to 29%. These reductions could have been larger if the number of CG iterations performed had not increased. Note also that for the VRPTW, the total time needed to make the predictions is larger than for the VCSP because the total number of iterations is significantly larger. The total prediction time can vary from 50 s to 300 s depending on the number of iterations and the instance size, which can explain the limited effectiveness of column selection for the largest tested instances. However, despite this, the selection with the GNN model is still more effective than a completely random selection, even if the latter does not require much computational effort.

4.6 Conclusion

In this paper, we have presented a new approach for accelerating CG. The approach, based on ML, is more suitable to problems that take more time to solve the RMP than the PP as it focuses on reducing the RMP computational time. The objective is to select the most promising columns at each iteration using a supervised learning algorithm trained on data collected from previous executions. We started by defining an effective but expensive column selection strategy with a MILP. Then we replaced such expensive computation with a fast prediction. The learning problem was brought down to a node classification problem on a graph using a GNN. The advantage of using this approach is that, to determine whether a column should be selected or not, the information from the other considered columns is used. Computational results on two well-known problems from the literature, the VCSP and the VRPTW, have indicated average reductions of 20 to 30% in the total computational time required to solve the linear relaxation. These results also showed the ability of the GNN model

TABLE 4.11 Results for the VRPTW instances.

Instance	NO-S				GNN-S				Rand-S				Time reduction
	Time (s)			# Itr	Time (s)			# Itr	Time (s)			# Itr	
	Total	RMP	PP		Total	RMP	PP		Total	RMP	PP		
400_1	349	245	103	668	244	141	103	790	342	198	144	829	**30%**
400_2	234	177	56	352	162	101	60	384	204	132	72	436	**30%**
400_3	310	225	85	572	219	146	73	612	276	175	101	632	**29%**
400_4	169	121	48	292	116	71	45	333	189	120	69	395	**31%**
400_5	260	195	64	393	201	132	68	509	273	175	97	560	**23%**
400_6	390	278	111	817	308	177	130	762	360	213	145	912	**21%**
400_7	283	207	75	467	181	90	90	557	250	143	106	653	**36%**
400_8	265	227	38	602	177	139	38	678	237	187	50	679	**33%**
400_9	190	155	35	366	94	68	21	305	157	116	40	321	**50%**
400_10	270	168	102	754	211	118	93	665	287	145	142	792	**22%**
Average	**272**	**200**	**72**	**528**	**191**	**118**	**73**	**559**	**267**	**160**	**97**	**620**	**29%**
600_1	1066	898	168	910	712	455	257	1178	1093	821	272	1226	**33%**
600_2	630	530	100	625	461	323	137	758	541	408	132	765	**27%**
600_3	1164	1003	161	929	935	678	256	1321	1433	1103	330	1385	**20%**
600_4	1187	739	448	378	803	532	270	820	1097	792	305	792	**32%**
600_5	1800	1614	185	1130	1138	864	273	1475	1289	1030	259	1498	**37%**
600_6	1105	986	119	768	725	552	173	791	884	678	206	772	**34%**
600_7	1439	1244	195	980	1268	920	348	1194	1585	1230	355	1250	**12%**
600_8	2116	1812	304	1007	1867	1270	597	2211	1965	1500	465	1277	**12%**
600_9	1158	920	238	960	1029	620	409	1054	1117	626	491	1310	**11%**
600_10	1690	1547	143	1191	1374	1182	192	1536	1617	1366	251	1698	**19%**
600_11	754	637	117	746	597	443	154	680	726	570	156	827	**21%**
600_12	1503	1189	314	1062	1265	821	444	1300	1627	1151	476	1335	**16%**
600_13	1957	1770	187	1469	1338	1119	219	1476	1873	1579	294	1982	**32%**
600_14	821	687	134	739	600	442	158	842	825	594	231	1077	**27%**
Average	**1314**	**1113**	**201**	**921**	**1008**	**730**	**278**	**1188**	**1262**	**960**	**302**	**1228**	**23%**
800_1	4195	3726	469	1258	3307	2534	773	1630	4210	3531	679	1748	**21%**
800_2	4002	3318	684	1404	3349	2522	827	1833	3528	2732	796	1873	**20%**
800_3	4228	3550	678	1563	3221	2322	899	1843	3481	2368	1113	2204	**24%**
800_4	3182	2658	524	1572	2461	1750	711	1788	2968	2125	843	2008	**23%**
800_5	4594	4097	497	1347	3925	3229	696	2410	4625	3765	860	2570	**15%**
Average	**4040**	**3470**	**570**	**1159**	**3253**	**2471**	**782**	**1900**	**3762**	**2904**	**858**	**2080**	**20%**

to generalize to instances of sizes different than those used in training. Still, improvements to the combined ML-CG algorithm can be made, either in the choice of features or the learning algorithm used, or the method applied to identify the promising columns (maybe by combining information from more than one “expert" instead of the MILP only).

Acknowledgement : We thank Giro Inc. and the Natural Sciences and Engineering Research Council of Canada for their financial support through the grant CRDPJ 520349-17.

CHAPITRE 5 ARTICLE 2: MACHINE-LEARNING-BASED ARC SELECTION FOR CONSTRAINED SHORTEST PATH PROBLEMS IN COLUMN GENERATION

Cet article a été publié en ligne comme article en avance dans la revue *INFORMS Journal on Optimization* le 11 Octobre 2022.

Référence : M. Morabit, G. Desaulniers, A. Lodi (2022), Machine-Learning–Based Arc Selection for Constrained Shortest Path Problems in Column Generation. *INFORMS Journal on Optimization* 0(0).

Abstract

Column generation is an iterative method used to solve a variety of optimization problems. It decomposes the problem into two parts : a master problem, and one or more pricing problems (PP). The total computing time taken by the method is divided between these two parts. In routing or scheduling applications, the problems are mostly defined on a network, and the PP is usually an NP-hard shortest path problem with resource constraints. In this work, we propose a new heuristic pricing algorithm based on machine learning. By taking advantage of the data collected during previous executions, the objective is to reduce the size of the network and accelerate the PP, keeping only the arcs that have a high chance to be part of the linear relaxation solution. The method has been applied to two specific problems : the vehicle and crew scheduling problem in public transit and the vehicle routing problem with time windows. Reductions in computational time of up to 40% can be obtained.

Keywords : Column generation, machine learning, arc selection, heuristic pricing, vehicle routing and crew scheduling.

5.1 Introduction

Column generation (CG, [1]) is an iterative method used to solve the linear relaxation of optimization models involving a large number of variables. It splits the problem into two parts. The first part is a restricted master problem (RMP) that corresponds to the original linear program (called the master problem) but restricted to a subset of its variables. The RMP is solved at each iteration, often using the Simplex method, to yield a primal and a dual solution. The second part is the subproblem, also called the pricing problem (PP), that is also solved at each iteration. Its role consists in generating columns of negative reduced cost (in case of a minimization problem) that can improve the current RMP solution. The method

stops when no more columns can be generated. Otherwise, the new columns are added to the RMP to start a new iteration. To obtain integer solutions, CG is often embedded within a branch-and-bound process, where CG is used to solve the linear relaxation at each node of the search tree. In this case the method is referred to as branch-and-price, see [1, 4].

CG has been successfully used to solve a variety of optimization problems, notably crew scheduling and vehicle routing problems arising in freight, urban and airline transportation. For most of these problems, the master problem corresponds to a set partitioning problem with side constraints, and the PP is a shortest path problem with resource constraints (SPPRC) or one of its variants. In these applications, the PP is most likely the part that consumes the largest portion of the computing time and heuristics are often used to solve it.

The purpose of this paper is to propose a new heuristic pricing algorithm based on machine learning. This algorithm takes advantage of previously solved problem instances in order to speed up the solution process of future ones. This is achieved by learning a classifier that builds a reduced network by selecting the arcs with a higher chance to be a part of an optimal master problem solution or at least be used during CG. A good reduction of the network size decreases the computing time of the PP without increasing significantly the number of CG iterations, thus reducing the overall computing time. To ensure the exactness of the CG algorithm, the complete network is used when the reduced one fails to generate new columns. The developed method is considered general enough to be adapted to a variety of optimization problems involving an SPPRC PP, where a reduction in the size of the underlying network can save computing time.

The success of using CG for scheduling and routing problems is in large part due to the efficient methods for solving the SPPRC, which are mainly dynamic programming algorithms [18, 74]. It is well known that the SPPRC is NP-hard, but the methods developed to solve it are quite efficient (i.e., pseudopolynomial time complexity). The dynamic programming algorithms are also flexible enough to tackle different variants of the problems and take into account complex constraints that influence the feasibility of a route or a schedule. An improvement over the basic SPPRC algorithm is that of Righini and Salani [20] who proposed a bi-directional search that propagates labels in both directions at the same time, i.e., from the source node to the destination, and backward from the destination to the source, before merging forward and backward labels to possibly form complete feasible paths.

The standard SPPRC often arises in problems that are defined on acyclic time-space networks, e.g., in vehicle and crew scheduling problems [31, 75] and rostering problems [76]. Another well-known variant that is more difficult to solve is the elementary SPPRC (ESPPRC) occurring in vehicle routing problems (VRP) [21,26,27]. The use of an exact method to solve the ESPPRC is known to yield a tight lower bound, but given the difficulty of the pro-

blem, other alternatives based on relaxations have been explored ; see, e.g., SPPRC-k-cyc [22] and ng-routes [24].

The goal of these approaches is to find a trade-off between the quality of the bound and the difficulty of solving the PP. Several SPPRC heuristics have also been proposed. For instance, some authors relax the dominance rule when solving the PP with a labeling algorithm by only dominating on a subset of the resources [23]. Another option is to keep only a limited number of labels at each node [33]. Other strategies are based on reducing the size of the network. One consists of sorting the incoming and outgoing arcs of each node by their reduced cost [23] and retaining only a subset of the arcs with the least costs. Alternatively, Fukasawa et *al.* [33] apply an extension of Kruskal's algorithm to build disjoint spanning trees from the network, and consider only the arcs used in those trees in addition to the depot outgoing and incoming arcs. All of the above strategies can be used in the same execution. In most cases, an exact algorithm is invoked at the last iteration to prove the solution optimality.

Combining machine learning methods with operations research and combinatorial optimization algorithms [43] has been the subject of many studies over the last few years. These studies generally fall into two categories. The first category is based on what is called *imitation learning*, where the learner tries to imitate an expert in order to get a fast prediction to make decisions that are otherwise computationally expensive. In this category one seeks to reproduce decisions that are as close as possible to those of the expert. The presence of data from previous executions of the expert is therefore necessary. The second category is *reinforcement learning*, where the learner (i.e., agent) interacts with the environment and tries to optimize the results (i.e., maximize the rewards) of the decisions it makes. In this context, the agent learns by trial and error while optimizing its reward function. After a number of iterations, a policy is learned telling the agent what is the best decision to make in each situation. The approach described in this article falls in the first category, that is learning by imitation and is thus based on supervised learning.

Several papers (e.g., [49–51]) have focused on the branch-and-bound algorithm in mixed integer programming, many of which seek to imitate *strong branching*. Several other methods exist as covered in the survey of Lodi and Zarpellon [62]. In the context of CG and branch-and-price, the following two papers are worth mentioning. Václavík et *al.* [45] proposed a regression model for estimating an upper bound on the optimal value of the PP, which is then used to speed up the PP solution process. In this case, the learning is performed at the same time as the optimization. More recently, Morabit et *al.* [77] presented a learned column selector that selects, from a large set of generated columns, the columns to be added to the RMP at each CG iteration. The goal is to add the most promising columns that have a high probability of improving the current solution. The model used is based on graph

neural networks, i.e., neural networks applied to graph-structured data. Differently from the current paper, the strategy in [77] is designed for problems that take a larger portion of the computing time in solving the RMP.

As previously mentioned, the approach developed in this paper can be applied to different optimization problems involving a SPPRC or one of its variants. For our computational experiments, we chose to demonstrate its effectiveness on two well-known problems in the literature : the vehicle and crew scheduling problem (VCSP) and the vehicle routing problem with time windows (VRPTW). Each of these two problems have a different network structure (i.e., different types of nodes and arcs, etc), but nothing prevents from restricting the selection strategy to a specific type of arcs if needed.

The remainder of the paper is organized as follows. Section 2 is devoted to some preliminaries and details about CG and the dynamic programming method used to solve the SPPRC. In Section 3, we describe the proposed selection strategy. Sections 4 and 5 will cover our two case studies, the VCSP and VRPTW respectively, highlighting the implementation differences between the two problems and reporting the computational results for each of them. Finally, we draw some conclusions in Section 6.

5.2 Preliminaries

In the CG context, and by focusing on the linear relaxation of the problem, let us consider the following master problem (MP) formulated as follows :

$$z^*_{MP} := \min_{x} \quad \sum_{p\in\Omega} c_p x_p \tag{5.1}$$

$$\text{s.t.} \quad \sum_{p\in\Omega} \mathbf{a_p} x_p = \mathbf{b}, \tag{5.2}$$

$$x_p \geq 0, \quad \forall p \in \Omega, \tag{5.3}$$

where Ω represents the set of variable indices (e.g., feasible routes or schedules), $c_p \in \mathbb{R}$ the variable cost, $\mathbf{a_p} \in \mathbb{R}^m$ the constraints coefficients and $\mathbf{b} \in \mathbb{R}^m$ the constraints right-hand side vector. When $|\Omega|$ is large and the variables cannot be enumerated explicitly, we consider only a subset $\mathcal{F} \subseteq \Omega$ of the variables, obtaining a restricted version of the problem above called a restricted master problem, RMP. At each CG iteration, the RMP is optimized, and an optimal solution x is obtained along with a dual solution $\pi \in \mathbb{R}^m$ associated with the constraints (5.2). The dual values π are then used to define the PP $\min_{p\in\Omega} \{c_p - \pi^\top \mathbf{a_p}\}$ and find new columns with negative reduced cost by solving it. The method stops when no such columns exist. In our applications, the variables represent either routes or schedules, and

they can be generated by solving one or more PP (in our case a SPPRC or an ESPPRC).

5.2.1 SPPRC formulation

The SPPRC is a problem defined over a network where the goal is to find a path with the least cost between a source and a sink node, while respecting constraints on a set of resources. In the CG context, paths can represent different entities (e.g., vehicle routes, driver schedules, ...) depending on the problem at hand. Examples of possible resources are the time, the capacity of a vehicle (i.e., the maximum load allowed), the break duration, etc. These resources are, typically, consumed (or accumulated) along the arcs of a path and their consumption (which can be reset/adjusted) must fall within predetermined limits at the visited nodes. For example, in the VRPTW, two resources are considered, namely, time and load. The time resource is used to restrict the start of service time at each visited client to a predetermined time interval (the client's opening hours), while the load resource ensures that the total amount of delivered items does not exceed the vehicle capacity. In this section, we provide a general overview of the SPPRC. Readers are referred to [18] for more details.

Let $G = (V, A)$ be a directed graph with a set of nodes V including the source and destination nodes denoted by s and t respectively, and A the set of arcs. Let R be the set of resources. For each node $i \in V$, we define T_i^r as the quantity of resource $r \in R$ consumed on a partial path from source node s to node i, and resource windows $[\underline{w}_i^r, \bar{w}_i^r]$, $\underline{w}_i^r, \bar{w}_i^r \in \mathbb{R}$ restricting the values that resource $r \in R$ can take on node i. For each arc $(i, j) \in A$, we denote by c_{ij} its cost and by t_{ij}^r the quantity of resource $r \in R$ consumed along this arc. Let X_{ij} be the decision variables of the model, which represent the flow on the arcs $(i, j) \in A$. The SPPRC is then formulated as follows :

$$\min \quad \sum_{(i,j)\in A} c_{ij} X_{ij} \tag{5.4}$$

$$\text{s.t.} \quad \sum_{j\in V} X_{ij} - \sum_{j\in V} X_{ji} = \begin{cases} +1 & \text{if } i = s \\ 0 & \text{if } i \in V\backslash\{s,t\} \\ -1 & \text{if } i = t \end{cases} \tag{5.5}$$

$$X_{ij}(T_i^r + t_{ij}^r - T_j^r) \leq 0, \quad \forall r \in R, \forall (i,j) \in A, \tag{5.6}$$

$$\underline{w}_i^r \leq T_i^r \leq \bar{w}_i^r, \quad \forall r \in R, \forall i \in V, \tag{5.7}$$

$$X_{ij} \in \{0, 1\}, \quad \forall (i,j) \in A, \tag{5.8}$$

where the objective (5.4) minimizes the total cost of the path, the constraints (5.5) ensure the flow conservation along the path, while the constraints (5.6) model the consumption of

each resource $r \in R$ along an arc $(i,j) \in A$. Constraints (5.7) make sure that the resource values respect the resource windows defined at each node and constraints (5.8) are the binary requirements on the flow variables X_{ij}. Note that there are more complex SPPRC than those expressed by (5.4)-(5.8), i.e., using complex resource extension functions that, given the resource values T_i^r at node i, provide a lower bound on the resource values at node j.

As mentioned above, at each CG iteration, the dual values are used find new columns to add in the RMP. This dual information is included in the SPPRC by using a modified cost $\bar{c}_{ij} = c_{ij} - \sum_{k=1}^{m} \rho_{ij}^k \pi_k$ on each arc $(i,j) \in A$, where ρ_{ij}^k is a binary parameter, taking value 1 if the arc $(i,j) \in A$ contributes to the master problem constraint k, 0 otherwise. As an example, in vehicle routing, each constraint of the master problem (i.e., a set partitioning formulation) is associated with the task of visiting a client. Thus, an arc (i,j), where j is a client node, is considered as contributing to the constraint associated with the client j. The dual value of the constraint is therefore included in the modified cost of the arc (i,j).
The cost of a path $p \in \Omega$ can be written in terms of the individual arcs composing it, i.e.,

$$c_p = \sum_{(i,j)\in A} c_{ij} s_{ij}^p, \tag{5.9}$$

where s_{ij}^p is a binary parameter equal to 1 if the arc $(i,j) \in A$ is included in path p. The constraint coefficients $a_p^k, k \in \{1,2,\ldots,m\}$, with $\mathbf{a_p}^\top = [a_p^1, a_p^2, \ldots, a_p^m]$, can also be written as

$$a_p^k = \sum_{(i,j)\in A} \rho_{ij}^k s_{ij}^p. \tag{5.10}$$

The reduced cost $\bar{c}_p$ of a variable/path can then be defined as the sum of the modified costs of the arcs composing the path, i.e.,

$$\bar{c}_p = c_p - \pi^\top \mathbf{a_p} = c_p - \sum_{k=1}^{m} \pi_k a_p^k = \sum_{(i,j)\in A} s_{ij}^p (c_{ij} - \sum_{k=1}^{m} \rho_{ij}^k \pi_k) = \sum_{(i,j)\in A} s_{ij}^p \bar{c}_{ij}. \tag{5.11}$$

Therefore, solving the program (5.4)-(5.8) using the modified costs in (5.4) yields a column with the least reduced cost. However, by solving the program using the dynamic programming formulation, it is possible to obtain several paths at once, which is another strong point for using these methods, as it has proven to speed up the resolution and minimize the number of iterations performed.

5.2.2 Labeling algorithm

In dynamic programming, shortest path problems are solved by a labeling algorithm. Such an algorithm starts from the trivial path that contains only the source node s, then extends it recursively along all arcs that yields a feasible path until reaching the destination node t. A path $P = (v_0, v_1, \ldots, v_p)$ in G represents a sequence of visited nodes such that $(v_{i-1}, v_i) \in A, v_i \in V, \forall i = 1, 2, \ldots, p$ where v_p is the last node of the partial path and $v_0 = s$. In the algorithm, a label L encodes information about a partial path P such as the reduced cost $\bar{c}(L)$, the set of nodes $V(L)$ visited by the path, the last node visited by the path $v(L)$ and the accumulated quantity $T^r(L)$ of each resource $r \in R$. For a label L ending at node $v_i (v_i \neq t)$, a new label L' is obtained when an extension is performed along an arc $(i, j) \in A$, such that

$$v(L') = j, \tag{5.12}$$
$$V(L') = V(L) \cup \{j\}, \tag{5.13}$$
$$\bar{c}(L') = \bar{c}(L) + \bar{c}_{ij}, \tag{5.14}$$
$$T^r(L') = \max\{\underline{w}_j^r, T^r(L) + t_{ij}^r\}, \forall r \in R. \tag{5.15}$$

The label L' also keeps a reference to its predecessor label L, in order to build the complete path when the algorithm ends. The label is considered infeasible if $T^r(L') > \bar{w}_j^r$ and is therefore deleted.
In addition, to the infeasible labels that get rejected when extending the labels, a dominance rule can be applied to eliminate non-promising paths. A label L_1 is said to be dominating a label L_2 if the following conditions are verified :

$$v(L_1) = v(L_2), \tag{5.16}$$
$$V(L_2) \subseteq V(L_1), \tag{5.17}$$
$$\bar{c}(L_1) \leq \bar{c}(L_2), \tag{5.18}$$
$$T^r(L_1) \leq T^r(L_2), \quad \forall r \in R \tag{5.19}$$

The performance of the labeling algorithm is closely linked to the choice of the dominance rule, a good choice of the latter allows to reduce the search space and to eliminate a maximum of labels, allowing to accelerate the resolution. Note that the validity of a dominance rule depends closely on the variant of the SPPRC being solved. For simplicity, some details have been omitted, such as resource extension functions (REFs, [19]), which are considered being a more general way to define the resource arc consumptions, and extensions of the basic algorithm for the ESPPRC case to deal with cycle elimination.

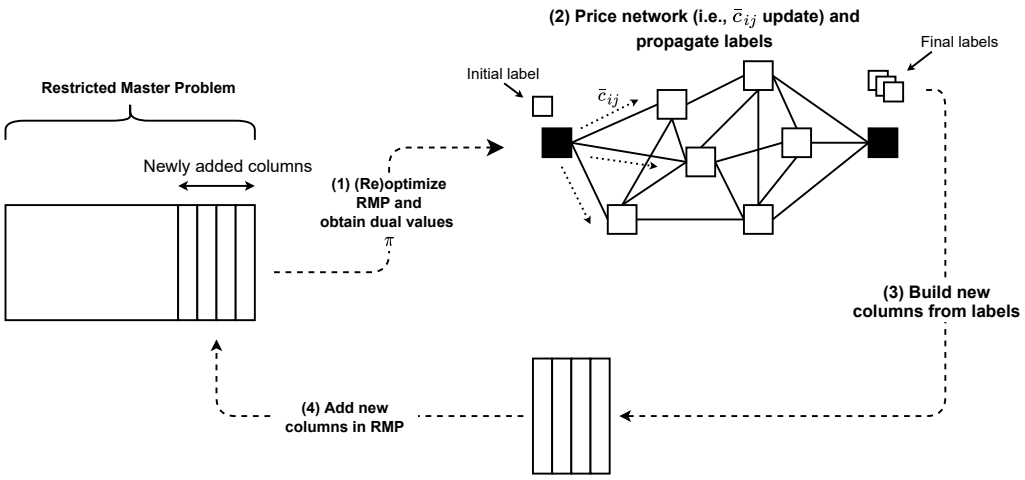


FIGURE 5.1 Summary of the different steps of a CG iteration.

At the end of the algorithm, several labels are obtained at the destination node t. The labels with a non-negative reduced cost are discarded, and those with a negative reduced cost are kept to build the new columns to be added in the RMP. The different steps discussed in this section are summarized in Figure 5.1.

5.3 Methodology

The goal of the project is to reduce the size of the underlying SPPRC network by keeping only the most promising arcs, therefore obtaining a reduced network. At each iteration, the same reduced network is used as long as it generates a satisfactory number of new negative reduced cost columns. The arc selection is thus performed only once before starting to solve the problem. In machine learning terms, this is a classification problem (i.e., supervised machine learning) and requires a labeled dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$, where $n = |\mathcal{D}|$ and each entry is a tuple that represents an input/output pair, the inputs $\boldsymbol{x}_i$ are called features and y_i are the desired outputs, also called labels (not the same notion of “label" as in a labeling algorithm) given by an expert or known in advance. Given the dataset, the learning algorithm learns to predict the outputs y_i from the feature vectors $\boldsymbol{x_i}$. In our case, it is a binary classification problem (i.e., $y_i \in \{0, 1\}$), so each arc is either selected or not. During the training phase, the algorithm tries to minimize the difference between the true known values y_i and the values predicted by the model denoted by $\hat{y}_i$. However, the goal is to be able to generalize and give good predictions for inputs never encountered during training, so a portion of the dataset is reserved for testing, in order to evaluate more accurately the performance of the model.

5.3.1 Features

The extracted features represent the characteristics of each individual arc $(i, j) \in A$. They can be different depending on the problem in hand and the structure of the underlying network. For the VCSP and VRPTW considered in this paper, some of the features are similar, namely :

- Cost of the arc c_{ij}
- For each resource $r \in R$, the resource consumption t_{ij}^r along the arc (i, j)
- Number of arcs leaving node i
- Number of arcs entering node j
- For each resource $r \in R$, the minimum, maximum and average resource consumptions along the arcs leaving node i
- For each resource $r \in R$, the minimum, maximum and average resource consumptions along the arcs entering node j
- For each resource $r \in R$, the upper and lower resource bounds for nodes i and j.

The other features specific to each problem will be described in their dedicated sections.

5.3.2 Labels

In supervised learning, the labels are either known in advance or assigned by an expert. In our case, we have to define what are the promising arcs to keep. The algorithm followed to assign the labels is in most cases computationally expensive, so we try to collect as much data as possible, then train a machine learning model that gives predictions in a reasonably fast computing time. Let $\mathcal{C}_i$ be the set of columns generated at iteration i, and A_c the set of arcs visited by the path represented by column $c \in \mathcal{C}_i$. A promising arc is an arc that has been used at least once to generate columns at any iteration. The idea is that the generated columns must have dominated many other columns during the resolution of the PP and are therefore all good candidates, and so are the arcs composing their paths. According to the tests on the VCSP and VRPTW instances, the percentage of arcs used at least once during the resolution of the linear relaxation by CG does not exceed 18%, and this number decreases when moving to larger instances. Another idea is to consider only the positive basic columns (i.e., columns in the optimal basis of an RMP with a positive value), which results on a larger reduction of the graph, but according to the results, the number of iterations slightly increases compared to considering all the generated columns and the gain is therefore lost. Figure 5.2 shows a visual comparison of the number of arcs for a VRPTW instance, considering different scenarios : (a) all arcs, (b) arcs in the generated columns, (c) arcs in an RMP solution, and (d) arcs in the linear relaxation solution. The figure also specifies for each scenario the percentage of arcs selected with respect to the complete arc set.

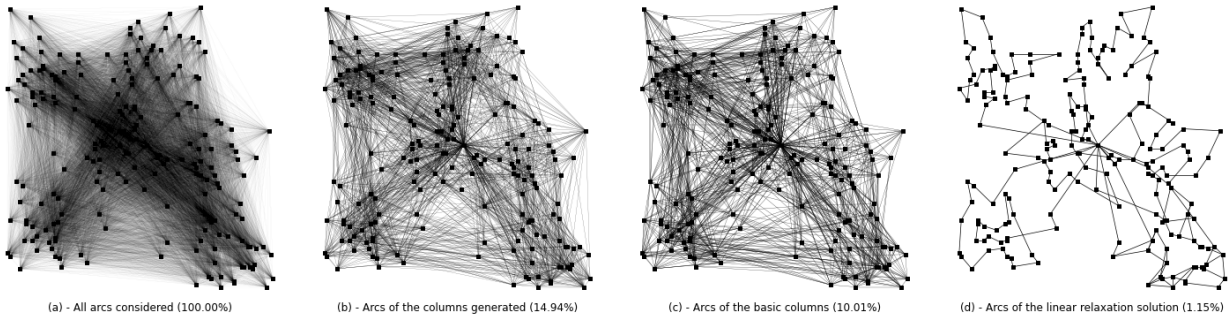


FIGURE 5.2 Number of arc comparisons across different scenarios

5.3.3 Data collection

Now that we have defined the arcs to be selected and the features, we can proceed and put all the pieces together to define our algorithm for data collection described in Algorithm 2. The steps of the algorithm are fairly straightforward. First, we start with an empty dataset in Step (1) and we initialize all arc labels with the value 0 in Steps (2)-(4). In (5) an initial solution is obtained, either by using a heuristic or a two-phase simplex algorithm. Then, at each CG iteration, the RMP is reoptimized and the PP is solved, generating a new set of negative reduced cost columns $\mathcal{C}_i$ (Steps (7)-(8)). In case the PP defined over the full network G fails to generate any column, the CG process stops and the current solution is optimal (Steps (9)-(11)). Otherwise, for each newly generated column $c \in \mathcal{C}_i$, the label $y_a = 1$ is assigned to each arc $a \in A_c$ composing the path represented by the column c (Steps (12)-(16)) and the generated columns are added to the RMP which is repotimized (Steps (17)-(18)). Finally, when the CG process ends, for each arc $a \in A$ the pair of features $\boldsymbol{x}_a$ and labels y_a are collected and added to the dataset $\mathcal{D}$ (Steps (20)-(23)).
Note that it is necessary to solve several instances of the problem to optimality in order to collect enough data. The features can be extracted before solving the problem but for the labels it is necessary to wait until the end of the optimization. In our context, the instances to solve with CG will not necessarily be of the same size as those used during the training of the model. For this reason, we chose to collect data from instances of the same size for the machine learning phase, and use instances of different sizes during the evaluation of the model when integrated in CG. The size of the instances will be highlighted in the sections dedicated for each of the two applications.

5.3.4 ML pricing algorithm

Once the data is available, a preprocessing phase is performed. It consists of encoding the categorical values and the normalization of the data. The dataset is then split into a training,

Algorithm 2 Data collection : features and labels extraction

$\mathcal{D} = \emptyset$
for `each` $a \in A$ **do**
    $y_a = 0$
end for
$\pi \longleftarrow$ `initialRMPSolution()`
for `each CG iteration` i **do**
    `priceNetwork(G,`π`)`
    $\mathcal{C}_i \longleftarrow$ `generateColumns(G)`
    if $\mathcal{C}_i == \emptyset$ **then**
        `break`
    end if
    for `each` $c \in \mathcal{C}_i$ **do**
        for `each` $a \in A_c$ **do**
            $y_a = 1$
        end for
    end for
    `addColumns(`$\mathcal{C}_i$`)`
    $\pi \longleftarrow$ `reoptimizeRMP()`
end for
for `each` $a \in A$ **do**
    $\boldsymbol{x}_a \longleftarrow$ `extractData(`a`)`
    $\mathcal{D} \longleftarrow \mathcal{D} \cup \{(\boldsymbol{x}_a, y_a)\}$
end for

a validation and a test set. The validation set is used to tune the different hyperparameters of the classification algorithms (more details about the algorithms and their results will be given for each of the two applications), while the purpose of the test set is to give a final evaluation before integrating the learned model into CG. Once the learning phase is completed, the obtained model can be used to select the arcs and build the reduced network. Before starting the first CG iteration, the learned model takes the features as an input and gives predictions $\hat{y}_a$ for each arc $a \in A$. The reduced network is defined as $G_r = (V, A_r)$ where $A_r = \{a \in A \mid \hat{y}_a = 1\}$ and it is used to generate columns as long as it yields a satisfactory number of columns, i.e., a number higher than a parameter value η_{min}. When the reduced network fails to generate enough columns, the complete network is used instead. If the latter generates a number of columns higher than a parameter value η_{max}, we switch back to the reduced network until we reach an optimal solution. Algorithm 3 details how the ML model is used in practice.

Algorithm 3 Machine-learning-based pricing heuristic in CG

for `each` $a \in A$ **do**
    $\boldsymbol{x}_a \longleftarrow$ `extractFeatures(`a`)`
    $\hat{y}_a \longleftarrow$`predict(`$\boldsymbol{x}_a$`)`
end for
$A_r := \{a \in A \mid \hat{y}_a = 1\}$
$G_r := (V, A_r)$
$G_{active} \longleftarrow G_r$
`useReducedG` $\longleftarrow$ `True`
$\pi \longleftarrow$ `initialRMPSolution()`
for `each CG iteration` i **do**
    `priceNetwork(`G_{active}`,`π`)`
    $\mathcal{C}_i \longleftarrow$ `generateColumns(`G_{active}`)`
    if $(|\mathcal{C}_i| < \eta_{min} \wedge$ `useReducedG`$)$ **then**
        `useReducedG` $\longleftarrow$ `False`
        $G_{active} \longleftarrow G$
    else if $|\mathcal{C}_i| \geq \eta_{max} \wedge (\neg$`useReducedG`$)$ **then**
        `useReducedG` $\longleftarrow$ `True`
        $G_{active} \longleftarrow G_r$
    else if $\mathcal{C}_i == \emptyset \wedge (\neg$`useReducedG`$)$ **then**
        `Exit`
    end if
    if $|\mathcal{C}_i| > 0$ **then**
        `addColumns(`$\mathcal{C}_i$`)`
        $\pi \longleftarrow$ `reoptimizeRMP()`
    end if
end for

The algorithm starts by extracting the features of the arcs and getting the model predictions (Steps (1)-(4)). The predictions $\hat{y}_a$ are then used to build the reduced graph G_r that is set as the active graph to be used when solving the PP (Steps (5)-(8)). Note that the active graph G_{active} can either be the original complete graph G or the reduced one G_r. The boolean variable *useReducedG* is also initialized to `True`, indicating that the reduced graph is being used (e.g., equivalent to $G_{active} == G_r$). In Step (9), an initial solution to the RMP is obtained along with dual values π. At each CG iteration, the currently active network is priced and a new set of columns $\mathcal{C}_i$ is generated (Steps (11) and (12)). If the columns were generated by the reduced network and the number of columns $|\mathcal{C}_i| < \eta_{min}$, we switch to using the full network by setting the variables G_{active} to G and *useReducedG* to `False` (Steps (13)-(15)). On the contrary, if the full network was used and $|\mathcal{C}_i| \geq \eta_{max}$, we switch back to using the reduced network G_r and set *usedReducedG* to `True` (Steps (16)-(18)). By using the two previous conditions, the algorithm switches between the two graphs until an optimal solution is obtained. Normally, with the right choice of the two parameters η_{min} and η_{max}, at the last iterations of the CG, the reduced network will not generate enough columns (i.e., less than η_{min}) and so the full network G will be used instead. Most likely, the full network will neither generate many columns (i.e., less than η_{max}), so the algorithm will not switch back to the reduced network and will keep using the full network until it generates no more columns, indicating that the obtained solution is optimal (Steps (19)-(21)). If columns were generated, they are added to the RMP which is reoptimized afterwards (Steps (22)-(25)).

5.4 Application I : Vehicle and crew scheduling problem

In a public transit system, the planning process goes through several stages. The first step in the process is to determine the bus lines, the stops on each line, and the frequency of the trips. Based on these pieces of information, a timetable is created, describing the trips with their corresponding starting and ending times and locations. The next two steps in the process are the construction of vehicle routes and crew schedules, which means solving, respectively, the two scheduling problems : the *Vehicle scheduling problem* and the *Crew scheduling problem*. Traditionally, these two problems are solved in a sequential manner, where the vehicle routes are determined first before the crew schedules. However, this approach is not guaranteed to provide the best solution, because in most cases driver costs dominate the costs of vehicle use. A first formulation that integrates and considers the two problems simultaneously was proposed by Freling et *al.* [64], giving rise to the VCSP. For solving the problem, the authors described a CG approach applied to a Lagrangian relaxation of the master problem, since the problem contains a very large number of variables representing the feasible duties. Most of the formulations proposed in the literature are set partitioning ones, where CG plays an

important role to deal with the large number of variables and to generate schedules as needed. Given the difficulty of the problem, several heuristics have been proposed in the literature as well. In this paper, we consider the formulation described in [31], where the RMP is the linear relaxation of a set partitioning problem with side constraints and the PP is a SPPRC. The RMP formulation contains only the crew schedule variables, but side constraints are added to the problem to ensure that the vehicle schedules can be obtained afterwards in a polynomial time. In addition, the costs of the vehicles are included in the objective function to ensure that an overall optimal solution is obtained.

5.4.1 Network structure

Since the SPPRC is the part of interest for our work, we will take a closer look at the network structure of the problem before diving into the details about the data collection, the instances used and the results obtained.

In this problem, a network is used to generate driver schedules and there is one network for each schedule type (e.g, regular schedule with one meal break or straight without a meal break). Additionally, the arcs can be divided into two categories : 1) bus movement when the driver is driving or attending a bus and 2) walking when the driver is moving on his/her own for repositioning purposes. As mentioned before, each bus line is defined by three different trip nodes : departure node, arrival node and intermediate nodes where an exchange of drivers can be performed. For each bus line, several trips are planned during the day at different times, the exact time at which a bus should arrive at each of the trip nodes is therefore known. To cover the trips, the buses leave the depot in the direction of the trip departure nodes. Each bus can be used for multiple trips, and can be operated by different drivers before returning to the depot at the end of the day. Since the same bus must service the whole trip, a bus moving to the departure node of a trip must be retained until it arrives at the destination node, at which point it can head to the start of the next trip. Therefore, no bus movements leaving the departure or intermediate nodes to a different trip are allowed. In fact, the bus movements represent less than 7% of the total number of arcs, while the majority of the remaining 93% are the walking movements of the drivers. Unlike the bus movements, the walking movements are not as restricted because they can occur at the beginning, the end or the middle of the trips, as long as they are feasible. Figure 5.3 is a simplified and non-detailed representation of the main components found in a VCSP. In addition to the main components, the network can also include break nodes and their corresponding arcs to model the breaks that drivers can take after a certain working time. Some details are not involved in our selection strategy and therefore not detailed here, but the interested readers can consult [31] for an in-depth overview.

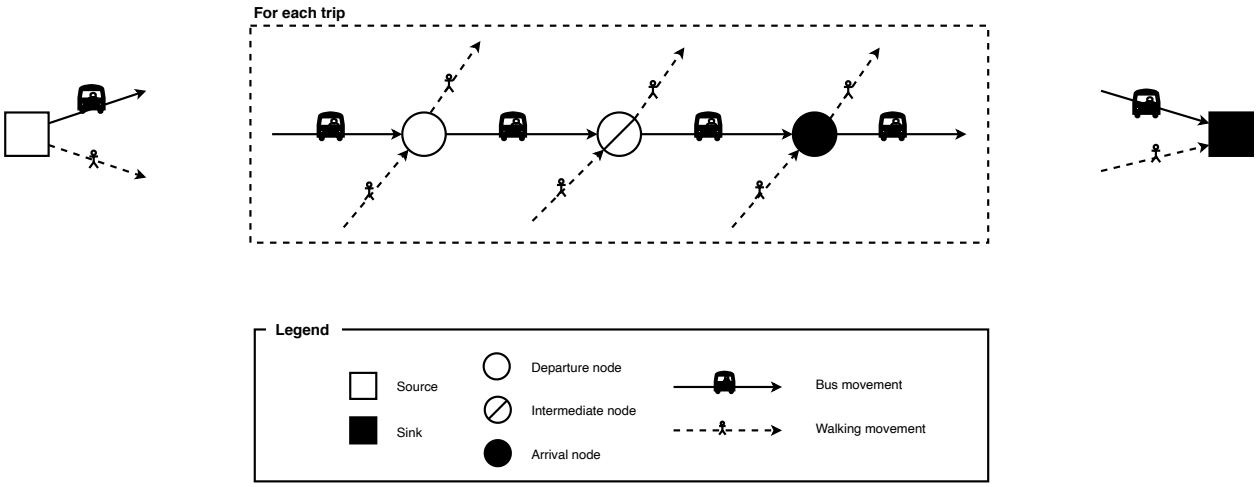


FIGURE 5.3 A simplified version of the network components in the VCSP.

For us, the movements we are interested in are the walking movements of the drivers between trip nodes, since they represent more than 90% of the total number of arcs. The selection strategy only covers this subset of arcs, denoted as $A_s \subset A$ and not all arcs in A. For a walking movement arc linking the trip nodes of two different trips $T1$ and $T2$, the classification model will try to predict whether there is a high chance that a driver leaves his/her current trip $T1$ in the direction of trip $T2$. The arcs entering and leaving the depot, as well as the bus movements entering and leaving the trip nodes are not involved, in addition to any extra arcs for breaks, etc.

5.4.2 Data collection, features and labels

All the features mentioned in Section 5.3.1 are valid for the VCSP. For the resource consumptions on the arcs, the two representing the working time and waiting time, are respectively extracted. The first resource is the time taken to walk to the next trip node, while the second resource is the waiting time required before actually starting the trip. Note that the waiting time is bounded and cannot exceed the upper bound indicated by the resource windows. Given that the exact time at which the trip nodes must be visited is known, the times associated with node i and j for each arc $(i, j) \in A_s$ are added to the list of features as well. The difference between j and i times is equal to the time required to walk from i to j, plus the waiting time, but since there are more trips during peak hours, the time of the day information can be useful. Another feature that can be added to the list is the arc type since the arcs we are interested in can link three different types of nodes (i.e., departure, intermediate and arrival nodes) yielding nine possible connection types. Therefore, with the additional features, a total of 22 features is collected for each arc.

As described in Section 5.3.2, labels are assigned to the arcs that are part of the generated columns. However, only the arcs representing the walking movements between the trip nodes are concerned.

5.4.3 VCSP instances

To collect enough data, an instance generator is used as in [31]. By taking a total number of trips as an input, the generator splits this number over the different bus lines. The trips are then randomly distributed over the time horizon, with more trips during peak hours. A total of 20 instances with 400 trips were used in the training phase, each instance added about 250,000 data points to the dataset. Other new instances of different sizes (i.e., $300, 350, \ldots, 600$) are generated and used to test the integration of the ML model in the CG algorithm. The instances of size other than 400 also serve to evaluate the generalization of the model to unseen instances of different sizes.

5.4.4 Computational results

This section is divided into three parts. The first one is devoted to the machine learning phase, where additional details about data collection, the algorithms used and the results obtained are given, whereas the second part presents the results of the integration of the model in the CG algorithm. In the third part, we highlight the results of some additional experiments that are worth mentioning. All the experiments were performed on a Linux machine with an i7-8700 CPU @ 3.20GHz and 64GB of RAM.

Machine learning

With the dataset in hand, some preprocessing is performed such as data normalization and one-hot encoding of categorical features (i.e., arc type). The next step is to choose the appropriate classification algorithm to use. There are several possibilities, from the simplest linear model (i.e., Logistic regression) to the more classic algorithms (SVM, Decision Trees, etc.) and more complex models (Artificial Neural Networks, etc). To compare the models with each other, the main criteria is their accuracy on the test set. However, when there is an imbalance between the output classes (i.e., in our case, only about 15% of the data have label 1 and the remaining have label 0), it is better to consider other metrics such as the true positive rate (TPR, also called the Recall), the true negative rate (TNR), precision, etc. These metrics provide an idea about the performance obtained for each individual class allowing a better comparison between the different models.
Three different algorithms have been chosen for the training phase, starting with a linear classifier : Logistic regression, then moving to two more advanced algorithms : Random Forest

TABLE 5.1 Hyperparameters values used in the training phase.

Algorithm	Hyperparameter	Value
Logistic regression	C parameter	1
	Solver	lbfgs
	Class weights	Balanced
Random forest	Bootstrap	True
	Max depth	10
	Max features	7
	Min samples per leaf	50
	Min samples per split	100
	Number of trees	500
	Class weights	Balanced
Neural Network	Learning rate	10^{-3}
	Epochs	1000
	Batch size	32
	Loss function	Binary cross entropy
	Activation function	ReLU
	Architecture	32x32x32x1
	Optimizer	Adam
	Class weights	7 :1

(RF) and Artificial Neural Networks (ANN). Each of these algorithms requires hyperparameters to be tuned, a “cross validation" approach is used to compare the different set of values. The best values obtained are described in Table 5.1, while Table 5.2 shows the results of the three algorithms on the test set.

The Recall represents the percentage of arcs with label $y_a = 1$ that are correctly classified, while the TNR represents the accuracy of the opposite class (i.e., classification accuracy of the arcs with label 0). Ideally, a high percentage of both is sought. The last column represents the balanced accuracy which is the average of the two previous columns. To deal with the unbalanced data, weights are used during the optimization of the loss function. The goal is to give a higher penalty to the arcs with label 1 that are misclassified since they are a minority in the dataset, hence the use of the “balanced" setting in the class weights hyperparameter. One can imagine the dataset as a cloud of points, where 85% are red and 15% are green. With logistic regression, one tries to separate these two classes with a linear separator, but since there are more red points than green, there is more chance to have a large number of red points on one side of the separator and thus a high TNR. A high recall can also be obtained if the two classes are easily separable, which does not seem to be the case according

TABLE 5.2 VCSP metrics of three different algorithms.

Algorithm	**Recall**	**TNR**	**Balanced accuracy**
Logistic regression	64%	81%	72%
Random forest	76%	81%	78%
Neural network	78%	80%	79%

to the results obtained (i.e., 64% Recall and 81% TNR). The model slightly underfits but overall, the accuracy obtained is not far from the other two algorithms. On the other hand, the RF and ANN models are able to reach a higher accuracy of 78-79%, with more balance between the Recall and the TNR. The values obtained by the two algorithms are quite close to each other.

CG with arc selection

After the training phase, both the RF and ANN models seem to be good candidates to be integrated in the CG algorithm. Given their almost similar accuracy, one can expect them to give comparable results. When testing the models on several groups of instances with different sizes, the RF model performs better on some instances, but on others it is the ANN model that is better. However, because the RF model is a few percents ahead on average, we retained this candidate for the subsequent tests.

As described in Algorithm 3, the model is used before the beginning of the CG process to build the reduced network G_r. Different scenarios are compared to assess the efficiency of the arc selection, namely :

Baseline-CG : This corresponds to the basic CG with no arc selection, i.e., network G is used in all iterations.

ML-S : Arc selection is used and the CG alternates between the use of the reduced network G_r and the full network G. The reduced network is built from the predictions obtained by the RF model. The parameters used during the CG process are : $\eta_{min} = 30, \eta_{max} = 100$.

Random-S : This scenario corresponds to a completely random selection of the arcs in A_s, the number of selected arcs is the same as with the model. This can be achieved by simply shuffling the values obtained by the model.

Cost-S : For each trip node i, incoming arcs $(n, i) \in A_s$ and outgoing arcs $(i, n) \in A_s$ are sorted in ascending order of their cost, and a subset of arcs with the lowest costs is selected. The total number of selected arcs is approximately the same as with the model.

The results obtained by the four algorithms are reported in Table 5.3. The names of the

TABLE 5.3 VCSP results.

Instance	Baseline-CG				ML-S					Random-S					Cost-S				
	#Itr	Time (s)			#Itr	Time (s)			Gain	#Itr	Time (s)			Gain	#Itr	Time (s)			Gain
		PP	RMP	Total		PP	RMP	Total			PP	RMP	Total			PP	RMP	Total	
VCS_300_1	158	133	80	213	146 (12)	75	76	151	**29%**	180 (59)	114	91	205	**4%**	168 (34)	94	82	176	**17%**
VCS_300_2	154	174	85	259	158 (8)	106	91	197	**24%**	242 (72)	214	122	336	**-30%**	183 (37)	143	94	237	**8%**
VCS_300_3	157	182	81	263	154 (14)	115	76	191	**27%**	204 (70)	198	100	298	**-13%**	182 (37)	160	86	246	**6%**
VCS_300_4	176	222	81	303	185 (12)	166	81	247	**18%**	227 (105)	220	96	316	**-4%**	220 (66)	241	84	325	**-7%**
VCS_300_5	199	204	89	293	191 (27)	133	87	220	**25%**	238 (60)	184	95	279	**5%**	192 (44)	166	87	253	**14%**
Average	**169**	**183**	**83**	**266**	**167 (14)**	**119**	**82**	**201**	**25%**	**218 (73)**	**186**	**101**	**287**	**-13%**	**189 (43)**	**161**	**87**	**248**	**8%**
VCS_350_1	136	183	126	309	143 (15)	112	129	241	**22%**	199 (67)	207	151	359	**-16%**	175 (42)	166	144	310	**0%**
VCS_350_2	148	158	136	294	135 (9)	85	119	204	**31%**	184 (67)	146	142	288	**2%**	192 (52)	153	150	303	**-3%**
VCS_350_3	182	253	179	432	170 (10)	141	167	308	**29%**	219 (61)	226	207	433	**0%**	192 (37)	192	178	370	**14%**
VCS_350_4	171	276	162	438	188 (7)	173	174	347	**21%**	302 (97)	362	243	605	**-38%**	227 (51)	291	173	464	**-6%**
VCS_350_5	201	360	164	524	234 (12)	272	183	455	**13%**	273 (101)	392	222	614	**-17%**	249 (48)	347	179	526	**0%**
Average	**168**	**246**	**153**	**399**	**174 (10)**	**157**	**154**	**311**	**23%**	**215 (78)**	**267**	**193**	**460**	**-5%**	**207 (46)**	**230**	**165**	**395**	**1%**
VCS_400_1	177	370	253	623	177 (10)	238	258	496	**20%**	212 (89)	361	293	654	**-5%**	216 (38)	342	297	639	**-3%**
VCS_400_2	176	422	246	668	189 (8)	255	258	513	**23%**	240 (86)	471	298	769	**-15%**	214 (58)	378	257	635	**5%**
VCS_400_3	153	266	242	508	163 (9)	177	233	410	**19%**	176 (64)	263	238	501	**1%**	184 (42)	245	254	499	**2%**
VCS_400_4	147	209	206	415	147 (12)	126	190	316	**24%**	199 (64)	225	224	449	**-8%**	173 (30)	173	222	395	**5%**
VCS_400_5	207	518	256	774	180 (12)	283	230	513	**34%**	266 (100)	558	311	869	**-12%**	214 (44)	391	252	643	**17%**
Average	**172**	**357**	**241**	**598**	**171 (10)**	**216**	**234**	**450**	**24%**	**218 (80)**	**376**	**273**	**648**	**-8%**	**200 (42)**	**306**	**256**	**562**	**5%**
VCS_450_1	340	1114	640	1754	368 (14)	809	636	1445	**18%**	426 (138)	1034	713	1747	**0%**	366 (64)	868	637	1505	**14%**
VCS_450_2	269	1540	544	2084	241 (11)	797	486	1283	**38%**	358 (126)	1453	663	2116	**-2%**	281 (45)	1071	565	1636	**21%**
VCS_450_3	194	570	396	966	194 (10)	319	391	710	**27%**	243 (81)	497	463	960	**1%**	251 (64)	508	435	943	**2%**
VCS_450_4	209	505	387	892	234 (8)	312	440	752	**16%**	264 (110)	473	446	919	**-3%**	255 (52)	439	439	878	**2%**
VCS_450_5	279	1507	551	2058	289 (21)	976	564	1540	**25%**	376 (124)	1566	681	2247	**-9%**	350 (61)	1510	568	2078	**-1%**
Average	**258**	**1047**	**504**	**1551**	**265 (12)**	**643**	**503**	**1146**	**25%**	**333 (115)**	**1005**	**593**	**1598**	**-3%**	**300 (57)**	**879**	**529**	**1408**	**8%**
VCS_500_1	277	1618	856	2474	304 (8)	1030	902	1932	**22%**	387 (132)	1738	1021	2759	**-12%**	308 (60)	1272	916	2188	**12%**
VCS_500_2	296	1654	620	2274	270 (13)	846	576	1422	**37%**	366 (130)	1404	738	2142	**6%**	342 (76)	1318	691	2009	**12%**
VCS_500_3	203	770	604	1374	221 (10)	453	611	1064	**23%**	291 (101)	887	736	1623	**-18%**	266 (54)	688	713	1401	**-2%**
VCS_500_4	266	1839	748	2587	281 (8)	1235	749	1984	**23%**	357 (130)	1842	908	2750	**-6%**	313 (54)	1526	789	2315	**11%**
VCS_500_5	322	1720	627	2347	340 (23)	1024	656	1680	**28%**	442 (173)	1635	769	2404	**-2%**	386 (103)	1486	701	2187	**7%**
Average	**273**	**1520**	**691**	**2211**	**283 (12)**	**918**	**699**	**1616**	**27%**	**368 (133)**	**1501**	**834**	**2336**	**-8%**	**323 (69)**	**1258**	**762**	**2020**	**8%**
VCS_550_1	254	1723	842	2565	275 (9)	1125	913	2038	**21%**	368 (137)	2101	1084	3185	**-24%**	309 (61)	1421	899	2320	**10%**
VCS_550_2	280	1347	894	2268	280 (18)	801	971	1772	**22%**	393 (145)	1583	1116	2699	**-19%**	324 (87)	1289	976	2265	**0%**
VCS_550_3	286	1245	822	2067	269 (21)	659	829	1488	**28%**	326 (103)	960	943	1903	**8%**	302 (70)	888	907	1795	**13%**
VCS_550_4	259	1870	863	2733	262 (13)	980	895	1875	**31%**	406 (132)	2130	1186	3316	**-21%**	294 (48)	1207	954	2161	**21%**
VCS_550_5	199	782	774	1556	206 (14)	472	788	1260	**19%**	242 (67)	632	905	1537	**1%**	228 (54)	608	858	1466	**6%**
Average	**256**	**1399**	**839**	**2238**	**258 (15)**	**807**	**879**	**1687**	**24%**	**347 (116)**	**1481**	**1047**	**2528**	**-11%**	**291 (64)**	**1083**	**919**	**2001**	**10%**
VCS_600_1	290	2840	1229	4069	298 (32)	1694	1235	2929	**28%**	442 (171)	3416	1595	5011	**-23%**	320 (69)	1972	1300	3272	**20%**
VCS_600_2	246	1286	948	2234	249 (20)	815	972	1787	**20%**	319 (114)	1327	1136	2463	**-10%**	331 (81)	1324	1185	2509	**-12%**
VCS_600_3	279	2120	1398	3518	290 (11)	1236	1344	2580	**27%**	392 (114)	2208	1693	3901	**-11%**	373 (57)	1937	1537	3474	**1%**
VCS_600_4	258	2103	1327	3430	272 (15)	1286	1322	2608	**24%**	379 (104)	2483	1618	4101	**-20%**	321 (48)	1767	1475	3242	**5%**
VCS_600_5	247	1722	1186	2908	245 (16)	905	1190	2095	**28%**	313 (96)	1554	1395	2949	**-1%**	290 (54)	1510	1315	2825	**3%**
Average	**264**	**2014**	**1218**	**3232**	**270 (18)**	**1187**	**1213**	**2400**	**25%**	**369 (119)**	**2198**	**1487**	**3685**	**-13%**	**327 (61)**	**1702**	**1362**	**3064**	**3%**

instances are written in the form "VCS_*Size*_*Id*", where *Size* is the instance size, i.e., the number of trips, and *Id* is the instance identifier. For each algorithm, we report the number of CG iterations required to reach the linear relaxation solution, the computing time in seconds of : the PP, the RMP and in total. For the algorithms involving an arc selection, an additional column shows the reduction of the computing time gained in comparison with the "Baseline-CG" algorithm. Additionally, the number of iterations performed using the full network G is indicated in parentheses. After each group of instances of the same size, a line indicating the average values is added.

According to the results, if we start by comparing Baseline-CG and ML-S, we can observe a reduction in computing time ranging from 23 to 27%, mostly on the PP side. One can also notice that the highest reductions are obtained when the number of iterations is lower

than the one with Baseline-CG. On average, the number of iterations is very close or slightly higher. Therefore, the ML selection allows a reduction of the size of the PP network without increasing considerably the number of iterations. We can also notice that the performance of the model is maintained across all groups of instances. Considering that the model has been trained only on 400-trip instances, this also shows the ability of the model to generalize to different instance sizes.

For the random selection case (i.e., Random-S), from the negative gains obtained we can conclude that this selection is not very promising. This is reflected by the number of iterations performed that is higher than the one obtained using Baseline-CG, and also by the number of times the full network was used if we compare it to ML-S. On the other hand, the results obtained by Cost-S show that a selection based on the costs is more effective. One can consider that Cost-S is a selection based on a single feature, which is the cost, while ML-S uses several features at the same time to decide whether an arc is to be retained or not. The results lie between those of the two algorithms Random-S and ML-S, i.e., it is better than a completely random selection but worse than the one with a learned model. This is reflected by the total number of iterations, the number of times the full network G was used and the reductions obtained.

Additional experiments

This section is dedicated to some additional experiments that have been conducted. Even though they did not give significantly better results, we think that they deserve to be highlighted here.

- The first idea has been previously mentioned, which consists of selecting only the arcs that are part of the columns in the optimal basis of the solved RMPs. In the training phase, this strategy gave slightly worse results, i.e., 75% accuracy, and when integrated in the CG algorithm, the extra gain obtained by reducing further the network size is nullified by doing more iterations.
- Another idea is to assign weights to the arcs according to the number of times they have been used in the generated columns. An accuracy of 78% is obtained in the training phase, distributed as 86% recall and 70% TNR, but again, the results were not any better when integrated in CG.
- We made a small modification to the initial algorithm which consists of skipping the label assignment at the first n iterations, where n is a given parameter. The idea is that some arcs can be interesting at the beginning of the optimization but never used at later iterations. Like the other changes, the results obtained were not really compelling.

TABLE 5.4 VCSP results with column selection.

Instance	ML-S				ML-COL-S				Gain
	#Itr	Time (s)			#Itr	Time (s)			
		PP	RMP	Total		PP	RMP	Total	
VCS_300_1	146 (12)	75	76	151	123 (10)	64	70	134	**11%**
VCS_300_2	158 (8)	106	91	197	140 (7)	103	78	181	**8%**
VCS_300_3	154 (14)	115	76	191	135 (10)	101	66	167	**13%**
VCS_300_4	185 (12)	166	81	247	161 (6)	128	67	195	**21%**
VCS_300_5	191 (27)	133	87	220	142 (9)	102	76	178	**19%**
Average	**167 (14)**	**119**	**82**	**201**	**140 (8)**	**99**	**71**	**171**	**14%**
VCS_350_1	143 (15)	112	129	241	126 (7)	106	125	231	**4%**
VCS_350_2	135 (9)	85	119	204	124 (9)	81	105	186	**9%**
VCS_350_3	170 (10)	141	167	308	163 (7)	139	156	295	**4%**
VCS_350_4	188 (7)	173	174	347	179 (9)	198	148	346	**0%**
VCS_350_5	234 (12)	272	183	455	187 (9)	234	155	389	**15%**
Average	**174 (10)**	**157**	**154**	**311**	**155 (8)**	**151**	**137**	**289**	**6%**
VCS_400_1	177 (10)	238	258	496	161 (7)	229	232	461	**7%**
VCS_400_2	189 (8)	255	258	513	175 (11)	255	228	483	**6%**
VCS_400_3	163 (9)	177	233	410	148 (7)	176	203	379	**8%**
VCS_400_4	147 (12)	126	190	316	122 (6)	108	166	274	**13%**
VCS_400_5	180 (12)	283	230	513	200 (7)	316	236	552	**-8%**
Average	**171 (10)**	**216**	**234**	**450**	**161 (7)**	**216**	**213**	**429**	**5%**
VCS_450_1	368 (14)	809	636	1445	319 (10)	723	513	1236	**14%**
VCS_450_2	241 (11)	797	486	1283	232 (6)	833	430	1263	**2%**
VCS_450_3	194 (10)	319	391	710	173 (9)	314	355	669	**6%**
VCS_450_4	234 (8)	312	440	752	190 (9)	280	349	629	**16%**
VCS_450_5	289 (21)	976	564	1540	263 (8)	918	489	1407	**9%**
Average	**265 (12)**	**643**	**503**	**1146**	**235 (8)**	**613**	**427**	**1040**	**9%**
VCS_500_1	304 (8)	1030	902	1932	292 (7)	986	888	1874	**3%**
VCS_500_2	270 (13)	846	576	1422	259 (8)	852	570	1422	**0%**
VCS_500_3	221 (10)	453	611	1064	177 (7)	416	550	966	**9%**
VCS_500_4	281 (8)	1235	749	1984	247 (6)	1134	670	1804	**9%**
VCS_500_5	340 (23)	1024	656	1680	299 (11)	916	604	1520	**10%**
Average	**283 (12)**	**918**	**699**	**1616**	**254 (7)**	**860**	**656**	**1517**	**6%**

– In the same context of combining ML and CG, in our previous paper [77], we were interested in reducing the computing time of the RMPs by selecting the most promising columns at each iteration. Reductions in computation time of up to 30% were achievable. However, the developed approach was more intended for problems that take the majority of the time in solving the RMPs and not the PPs, e.g., the tests were performed on instances that take on average 75% or more of the computing time in solving the RMPs. Nevertheless, it was interesting to check if column selection can yield an additional gain when used with ML-S. The results comparing ML-S and ML-S with column selection, i.e., ML-COL-S, are reported in Table 5.4.
From these results, we observe a small additional reduction in the average computing time per instance group ranging from 5% to 14%.

5.5 Application II : Vehicle routing problem with time windows

The vehicle routing problem (VRP) is a well-known and classical problem that has been extensively studied in the literature. Given a fleet of vehicles, the VRP consists of constructing routes to serve geographically dispersed clients, while minimizing the travel costs and respecting the capacity of the vehicles. Each route must start at the depot, visit a set of clients and then return to the depot at the end of the tour. The variant with time windows (VRPTW) adds an additional complexity by introducing a restriction on the times during which the clients can be visited (see [72]). Each client must be served during the associated time window by exactly one vehicle, and in case the vehicle arrives earlier, it has the possibility to wait until the client is available.

The most efficient and recent methods to solve this problem are based on CG, embedded in a branch-and-bound framework, with the addition of cuts to improve the bounds quality, giving rise to what is called "*branch-price-and-cut*" algorithms. CG is used to solve the linear relaxation at each node of the tree, where the master problem is most likely a set partitioning problem and the PP is an ESPPRC solved using dynamic programming algorithms. The elementarity requirement of the PP makes the problem much more difficult to solve than the standard SPPRC, especially for large instances where several clients can be visited by one route. To overcome these difficulties, researchers have investigated several relaxations (see [22–24]), seeking a good trade-off between the quality of the bound and the difficulty to solve the problem. Other studies have explored heuristic solutions [23,33]), such as dominating on a subset of resources, limiting the number of labels kept at each node, or reducing the number of arcs in the network based on their reduced cost. Note that many of these techniques can be combined and used simultaneously, allowing a large reduction in computing time. For more details on the exact methods for solving the VRP, the survey of Costa et *al.* [27] covers a wide range of methods and explores different variants of the problem. Similar to the recent formulations proposed in the literature, the model used in this work is a set partitioning problem and the PP is a SPPRC with 2-cycle elimination [69] solved using a labeling algorithm.

In this section, we will first describe the network structure of the problem, before giving more details about the the data collection and the instances used. Finally, we will discuss the results obtained.

5.5.1 Network structure

We consider the basic network structure of the VRPTW with a single depot, where only one type of arcs exists, representing the possible vehicle movements between the depot and the clients and between the clients. Therefore, all the arcs will be targeted by the selection

strategy, with the exception of the ones entering and leaving the depot. For other problem variants and depending on the network structure, it would be possible to limit the selection to only a subset of arcs as we did for the VCSP.

5.5.2 Data collection, features and labels

In this section, a few additional details specific to the VRPTW are given regarding the features described in Section 5.3.1. Starting with the resource consumptions on the arcs, two resources are considered as features : time and capacity (i.e., the demand). For each arc $(i, j) \in A, i \neq s, j \neq t$, the lower and upper bounds of the time windows of both client nodes i and j are collected. For the capacity resource, since the windows are the same for all clients (i.e., $[0, Q]$ where Q is the vehicle capacity), their bounds are not included in the features.
Finally, the labels are assigned as described in Section 5.3.2, i.e., the label 1 is assigned to the arcs that are part of the generated routes, and 0 for the others.

5.5.3 VRPTW instances

The VRPTW instances used are based on the Gehring & Homberger instances [73]. Three classes of instances are available : the R instances, standing for “Random", where the clients positions are randomly scattered, the C instances, standing for “Cluster", where clients are grouped together in clusters and thus have more chance to be served by the same vehicle, and finally the RC instances that are a combination of both. Moreover, for each of the three classes, there are two categories numbered 1 and 2 ($R1$, $C1$, $RC1$ and $R2$, $C2$, $RC2$). The difference between the two is in the width of the time windows : the category 1 instances tend to have larger time windows than those of category 2, and are therefore more difficult to solve.
Since we are only interested in solving the linear relaxation of the problem, and that most of the $R1$, $C1$ and $RC1$ instances we considered, i.e., 200 and 400 clients, are actually very easy to solve and take only a few seconds, we decided to only use the $R2$, $C2$ and $RC2$ instances. Some of these instances have been slightly modified, more precisely, the width of the time windows has been slightly reduced so that the computing time becomes reasonable and not very high. This is because some instances take hours to be solved, especially when no heuristics are used during the data collection phase.
Regarding the size of the instances, we used those of 200 and 400 clients. Half of the 200-client instances were used for training and testing in the ML phase (e.g., from the three classes), while the other half and the 400-client instances were used for testing the model obtained when integrated in the CG algorithm.

Table 5.5 Hyperparameters values used in the training phase.

Hyperparameter	**Value**
Bootstrap	True
Max depth	5
Max features	5
Min samples per leaf	50
Min samples per split	100
Number of trees	500
Class weights	Balanced

Table 5.6 Metrics of the model obtained for the VRPTW.

Metric	**Value**
Recall	93%
TNR	87%
Balanced Accuracy	90%

5.5.4 Computational results

As for the VCSP, this section will be divided into three parts. The first part is devoted to the ML phase where additional details about the features and data collection are covered as well as the results obtained. The second part presents the results of the model integration in the CG process, as well as a comparison with other algorithms. In the third part, we include some additional experiments that are worth mentioning. All the experiments were performed on a Linux machine with an i7-8700 CPU @ 3.20GHz and 64GB of RAM.

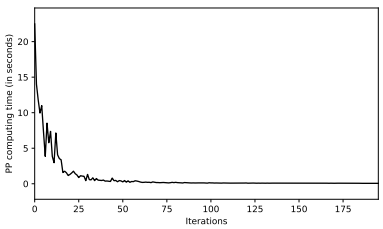
Machine learning

Because of the noticeable differences in the feature value range from one instance to another, especially between instances of different sizes, a preprocessing step is performed to scale and normalize the data of each instance individually. A random forest model is fit on the training data, and the hyperpameters are tuned through a cross validation approach. A neural network model was also tested, but without obtaining a significantly better accuracy, so we decided to retain the RF model as we did for the VCSP. The best hyperparameter values obtained are described in Table 5.5, whereas the results on the test set are reported in Table 5.6.
The hyperparameter values are slightly different from those used for the VCSP, more precisely the max_depth and max_features parameters. The class weights are used to balance the

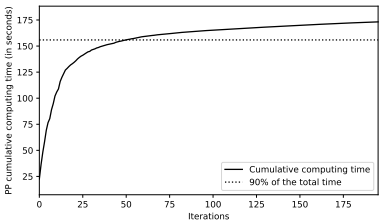
two classes as before, since there is only 14% and 9% of arcs with label 1 in the 200 and 400 instances, respectively. According to the results in Table 5.6, we notice a higher accuracy than the one obtained for the VCSP. The arcs to be selected are correctly predicted with 93% accuracy (i.e., Recall), whereas the TNR value is 87%, which means that the number of arcs selected by the model will not differ much from the number selected by the expert.
Note that the VCSP and the VRPTW are two completely different problems and they are not supposed to give comparable accuracies, especially considering that the arcs to be selected play different roles in each problem.

CG with arc selection

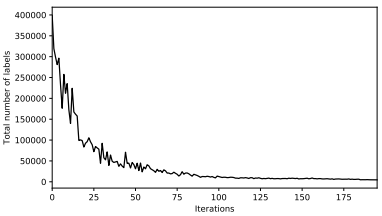
Now let us evaluate the ML model performance when integrated in the CG process. To do so, we have chosen to compare different algorithms described as follows :
Baseline-CG : This corresponds to the standard CG algorithm without arc selection where the complete network is used in all iterations.
ML-S : This is the CG algorithm with the arc selection activated as described in Algorithm 3, but with a small adjustment. In fact, for the VRPTW we noticed that towards the end of the optimization, the reduced network fails to generate columns for several consecutive iterations. Thus, the algorithm ends up solving the PP with both networks at multiple iterations (i.e., with G_r followed by G), which slows down the optimization. Instead, we decided to disable the use of the reduced network as soon as it fails to generate a column until the end of the optimization.
RedCost-S : This corresponds to the CG algorithm using a known heuristic pricing strategy that reduces the number of arcs in the network. It has been used by several researchers, and has shown to be very effective. For each node, excluding the depot nodes s and t, the heuristic consists in setting a minimum number N^{min} of incoming and outgoing arcs to keep. At each iteration, the arcs are sorted by their reduced cost, the N^{min} incoming and outgoing arcs with the least cost are kept, while the others are removed from the network. It is also possible to define multiple values $N_1^{min}, N_2^{min}, \ldots, \infty$ where $N_1^{min} < N_2^{min} < \cdots < \infty$. The PP tries first to generate columns with the parameter value N_1^{min}, if it fails it moves to the next parameter value, and so on. The values used in this experiment are $N_1^{min} = 10$, $N_2^{min} = 20$, $N_3^{min} = \infty$. Note that this algorithm performs the selection at each iteration while ML-S performs the selection only once before the beginning of the optimization.
The results obtained to compare the three algorithms are described in Table 5.7. For each algorithm, we report the same information as in Table 5.3. An additional column presents the time gain in percentage obtained by ML-S and RedCost-S in comparison to Baseline-CG. For ML-S, the number of iterations with the full network is indicated between parentheses.



(a) Computing time per iteration



(b) Cumulative computing time



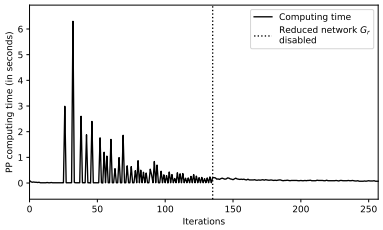
(c) Number of labels per iteration

FIGURE 5.4 PP computing time and number of labels with Baseline-CG.

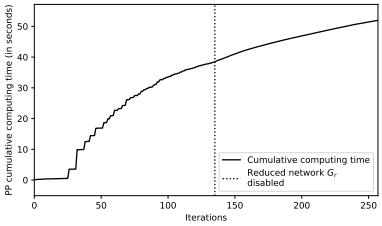
Starting with Baseline-CG, we can notice that most of the computing time is spent on the PP side. If we take a closer look at the time per iteration of one of the instances (R2_2_1 as an example), as shown in Figure 5.4a, the PP computing time decreases very rapidly, starting at 22 seconds at the first iteration and reaching less than 0.1 second after a hundred iterations. In fact, according to the cumulative time shown in Figure 5.4b, we can see that 90% of the computing time is spent in the first 50 iterations, which is just a quarter of the total number of iterations. The progressive decrease of computing time can be explained by looking at the number of labels created during the resolution of the PP as shown in Figure 5.4c. This shows that more labels are dominated as we progress in the optimization, and with less labels there is less processing and therefore less computing time.

Moving on to the ML-S results, we can notice a higher number of iterations compared to Baseline-CG. The number of iterations with the full network (i.e., in parentheses) is also quite high, representing more than 50% of the total number. Nevertheless, a significant reduction of computing time on the PP side is obtained. Since most of the iterations using the full network are performed in the second half of the optimization (i.e., when the use of the reduced network is disabled), they are not time consuming as previously shown in Figure 5.4. We can see a significant gain on the $R2$ instances. However, this gain decreases on the instances where the clients are more clustered (i.e., RC2 and C2) and also when moving from the 200 instances to the 400 ones. If we compare the increase in the number of iterations between ML-S and Baseline-CG for the 200-client instances, we can notice an increase of 32% for $R2$, 24% for $RC2$ and 73% for $C2$. It is true that the $C2$ instances seem to perform way more iterations than the other groups, and this may be one reason explaining the drop in performance. However, we see that the $RC2$ instances have a lower increase in the number of iterations than $R2$ and yet their results are less good.

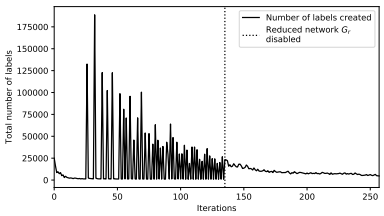
Let us consider that the optimization goes through two stages when using ML-S. The first stage starts from the beginning until the reduced network fails to generate any columns (i.e.,



(a) Computing time per iteration

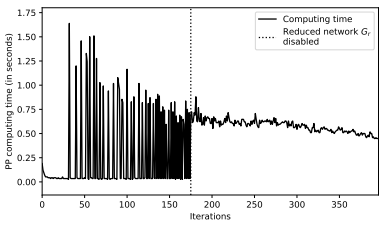


(b) Cumulative computing time

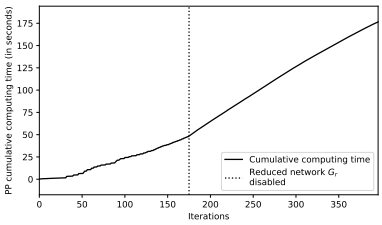


(c) Number of labels per iteration

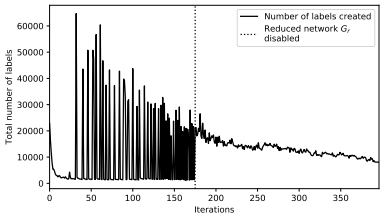
FIGURE 5.5 The PP computing time and number of labels with ML-S for instance R2_2_1.



(a) Computing time per iteration



(b) Cumulative computing time



(c) Number of labels per iteration

FIGURE 5.6 The PP computing time and number of labels with ML-S for instance C2_2_1.

the reduced network is thus disabled), and the second stage comes afterwards until the end of the optimization. By taking an $R2$ instance and comparing the computing time per iteration during the two stages, Figure 5.5a shows that there are multiple peaks of computing time during the first stage, which occur when we switch to the full network (i.e., when the reduced network does not generate enough columns). We can also see that the peaks heights decreases rapidly as we progress in the optimization. In the second stage (e.g., starting at the dashed line), the computing time is not too high, and it is also more stable since we are only using the full network. Figures 5.5b and 5.5c give some additional information, namely the cumulative time and the number of labels created, respectively. According to the cumulative time, we can see a step line in the first stage where the computing time increases significantly each time we switch to the full network, whereas in the second stage the line is almost linear. Notice that the total time spent on the first stage is larger than the second one. On the other hand, the number of labels follows exactly the same trend depicted in Figure 5.5a, which makes sense.

If we try to visualize the same data for a $C2$ instance in Figure 5.6 and compare it to the

previously seen data for an $R2$ instance (i.e., Figure 5.5), some differences can be noticed. We can observe that the computing time per iteration during the first stage decreases more slowly and is still quite high when we reach the second stage. The same thing can be noticed for the number of labels created. Looking at the cumulative time, we can see that the second stage takes more time than the first and the line is also steeper.

Finally, for the RedCost-S algorithm, we can notice that the results are significantly better than ML-S. A drop in performance is observed when dealing with the more clustered instances, but it is not as aggressive as ML-S. The number of iterations for the 200-client instances is larger or very close to the one of ML-S, but for the 400-client instances it is less on average.

Now the question is whether we can achieve better results than those obtained with RedCost-S. To answer that, we tried two additional strategies :

ML-RedCost-S : This corresponds to the combination of the two strategies ML-S and RedCost-S. In other words, the arc reduction according to the reduced cost is performed at each iteration on the two graphs G and G_r depending on which one is used.

ML-RedCost-S-2 : This corresponds to an ML model learned on the data generated using RedCost-S. This means that RedCost-S is considered the new expert. By going back to the first phase, i.e., data collection, the label 1 is assigned to the arcs that are part of the columns generated using RedCost-S, and 0 for the others. The training is performed on the new data in order to obtain a new model, which is then used to build the reduced network.

For both strategies, the same values of $N_1^{min} = 10$, $N_2^{min} = 20$, $N_3^{min} = \infty$ are used. The results comparing the two strategies to RedCost-S are reported in Table 5.8, providing the same information as in Table 5.7 except that the time gains are measured with respect to the time obtained by RedCost-S.

From the results, we can see that the gains obtained by the two new strategies are quite comparable. As before, a decrease in performance is noticed when dealing with the $RC2$ and $C2$ instances. On average, ML-RedCost-S-2 seems to be slightly better as it gives a few larger gains for some groups of instances. Overall, the results show that it is possible to get improvements on top of RedCost-S, up to 41% for the 200-client $R2$ instances and 28% for the 400-client ones, especially when the PP is much more time consuming than the RMP. However, it seems to be less effective on $RC2$ and $C2$ instances.

Additional experiments

Another attempt to deal with the inefficiency of the model on the $C2$ instances was conducted. On the ML side, one can think that the $C2$ instances are quite different from the other groups (i.e., RC and R), and that despite the overall accuracy of 90% obtained, the accuracy is

perhaps less for the $C2$ instances. It turns out that this is not the case, the metrics reported for the C2 instances alone are the following : 94% recall, 79% TNR and 87% accuracy. It is true that the accuracy is 3% below the average, but these are still solid results. Moreover, even with a new model that was only trained on the C2 instances, the results obtained when integrated to the CG algorithm were not much better.

5.6 Conclusion

In this paper, a new pricing heuristic based on ML was presented. The goal is to speed up the CG method for problems where the PP is a SPPRC or one of its variants defined on a network, and solved using a labeling algorithm. The method consists in reducing the network size, by keeping only the most promising arcs that have a high chance to be part of good columns. The ML model is trained on the data collected from previous executions through a supervised learning approach. The arcs selected by the trained model are used to build a new reduced network that is on average 15% to 25% the size of the original network. The new network is used at each iteration as long as it generates a satisfactory number of columns. If not, the full network is used instead, especially in the last iterations.
The approach is fairly general and can be used for different problems. We chose to demonstrate it on two well-known problems, the VCSP and VRPTW. For the VCSP, the selection was limited to the arcs representing walking movements, since they represent more than 95% of the arcs in the network. Whereas for the VRPTW, all the arcs were targeted by the selection. The results showed reductions in computing time of up to 27% for the VCSP, and 40% for the VRPTW. The resulting ML models have also shown the ability to generalize to new instances not encountered during the training phase. However, the ML model had some limitations when dealing with the C instances of the VRPTW.

Acknowledgment. We thank Giro Inc. and the Natural Sciences and Engineering Research Council of Canada for their financial support through the grant CRDPJ 520349-17. We are grateful to two anonymous referees for their careful reading and useful suggestions.

TABLE 5.7 VRPTW results.

Size	Instance	Baseline-CG				ML-S					RedCost-S				
		#Itr	Time (s)			#Itr	Time (s)			Gain	#Itr	Time (s)			Gain
			PP	RMP	Total		PP	RMP	Total			PP	RMP	Total	
200 clients	**R2_2_1**	197	173	8	181	259 (165)	52	8	54	**68%**	439	23	12	34	**81%**
	R2_2_2	159	312	5	317	211 (150)	99	6	105	**67%**	179	25	5	29	**91%**
	R2_2_3	198	128	7	135	301 (233)	36	8	44	**68%**	441	20	8	28	**79%**
	R2_2_4	114	210	2	212	152 (101)	69	3	72	**66%**	152	34	2	37	**83%**
	R2_2_5	239	238	10	248	275 (166)	44	9	53	**79%**	367	22	11	33	**87%**
	Average	**181**	**216**	**6**	**233**	**240 (163)**	**59**	**7**	**65**	**70%**	**316**	**25**	**7**	**32**	**84%**
	RC2_2_1	204	203	5	208	243 (172)	123	5	128	**38%**	374	32	6	38	**82%**
	RC2_2_2	119	138	3	141	156 (107)	49	3	52	**63%**	218	17	3	20	**86%**
	RC2_2_3	243	211	7	218	314 (214)	82	9	91	**59%**	852	40	11	51	**77%**
	RC2_2_4	198	189	7	196	239 (166)	150	7	157	**20%**	609	34	13	47	**76%**
	RC2_2_5	230	295	6	301	278 (192)	112	6	118	**61%**	313	32	6	39	**87%**
	Average	**199**	**207**	**6**	**213**	**246 (170)**	**103**	**6**	**109**	**48%**	**473**	**31**	**8**	**39**	**81%**
	C2_2_1	288	177	8	185	398 (268)	143	10	153	**18%**	760	72	11	83	**55%**
	C2_2_2	149	353	5	358	195 (126)	255	5	260	**27%**	316	121	6	126	**65%**
	C2_2_3	211	190	6	196	359 (176)	146	11	157	**20%**	272	54	7	61	**69%**
	C2_2_4	164	267	6	273	351 (163)	230	9	239	**12%**	181	65	5	70	**74%**
	C2_2_5	158	469	6	475	420 (161)	440	11	451	**5%**	237	140	6	146	**69%**
	Average	**194**	**291**	**6**	**297**	**345 (178)**	**243**	**9**	**252**	**16%**	**353**	**90**	**7**	**97**	**66%**
400 clients	**R2_4_1**	397	643	143	786	447 (239)	154	104	258	**67%**	373	66	119	185	**76%**
	R2_4_2	248	318	85	403	330 (165)	90	76	166	**59%**	289	45	84	129	**68%**
	R2_4_3	202	1058	42	1100	252 (188)	333	47	380	**65%**	235	97	37	134	**88%**
	R2_4_4	149	734	21	755	194 (112)	203	23	226	**70%**	239	142	20	162	**79%**
	R2_4_5	355	316	131	447	439 (244)	124	107	231	**48%**	361	51	127	178	**60%**
	R2_4_6	240	293	76	369	331 (162)	75	67	142	**62%**	277	44	78	122	**67%**
	R2_4_7	245	629	47	676	303 (179)	227	50	277	**59%**	218	81	44	125	**82%**
	R2_4_8	187	632	25	657	203 (116)	243	27	270	**59%**	242	144	23	167	**75%**
	R2_4_9	334	870	138	1008	472 (253)	277	111	388	**62%**	322	101	116	217	**78%**
	R2_4_10	308	988	113	1101	461 (257)	521	102	623	**43%**	380	154	108	262	**76%**
	Average	**267**	**648**	**82**	**730**	**343 (191)**	**255**	**71**	**296**	**59%**	**294**	**93**	**76**	**168**	**75%**
	RC2_4_1	392	281	123	404	555 (288)	144	116	260	**36%**	390	45	102	147	**64%**
	RC2_4_2	301	282	65	347	434 (216)	108	70	178	**49%**	295	37	67	104	**70%**
	RC2_4_3	202	1224	29	1253	293 (151)	288	33	321	**74%**	301	108	27	135	**89%**
	RC2_4_4	155	361	17	378	231 (125)	185	22	207	**45%**	183	81	16	97	**74%**
	RC2_4_5	797	337	198	535	1301 (848)	271	199	470	**12%**	1164	84	186	270	**50%**
	RC2_4_6	370	622	98	720	543 (295)	229	101	330	**54%**	434	98	118	216	**70%**
	RC2_4_7	340	615	87	702	510 (235)	264	92	356	**49%**	415	100	98	198	**72%**
	RC2_4_8	373	629	74	703	478 (254)	245	73	318	**55%**	435	90	72	162	**77%**
	RC2_4_9	305	460	83	543	417 (263)	188	91	279	**49%**	371	83	91	174	**68%**
	RC2_4_10	411	219	101	320	458 (347)	128	89	217	**32%**	438	40	91	131	**59%**
	Average	**365**	**503**	**88**	**591**	**522 (302)**	**205**	**89**	**294**	**46%**	**443**	**77**	**87**	**163**	**69%**
	C2_4_1	599	373	136	509	1125 (660)	424	178	602	**-18%**	966	144	183	327	**36%**
	C2_4_2	270	214	74	288	484 (237)	197	78	275	**5%**	445	71	88	159	**45%**
	C2_4_3	254	543	50	593	342 (180)	496	58	554	**7%**	484	176	55	231	**61%**
	C2_4_4	153	574	26	600	408 (121)	588	52	640	**-7%**	332	267	30	297	**51%**
	C2_4_5	261	632	62	694	714 (222)	637	93	730	**-5%**	417	194	63	257	**63%**
	Average	**307**	**467**	**69**	**536**	**614 (284)**	**468**	**91**	**560**	**-4%**	**528**	**170**	**83**	**254**	**51%**

TABLE 5.8 Additional VRPTW results.

Size	Instance	RedCost-S				ML-RedCost-S					ML-RedCost-S-2				
		#Itr	Time (s)			#Itr	Time (s)			Gain	#Itr	Time (s)			Gain
			PP	RMP	Total		PP	RMP	Total			PP	RMP	Total	
200 clients	**R2_2_1**	439	23	12	34	578 (403)	13	12	24	**31%**	420 (235)	8	10	18	**48%**
	R2_2_2	179	25	5	29	309 (127)	13	7	21	**30%**	343 (158)	16	8	24	**17%**
	R2_2_3	441	20	8	28	406 (255)	8	9	17	**38%**	378 (219)	8	8	16	**44%**
	R2_2_4	152	34	2	37	217 (98)	8	3	11	**69%**	245 (113)	9	4	12	**66%**
	R2_2_5	367	22	11	33	430 (244)	11	12	23	**32%**	444 (245)	11	12	23	**32%**
	Average	**316**	**25**	**7**	**32**	**388 (225)**	**11**	**8**	**19**	**40%**	**366 (194)**	**10**	**8**	**18**	**41%**
	RC2_2_1	372	32	6	38	586 (378)	18	10	28	**28%**	568 (376)	17	9	26	**32%**
	RC2_2_2	218	17	3	20	332 (179)	12	5	17	**15%**	282 (142)	11	5	15	**25%**
	RC2_2_3	852	40	11	51	912 (695)	28	11	39	**23%**	670 (446)	22	10	32	**38%**
	RC2_2_4	609	34	13	47	675 (476)	18	14	33	**30%**	531 (352)	17	13	29	**37%**
	RC2_2_5	313	32	6	39	451 (226)	15	7	23	**41%**	497 (270)	20	9	29	**25%**
	Average	**473**	**31**	**8**	**39**	**591 (391)**	**18**	**10**	**28**	**28%**	**510 (317)**	**17**	**9**	**26**	**31%**
	C2_2_1	760	72	11	83	924 (552)	54	15	69	**17%**	1020 (676)	62	15	77	**6%**
	C2_2_2	316	121	6	126	416 (242)	76	8	84	**34%**	385 (210)	71	8	79	**38%**
	C2_2_3	272	54	7	61	505 (208)	49	11	60	**2%**	489 (190)	51	11	62	**-1%**
	C2_2_4	181	65	5	70	442 (113)	61	11	72	**-3%**	464 (122)	65	12	77	**-10%**
	C2_2_5	237	140	6	146	501 (159)	127	12	139	**5%**	479 (174)	121	11	132	**10%**
	Average	**353**	**90**	**7**	**97**	**558 (255)**	**73**	**11**	**84**	**11%**	**567 (274)**	**74**	**11**	**85**	**10%**
400 clients	**R2_4_1**	373	66	119	185	647 (180)	39	119	158	**15%**	678 (230)	41	122	163	**12%**
	R2_4_2	289	45	84	129	436 (127)	24	81	105	**19%**	429 (117)	24	81	105	**19%**
	R2_4_3	235	97	37	134	392 (111)	40	54	94	**30%**	440 (157)	44	55	99	**0%**
	R2_4_4	239	142	20	162	297 (106)	40	29	69	**57%**	314 (134)	48	29	77	**52%**
	R2_4_5	361	51	127	178	619 (165)	32	114	146	**18%**	573 (159)	30	114	144	**19%**
	R2_4_6	277	44	78	122	523 (196)	32	94	126	**-3%**	460 (139)	25	81	106	**13%**
	R2_4_7	218	81	44	125	390 (114)	38	60	98	**22%**	384 (116)	31	53	84	**33%**
	R2_4_8	242	144	23	167	328 (131)	60	33	93	**44%**	309 (119)	45	31	76	**54%**
	R2_4_9	322	101	116	217	615 (168)	56	131	187	**14%**	632 (188)	50	121	171	**21%**
	R2_4_10	380	154	108	262	691 (230)	76	119	195	**26%**	719 (201)	64	116	180	**31%**
	Average	**294**	**93**	**76**	**168**	**494 (153)**	**44**	**83**	**127**	**24%**	**494 (156)**	**40**	**80**	**121**	**28%**
	RC2_4_1	390	45	102	147	762 (232)	42	124	166	**-13%**	801 (310)	50	132	182	**-24%**
	RC2_4_2	295	37	67	104	630 (210)	37	76	113	**-9%**	651 (309)	43	89	132	**-27%**
	RC2_4_3	301	108	27	135	446 (188)	52	35	87	**36%**	486 (206)	53	36	89	**34%**
	RC2_4_4	183	81	16	97	263 (76)	29	23	52	**46%**	278 (86)	31	23	54	**44%**
	RC2_4_5	1164	84	186	270	1600 (882)	83	196	279	**-3%**	1810 (1084)	101	251	352	**-30%**
	RC2_4_6	434	98	118	216	773 (244)	62	130	192	**11%**	867 (314)	62	122	184	**15%**
	RC2_4_7	415	100	98	198	792 (348)	79	118	197	**1%**	674 (274)	60	119	179	**10%**
	RC2_4_8	435	90	72	162	834 (357)	78	84	162	**0%**	669 (240)	56	74	130	**20%**
	RC2_4_9	371	83	91	174	723 (304)	67	116	183	**-5%**	952 (561)	111	108	219	**-26%**
	RC2_4_10	438	40	91	131	679 (274)	39	118	157	**-20%**	689 (266)	35	91	126	**4%**
	Average	**443**	**77**	**87**	**163**	**750 (312)**	**57**	**102**	**159**	**4%**	**788 (365)**	**60**	**105**	**165**	**2%**
	C2_4_1	966	144	183	327	1692 (669)	162	190	352	**-8%**	1785 (768)	170	186	356	**-9%**
	C2_4_2	445	71	88	159	867 (438)	99	117	216	**-36%**	726 (239)	68	110	178	**-12%**
	C2_4_3	484	176	55	231	741 (333)	215	78	293	**-27%**	913 (463)	210	87	297	**-29%**
	C2_4_4	332	267	30	297	522 (249)	327	44	371	**-25%**	536 (287)	290	47	337	**-13%**
	C2_4_5	417	194	63	257	887 (266)	248	101	349	**-36%**	895 (278)	222	101	323	**-26%**
	Average	**529**	**170**	**84**	**254**	**942 (391)**	**210**	**106**	**316**	**-26%**	**971 (407)**	**192**	**106**	**298**	**-17%**

CHAPITRE 6 ARTICLE 3: LEARNING TO REPEATEDLY SOLVE ROUTING PROBLEMS

Cet article a été soumis à la revue *Operations Research* le 03 Novembre 2022.
Auteurs : Mouad Morabit, Guy Desaulniers, Andrea Lodi

Abstract

In the last years, there has been a great interest in machine-learning-based heuristics for solving NP-hard combinatorial optimization problems. The developed methods have shown potential on many optimization problems. In this paper, we present a learned heuristic for the reoptimization of a problem after a minor change in its data. We focus on the case of the capacited vehicle routing problem with static clients (i.e., same client locations) and changed demands. Given the edges of an original solution, the goal is to predict and fix the ones that have a high chance of remaining in an optimal solution after a change of client demands. This partial prediction of the solution reduces the complexity of the problem and speeds up its resolution, while yielding a good quality solution. The proposed approach results in solutions with an optimality gap ranging from 0% to 1.7% on different benchmark instances within a reasonable computing time.

Keywords : heuristics, reoptimization, routing, machine learning.

6.1 Introduction

In the recent years, the idea of integrating machine learning (ML) and combinatorial optimization (CO) has been greatly explored. Many CO problems are considered NP-hard and there are generally two approaches to solve them. The *exact* methods that are guaranteed to obtain an optimal solution but can be computationally very expensive for large instances, and the *heuristic* methods that trade off the optimality of the solution for a reasonable computing time. The idea of leveraging ML for the development of new heuristics has shown potential in many CO problems such as traveling salesman problem (TSP), capacited vehicle routing problem (CVRP), etc. It is true that most of these learned heuristics do not outperform highly optimized and specialized CO algorithms, especially for problems that have been extensively studied in the literature. Nevertheless, the ideas behind them provide a certain flexibility for adjustments and applications to other problems for which no good heuristic exists, or they can be integrated in already existing algorithms to speed them up.

In this paper, we focus on CO applications in which a problem is repeatedly solved, e.g., daily or hourly, or even within a shorter interval, by changing neither its structure (e.g., its constraints) nor even its size (i.e., its variables), but only the data that define each instance solved in the specific time interval. This is the case of applications in which the infrastructure whose operations must be optimized does not change, for example a fleet of vehicles that deliver goods or the power plants producing energy, but the demand of goods or energy changes. And it is also the case of real-time changes to the data due to disruptions in the infrastructure, for example arcs disappearing from a network (i.e., their capacity going to 0) due to accidents.

Applications of this type might be difficult to solve in reasonable amount of time, or, more precisely, each instance in isolation might require a significant computational effort even if the solution method has been designed after intensively studying the characteristics of the CO problem. This is the theoretical consequence of NP-hardness, and, on the practical side, it is due to the fact that the solution methods are largely designed to be agnostic to the data.

The goal of this paper is to propose a learned heuristic allowing a fast reoptimization of a CO problem after a slight modification of its data. In other words, we put ourselves in a way more restrictive context with respect to the use of ML for CO problems : we do not want to leverage ML to devise a heuristic that produces good feasible solutions for all, say, TSP instances within the same distribution. We are settling for a lesser objective, i.e., that of learning what can be (more or less with high probability) safely left unchanged in the solution of a reference instance of a CO problem when the data of the instance are perturbed. Indeed, the intuition is that given an instance and its solution S_1, if the instance is reoptimized after a slight change in its data, the new solution S_2 will have a significant overlap with S_1, while only some (minor) parts of the solutions will be different. Therefore, instead of reoptimizing from scratch, the goal is to predict the parts of the solution that have a high probability of remaining the same. The corresponding variables can be fixed, thus reducing the search space and accelerating the resolution of the problem, almost independently of the solution method applied.

To provide a concrete example, let us consider the CVRP, which is the problem where we will apply the method proposed in this paper. The goal of the CVRP is to construct vehicle routes in order to serve geographically-dispersed clients while minimizing the travel costs and respecting vehicle capacity. Let us take the example of a delivery company that solves CVRPs on a daily basis. For a given day, the company observes that the clients are the same as in the instance solved the previous day (i.e., same client locations) and that only some clients have a different demand. After the optimization of the problem, a similarity between the solutions is noticed. Given the optimal solution (or a heuristic one) already in hand and

the new demands, the objective would be to predict and fix the sequences of edges that have a high probability to remain the same. In case of a graph-structured problem like the CVRP, it is also possible to reduce the size of the network by aggregating the nodes/edges of the fixed sequences, therefore accelerating the resolution of the problem and reducing its complexity. Note that the predictions obtained by the learned model are not necessarily 100% accurate, misclassifications may occur and thus affect the quality of the solution. The goal is to find a good compromise between the quality of the solution (i.e., optimality gap) and the computing time.

As observed, one can think of several CO problems where slight modifications to the problem data lead to similar solutions after reoptimization, especially for problems that are solved repeatedly and for which a data set is already available (e.g., the unit commitment problem in which the power plants producing energy are always the same but the demand changes daily or hourly). The learned model will try to partially predict an optimal solution. In this paper, we will only consider the case of the CVRP with changing demands and fixed customers, but the method offers some flexibility and has the potential to be applied to other problems or to be integrated into existing algorithms.

The remainder of the paper is organized as follows. In Section 2, we present some recent work on using ML for solving CO problems or accelerating their solution process. Section 3 is devoted to presenting the CVRP, with a focus on the methods we consider to solve it. In Section 4, we cover all the details of the method we propose. Section 5 reports our computational results. Finally, conclusions are drawn in Section 6.

6.2 Related work

In the literature, several heuristics incorporating ML models for solving NP-hard CO problems were explored (see, the recent surveys [43, 44, 78]). The proposed learning methods mostly fall into one of two categories. In the first category, *supervised learning* methods (examples of the imitation learning paradigm) are algorithms that learn from data and try to mimic an expert. The data is given to the learner as a pair of features and expected outputs (or labels) and the learner tries to find patterns in the data while optimizing a performance measure. Generally, the aim of this approach is to replace known expensive computations by fast approximations (e.g., for our case, the expensive computation corresponds to reoptimizing the problem from scratch). In the second category, we find the *reinforcement learning* (RL) algorithms that apply a “learning-by-experience" paradigm. Instead of giving the learner the data on which to learn, RL algorithms explore the decision space by interacting with its environment in order to achieve a certain goal. In response to a decision (i.e., an action), the learner receives a real-valued reward. The goal is to find the best decisions to make at

each state while maximizing the expected rewards. This learning approach has the advantage of not requiring any data and has generally shown a better generalization, i.e., the ability of continuing to be effective when the problems change for example in size or even better in data distribution.

Several studies have tried to tackle the TSP, the most classical CO problem where the goal is to visit a set of nodes exactly once with a single vehicle while minimizing the travel distance. For the supervised approaches, we mention the work of Joshi et *al.* [54], where a learned heuristic method is presented. The method is based on a graph convolutional network model that takes the entire graph as an input and outputs an adjacency matrix with associated edge probabilities, which are then used to build a valid tour using a beam search algorithm. The authors report good results on fixed size instances. However, a very poor generalization is noticed when testing the models on instances of different size. Instead, the methods based on RL have shown more potential. The idea of using RL to solve CO problems was explored in [55] with an application to the TSP and the knapsack problem. Further studies were then conducted, such as Kool et *al.* [56], where the authors present an encoder-decoder model based on the attention mechanism [79]. In a similar line, Nazari et *al.* [80] present a framework focusing on solving the CVRP, which can be considered as a generalization of the TSP for the case with multiple vehicles. The results reported by the authors show a better performance when compared to the OR-Tools solver and other heuristic algorithms. Other researchers contributed to the methods for solving the broader class of graph-structured CO problems. Dai et *al.* [81] initiated the idea of learning on graphs, and the works of Li et *al.* [82] and Manchanda et *al.* [83] enhanced further the scalability to larger graphs. These methods have been applied to various NP-hard graph problems such as the minimum vertex cover, maximum clique, influence maximization problem, etc.

Unlike the previous works that seek to build an end-to-end solution to the different problems, other methods focus on using ML to guide and accelerate the solution process. Since we pay special attention to mixed-integer programs (MIP), it is worth mentioning the various projects involving ML in the context of branch and bound [49–51, 84], many of which seek to learn a branching policy imitating the strong branching strategy. Another potential use of ML is embedding a learned model in MIP solvers in order to decide if a decomposition of the problem is beneficial or not [85], or if it is favorable to linearize the quadratic part of a mixed-integer quadratic program [86].

Other learning-based methods have proven to be very effective on problems that are solved repeatedly, especially when the input data changes only slightly, which is the key idea of our project. These methods can exploit existing data from previous solutions in order to speed up the resolution of similar unseen instances. Xavier et *al.* [58] exploit the idea and apply it on the

security-constrained unit commitment, a problem occuring in power systems and electricity markets. The authors report high speedups on computing time, up to 10 times faster than solving the problem from scratch, and without a noticeable loss in solution quality. Along the same lines, Lodi et *al.* [59] consider an application to the facility location problem. They seek to estimate the proportion of a solution that has a high probability of remaining unchanged after a perturbation. An additional constraint is added to the original formulation according to the predictions obtained by a regression model. Both Xavier et *al.* [58] and Lodi et *al.* [59] leverage ML for speeding up the resolution of repeatedly solved problems and do not seek to build an end-to-end solution, which is similar to our case. As opposed to Xavier et *al.* [58], our method is related to that of Lodi et *al.* [59] in the sense that both assume a reference solution to which changes (perturbations) are applied. However, the main difference lies on the fact that our approach revolves around fixing parts of the reference solution (i.e., edges) instead of estimating a bound on the number of changes without specifying which variables to set. Other than that, the CVRP remains a quite different problem and the approaches developed in this paper have the potential to be extended to other CVRP variants and also to other routing problems.

6.3 The capacited vehicle routing problem

The CVRP is a CO problem that has been studied for many years, resulting in several exact and heuristic methods to solve it. It is classified as an NP-hard problem and remains a difficult problem to solve to optimality even with just a few hundred clients. Given a fleet of vehicles assigned to a depot, the problem consists in determining a set of possible routes (i.e., one route per vehicle used) to deliver goods to a set of dispersed clients while minimizing the travel costs. A route starts from the depot and visits a sequence of clients before returning back, and is considered feasible if the total amount of goods delivered does not exceed the vehicle capacity Q.

For solving this problem, we consider the algorithms based on column generation (CG - [1]), which is an exact iterative method for solving large linear programs. To ensure the obtention of integer solutions, CG is often embedded in a branch-and-bound framework where the linear relaxation of the problem is solved at each node using CG. In this case the method is referred to as *branch-and-price* (B&P) and it is considered the state-of-the-art exact method for solving the CVRP. But due to the complexity of the problem, solving large instances to optimality can be computationally expensive. Therefore, several heuristics have been proposed in the literature.

Note that our approach does not necessarily require learning from optimal solutions. Since we want to apply it on instances of a reasonable size (i.e., 100 clients and more), solving the

instances to optimality in order to collect data can be time consuming. Therefore, we chose to use a heuristic in the data collection phase, but during the evaluation phase, we exploit an exact B&P algorithm. In the next sections, we present the problem formulation along with both the exact and the heuristic algorithms used.

6.3.1 CVRP formulation

In this section, we formulate the CVRP as a set partitioning problem. Let C be the set of clients to be serviced, Ω the set of all feasible routes and c_r the cost of a route $r \in \Omega$. We define a_i^r as a binary parameter equal to 1 if client $i \in C$ is serviced by route $r \in \Omega$ and 0 otherwise. Let θ_r be a binary decision variable equal to 1 if route r is part of the solution and 0 otherwise. The problem can therefore be formulated as follows :

$$(P) \quad \min_{\theta} \quad \sum_{r \in \Omega} c_r \theta_r \tag{6.1}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} a_i^r \theta_r = 1, \quad \forall i \in C, \tag{6.2}$$

$$\theta_r \in \{0, 1\}, \quad \forall r \in \Omega, \tag{6.3}$$

where the objective (6.1) minimizes the total cost of the routes. Constraints (6.2) ensure that each client is visited exactly once and constraints (6.3) are the binary requirements on the decision variables θ_r.

One can notice that for large instances, the size of the route set $|\Omega|$ becomes prohibitively large and it would not be possible to enumerate all the variables of the problem. This is why a CG-based algorithm is used. The goal is to start with a subset of variables and generate potentially improving columns when necessary.

6.3.2 Exact branch-and-price algorithm

B&P algorithms [4] are considered state-of-the-art exact algorithms for solving a variety of optimization problems (e.g., routing, scheduling, ...). B&P is based on the branch-and-bound method in which the linear relaxation at each node is solved using CG. The CG process is iterative and consists in alternating between the resolution of a restricted version of the original linear relaxation, called a restricted master problem (RMP), and a pricing problem (PP). The goal of the PP is to find new improving columns of negative reduced cost (for a minimization problem) that can be added to the RMP. The CG process stops when no such columns are found. A branching then occurs and the branch-and-bound tree exploration continues. Optionally, cutting planes can also be added to strengthen the relaxation at each

node, resulting on what is called a branch-cut-and-price method.

The restricted master problem

The RMP corresponds to the linear relaxation of the formulation (6.1)-(6.3) but limited to only a subset $\mathcal{R} \subset \Omega$ of the variables. It is solved at each CG iteration, yielding a pair of primal and dual solutions (θ, π). The dual values $(\pi_i)_{i \in C}$ associated with the constraints (6.2) are then used to find routes $r \in \Omega \backslash \mathcal{R}$ of negative reduced cost by solving the PP. If none exist, the CG process stops and the solution to the current RMP is thus optimal for the whole linear relaxation. Otherwise, the routes are added to the RMP (i.e., to the subset $\mathcal{R}$) which is then reoptimized.

The pricing problem

The PP can be defined as $min_{r \in \Omega} \{c_r - \sum_{i \in C} a_i^r \pi_i\}$. For many applications, especially routing and scheduling problems, this problem can be modeled as an elementary shortest path problem with resource constraints (ESPPRC [18]) where the goal is to find the least cost path between the source and destination nodes while visiting the nodes at most once (i.e., elementarity requirements) and respecting the resource constraints. For the CVRP, the only resource is the load, and a path is considered feasible if the load does not exceed the vehicle capacity. This problem can be defined over a graph $G = (V, A)$, where V is the set of nodes representing the clients, in addition to the depot nodes s and t, i.e., the source and destination nodes, respectively. The set A represents the arcs, where each arc has an associated cost $c_{ij}, (i, j) \in A$. Hence, the cost of a path in G is given by the sum of the costs c_{ij} of its arcs. In order to take into account the dual values obtained by the RMP, at each iteration and for each arc $(i, j) \in A$, a modified cost $\bar{c}_{ij} = c_{ij} - \pi_i$ is used instead, where π_i are the duals associated with constraints (6.2) and $\pi_s = 0$. This guarantees that the cost of a feasible route in the network is equal to its reduced cost :

$$\bar{c}_r = c_r - \sum_{i \in C} a_i^r \pi_i = \sum_{(i,j) \in A} c_{ij} b_{ij}^r - \sum_{i \in C} a_i^r \pi_i = \sum_{(i,j) \in A} (c_{ij} - \pi_i) b_{ij}^r = \sum_{(i,j) \in V} \bar{c}_{ij} b_{ij}^r \tag{6.4}$$

where b_{ij}^r is equal to 1 if arc $(i, j) \in A$ is traversed in route $r \in \Omega$, 0 otherwise.
The ESPPRC is an NP-hard problem, which is mainly due to the client elementarity requirements since negative cost cycles can exist when using the modified costs $\bar{c}_{ij}$. Relaxing this constraint (i.e., allowing the generation of paths with cycles) leads to the SPPRC, which is an easier problem that can be solved in pseudo-polynomial time but yields a lower bound of inferior quality if solved as the PP. Other alternatives based on relaxations have been proposed in the literature, e.g., SPPRC-k-cyc [22] and ng-routes [24].

The success of B&P methods for solving the CVRP and other variants is in good part due to the efficient methods for solving the PP, mainly using dynamic programming. A labeling algorithm is commonly used where a label corresponds to a partial path in G (not to confound with the notion of label in ML). The algorithm starts with an initial label representing the trivial path containing only the source node s, it then gets extended forward along the outgoing arcs until reaching the destination node t. A new label is created at each extension if it yields a feasible path. At the end of the algorithm, the labels at the destination node t representing negative reduced cost routes are used to build the new columns that are added to the RMP. Generally, several routes are added at once, which is known to speed up the solution process and to reduce the number of CG iterations.
The CVRP remains one of the most studied CO problems in the literature. An efficient B&P algorithm can be quite sophisticated and may contain several components. Discussing all the details is beyond the scope of this paper and we refer the interested reader to the survey of Costa et *al.* [27] for an in-depth overview of the methods.

Branch-and-price implementation

In this work, we consider using *VRPSolver* [28], which is a generic implementation of an exact branch-cut-and-price method for VRP problems. The advantage of using VRPSolver lies in the fact that it combines several algorithms and acceleration techniques introduced by several authors, e.g., ng-routes, path enumeration, bi-directional labeling, stabilization, etc. Implementing these techniques from scratch would be otherwise very time consuming. The authors report excellent performance on several benchmark instances of various CVRP variants (e.g., with time windows, heterogeneous fleet, multiple depots, pickups and deliveries, etc). For more information about the implementation and the algorithms included in VRPSolver, the reader is referred to [28].

6.3.3 Heuristic algorithm

As previously mentioned, the CVRP remains a difficult problem to solve to optimality and can be time consuming when working with large instances. For our method, we need to solve several instances to collect enough data for the training phase, but solving them to optimality is not required (although recommended), since it is possible to obtain very high-quality solutions using specialized CVRP heuristics. For this reason, we chose to use a recent heuristic called FILO [87], which is a short term for *Fast Iterated Localized Search Optimization*.
The method is based on the iterated local search paradigm and is specifically designed to solve large-scale instances of the CVRP. The algorithm starts by constructing an initial feasible solution using an adaptation of the savings algorithm by Clarke and Wright [88]. It is followed

by an optional step that aims at reducing the number of vehicles used in the initial solution, if the latter is larger than a computed estimate (using a bin-packing greedy algorithm). The algorithm then proceeds to the core optimization step, which is based on a sequence of ruin and recreate steps. The *ruin* step removes a certain number of vertices by means of a random walk of a given length (the ruin intensity can be controlled by adapting the random walk length). Then, the *recreate* step tries to reinsert the removed vertices while trying to improve upon the best solution found. This is achieved by means of a number of local search operators targetting the vertices involved in the disruptive effects of the ruin step. The algorithm tries to keep a good balance between intensification and diversification, i.e., the local search can be concentrated on the parts of the solution that have not seen any improvement after several attempts, and at the same time, a continuous diversification is intended in order to escape from local optima. The algorithm stops after a determined number of iterations and the best solution found is returned.

The performance of the FILO algorithm has been compared by the authors to other state-of-the-art heuristics and has proven to be highly competitive on the X instances introduced by Uchoa et *al.* [89], which are also the benchmark instances used by our method. Note that the purpose of this section is not to present or compare the different heuristics that exist for solving the CVRP, but simply to present the heuristic that we used to obtain high-quality solutions during the data collection phase (i.e., with optimality gaps of less than $0,1\%$ on average). The method has also the advantage of having an open-source implementation that is freely accessible.

6.4 Methodology

The goal of this project is to accelerate the reoptimization of repeatedly solved CO problems for which there is only a slight change in the problem data. Let $\mathcal{P}_o$ and $\mathcal{S}_o$ be a problem instance and a computed good-quality solution, respectively. Furthermore, let $\mathcal{P}_m$ be a modified instance obtained by applying minor changes to $\mathcal{P}_o$ and for which no solution is known. Instead of optimizing $\mathcal{P}_m$ from scratch, the objective is to identify the parts of $\mathcal{S}_o$ that have a high probability of also being part of a solution to $\mathcal{P}_m$ denoted by $\mathcal{S}_m$.

In this paper, we focus on the CVRP where the locations of the clients are the same but the demands are slightly different. Because we assume that the fleet of vehicles is unlimited and the travel costs are symmetric, we consider for the rest of this paper an undirected graph and denote by $E(\mathcal{S}_o)$ the set of edges used in the known solution $\mathcal{S}_o$ to the original instance $\mathcal{P}_o$. Since the solution consists of vehicle routes, if we consider $E(\mathcal{S}_o)$ as the set of edges used in a solution $\mathcal{S}_o$ already in hand, the method aims at predicting the edges $e \in E(\mathcal{S}_o)$ that have a high chance to also be part of $\mathcal{S}_m$. By fixing these edges, a significant reduction of

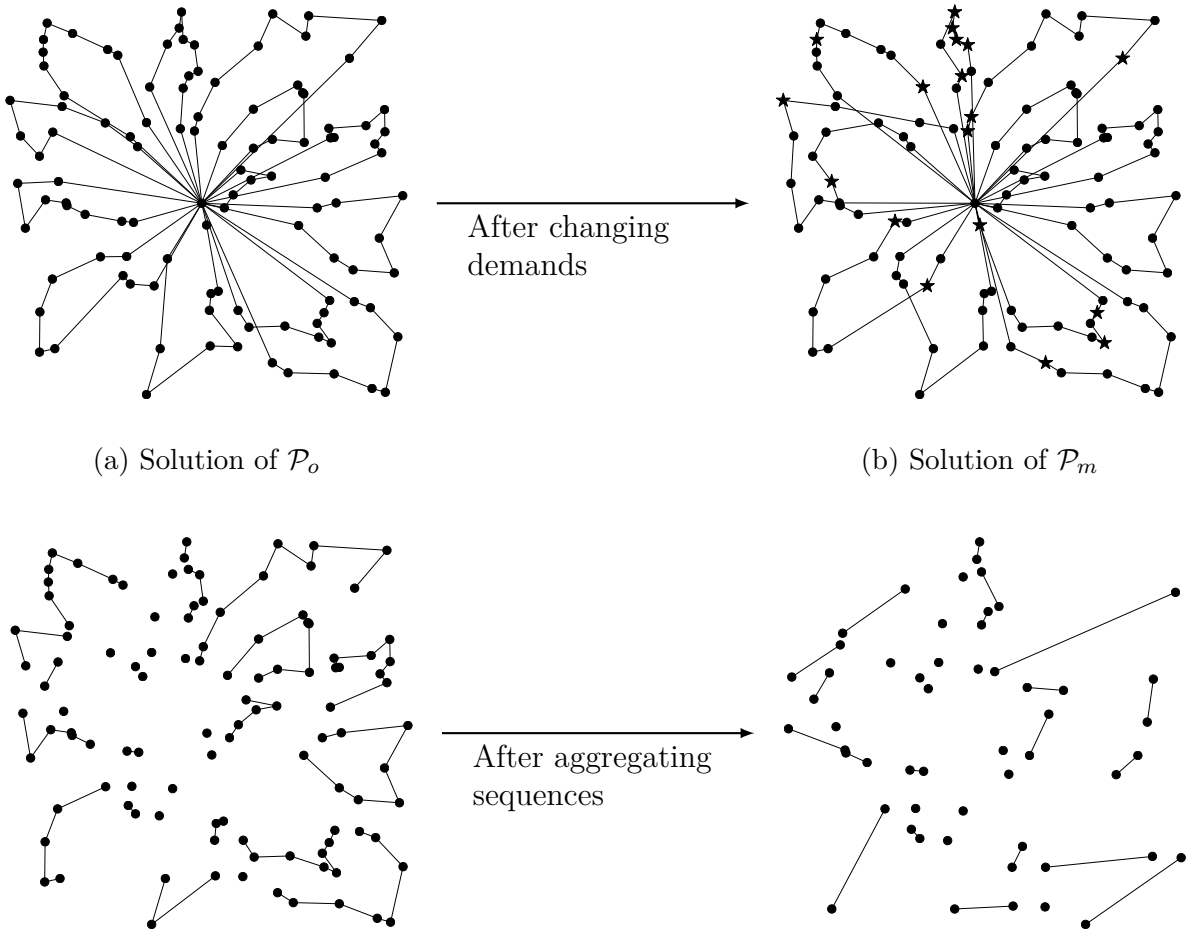
the problem complexity can be achieved, thus greatly reducing the computing time. Let us take the example illustrated in Figure 6.1, where we can observe the following : Figure (6.1a) represents a solution $\mathcal{S}_o$ to an original CVRP instance involving 110 nodes. The central node corresponds to the depot and the other nodes represent the clients. Figure (6.1b) shows a solution $\mathcal{S}_m$ to an instance $\mathcal{P}_m$ obtained by randomly changing the demands of 20% of the clients of $\mathcal{P}_o$ (the clients with a changed demand are marked with a star node). By comparing the two figures, one can notice a significant similarity between the two solutions. Assuming we do not have yet the solution $\mathcal{S}_m$, if we succeed in predicting and fixing the edges $e \in E(\mathcal{S}_o)$ that will be part of $\mathcal{S}_m$, a partial solution can be provided to the solver before starting the optimization. This is shown in Figure (6.1c), where the overlapping parts of the two solutions are highlighted, i.e., the edges $e \in E(\mathcal{S}_o) \cap E(\mathcal{S}_m)$. Furthermore, it is also possible to reduce the size of the network by aggregating the sequences comprised of two or more edges into a single one as shown in Figure (6.1d). As a result, the number of nodes is reduced from 110 to only 54.

On the other hand, since the predictions obtained by the ML model are not always accurate and errors may occur, fixing the wrong edges can affect the quality of the solution obtained. In fact, fixing many edges implies shorter computing times but more chances to fix the wrong ones. On the other hand, if the number of fixed edges is small, there is a greater chance of obtaining a better quality solution though with a higher computing time. By controlling the number of fixed edges (e.g., by tuning the hyperparameters of the model), it is possible to find the right compromise between the quality of the solution and the computing time.

6.4.1 Data collection

We chose to address this learning problem using a supervised learning approach. More precisely, a binary classification model is employed. The first step in the process is to collect enough data for the training. Given a tuple of original and modified instances and their solutions $(\mathcal{P}_o, \mathcal{S}_o, \mathcal{P}_m, \mathcal{S}_m)$, a labeled dataset $\mathcal{D} = \{\{\boldsymbol{x}_e, y_e\} | \forall e \in E(\mathcal{S}_o)\}$ is built where each entry represents an edge in the original solution, the vector $\boldsymbol{x}_e \in \mathbb{R}^n$ corresponds to the edge features (i.e., input), where n is the number of features and $y_e = \{0, 1\}$ is the desired output (i.e., label). One can notice that we are only interested in the edges of $\mathcal{S}_o$ and not all the edges of the graph.

The labels. Since we tackle this problem in a supervised manner, we need both solutions $\mathcal{S}_o$ and $\mathcal{S}_m$ to build the dataset, i.e., we need to give both the input and the desired output to the learner. The labels are assigned by simply checking the overlapping edges between the



(a) Solution of $\mathcal{P}_o$

(b) Solution of $\mathcal{P}_m$

(c) Edges to predict and fix

(d) Edges to fix after aggregating sequences

FIGURE 6.1 Overview of the different steps of the method.

solutions $\mathcal{S}_o$ and $\mathcal{S}_m$ as follows :

$$y_e = \begin{cases} 1 & \text{if } e \in E(\mathcal{S}_o) \cap E(\mathcal{S}_m) \\ 0 & \text{otherwise} \end{cases}, \quad e \in E(\mathcal{S}_o). \tag{6.5}$$

The features. The features $\boldsymbol{x}_e$ represent the characteristics of each edge $e = \langle i, j \rangle$ in the original solution. The extracted features are the following :

- The (x, y) coordinates of both nodes i and j ;
- The cost of edge c_e ;
- The old and new demands of nodes i and j (the demand is set to 0 for the depot nodes s and t) ;
- The distance between the depot and nodes i and j ;
- A boolean value indicating whether the edge is a depot edge, i.e., equal to 1 if either i or j is a depot node ;
- A boolean value indicating whether the client i or j, or both, have a changed demand ;
- The rank of i with respect to j (and vice versa) according to the neighbor distances, e.g., if j is the nearest neighbor to i, then its rank is 1, if j is the second closest neighbor to i then its rank is 2, and so on.

Note that since we are working on a symmetric version of the CVRP and that the order of the entries in the vector $\boldsymbol{x}_e$ is important, we always assume that $i < j$.

6.4.2 ML prediction

Once the dataset is in hand, we apply common ML practices such as data preprocessing (i.e., normalization, encoding) and data splitting (i.e., dividing the data into a training, a validation and a test set), etc. As mentioned before, the prediction task corresponds to a binary classification problem and different ML models can be considered, e.g., Random Forest, SVM, Neural network. In this section, we assume that we already have a trained predictive model that takes as input the edge features $\boldsymbol{x}_e$ and outputs the predictions $\hat{y}_e$ and we wish to know what to do with those predictions afterwards (the comparison of which ML model is best performing is discussed in the next section).

ML-based edge fixing

Let $G = (V, E)$ be an undirected graph, where V is the set of nodes including the depot nodes and E the set of edges. Let $c_e, e \in E$ be the cost of edge $e = \langle i, j \rangle$. The edge flow can

be written in terms of the master problem (6.1)-(6.3) variables as

$$x_e = \sum_{r \in \Omega} b_e^r \theta_r, \tag{6.6}$$

where $b_e^r \in \{0, 1\}$ indicates whether edge $e \in E$ is part of a route $r \in \Omega$. Therefore, an edge can be fixed by adding a new constraint to the master problem. For each edge in the original solution $\mathcal{S}_o$, the flow is set to 1 depending on the predictions obtained by the ML model, namely

$$x_e = 1, \quad \forall e \in E(\mathcal{S}_o) \ : \ \hat{y}_e = 1. \tag{6.7}$$

Infeasibility case

Sometimes fixing edges can lead to an infeasible restricted problem. This may only occur in the case when the total demand of a sequence of clients fixed by the ML model exceeds the vehicle capacity Q. For each edge, in addition to the output $\hat{y}_e$ returned by the model, it is possible to obtain its probability estimate $\hat{p}_e$ of being fixed. Generally, the model assigns the class 1 if the probability estimate $\hat{p}_e$ is greater than 0.5 and 0 otherwise. For a sequence of clients whose sum of the demands exceed Q, a possible solution would be to identify and unfix the edges with lowest probabilities, until obtaining one or several feasible subsequences. Let $G_{\hat{y}} = (V_{\hat{y}}, E_{\hat{y}})$ be the graph obtained by keeping only the fixed edges and their corresponding nodes, i.e., $E_{\hat{y}} = \{e \in E \mid \hat{y}_e = 1\}$ and $V_{\hat{y}} = \{i \in V \mid \exists \langle k, l \rangle \in E_{\hat{y}} : i = k \vee i = l\}$. Depending on the predictions obtained by the model, the graph may contain multiple sequences of two nodes or more. A sequence can be defined as $p = (v_1^p, v_2^p, \ldots, v_{|p|}^p), v_i \in V_{\hat{y}}$, where $|p|$ is its length. Note that since the graph is undirected, the sequence can start at either of its ends. Let $E(p) = \{\{i, i+1\} \mid i \in \{1, 2, \ldots, |p| - 1\}\}$ be the set of edges of sequence p and $S(G_{\hat{y}})$ the set of all sequences in $G_{\hat{y}}$. We denote by d_i the demand of node i ($d_s = d_t = 0$). The steps followed to identify and to deal with the infeasible sequences are described in Algorithm 4.

The algorithm takes as input the graph $G_{\hat{y}}$ and starts by initializing the variable `infeasibilityDetected` to `False` (Step 3). Then, it proceeds by looping over the sequences of the set $S(G_{\hat{y}})$ in Step 4. For each sequence p, if the total demand of the clients exceeds the capacity Q of a vehicle (Step 5), the edge with the lowest probability estimate is identified (Step 6), then unfixed and removed from the set of edges $E_{\hat{y}}$ (Steps 7-8). This procedure is repeated until there are no more infeasible sequences.

Algorithm 4 Infeasibility check.

procedure RESOLVE_INFEASIBILITY($G_{\hat{y}}, \hat{\boldsymbol{p}}$)
  do
    `infeasibilityDetected = False`
    for `each sequence` $p \in S(G_{\hat{y}})$ **do**
      if $\sum_{i=1}^{|p|} d_{v_i^p} > Q$ **then**
        $e = \arg\min_{e \in E(p)} \hat{p}_e$ ▷ Identify the edge with lowest probability
        $E_{\hat{y}} = E_{\hat{y}} \setminus \{e\}$
        $\hat{y}_e = 0$
        `infeasibilityDetected = True`
      end if
    end for
  while `infeasibilityDetected = True`
end procedure

Network reduction

Once there are no more infeasible sequences, it is possible to make an improvement that can further accelerate the optimization, which consists in reducing the number of nodes and edges in the network. The simple (and well-known) idea is to shrink the sequences of three nodes (i.e., two edges) or more into a single edge, while making sure that the cost and the demands are updated accordingly.

For a sequence $p = (v_1^p, \dots, v_{|p|}^p) \in S(G_{\hat{y}})$ with $|p| \geq 3$, the goal is to remove all intermediate nodes $v_2^p, \dots, v_{|p|-1}^p$ and link directly the two ends of the sequence by adding the edge $e_{new} = \langle v_1^p, v_{|p|}^p \rangle$. Its cost is updated to the cost of the whole sequence, i.e., $c_{e_{new}} = \sum_{e \in E(p)} c_e$, and the demand of the removed nodes $\sum_{i=2}^{|p|-1} d_{v_i^p}$ is added to the demand of either v_1^p or $v_{|p|}^p$. Lastly, the edge is fixed by setting the corresponding flow variable $x_{e_{new}} = 1$. By doing so, traversing the edge e_{new} becomes equivalent to traversing the sequence p. The reduction of the network size depends on the number of sequences and their lengths, in some cases a significant reduction can be obtained, as illustrated in Figure (6.1d).

Method summary

By putting all the pieces together, Algorithm 5 summarizes the different steps of our method. Given the initial data $(\mathcal{P}_o, \mathcal{P}_m, \mathcal{S}_o)$, the algorithm starts by extracting the edge features yielding the features matrix $\mathbf{X}$ (Step 1), then obtaining the predictions $\hat{\boldsymbol{y}}$ of the ML model and the probability estimates $\hat{\boldsymbol{p}}$ in Steps 2-3. The graph $G_{\hat{y}}$ representing the edges to fix is built (Steps 4-5) and the procedure `resolve_infeasibility` described in Algorithm 4 is called to check for any infeasible sequences. The algorithm proceeds by aggregating the sequences in $S(G_{\hat{y}})$ by connecting each sequence ends (Step 9), updating the cost and the

demand (we added the demands of the intermediate nodes to one of the two ends) in Steps 10-11, then removing the intermediate nodes and their adjacent edges (Steps 12-13). Any removed node from $G_{\hat{y}}$ must be removed from the original graph also. This is achieved by updating G to the induced subgraph $G[V_{\hat{y}}]$. At this point, the graph G corresponds to the original complete graph minus the removed nodes from the network reduction step. Finally, the master problem is initialized and solved after adding the flow constraints for each fixed edge (Steps 17-21).

Algorithm 5 ML-based edge fixing heuristic.

Data :
$\mathcal{P}_o$: Original instance
$\mathcal{P}_m$: Modified instance
$\mathcal{S}_o$: Solution of the original instance
$G = (V, E)$: Original graph

1: $\boldsymbol{X} \longleftarrow$ `extractEdgeFeatures(`$\mathcal{P}_o$`,` $\mathcal{P}_m$`,` $\mathcal{S}_o$`)`
2: $\hat{\boldsymbol{y}} \longleftarrow$ `predict(`$\boldsymbol{X}$`)`
3: $\hat{\boldsymbol{p}} \longleftarrow$ `predict_proba(`$\boldsymbol{X}$`)`
4: $V_{\hat{y}} = V, \;\; E_{\hat{y}} = \{e \in E \mid \hat{y}_e = 1\}$
5: $G_{\hat{y}} = (V_{\hat{y}}, E_{\hat{y}})$
6: `resolve_infeasibility(`$G_{\hat{y}}, \hat{\boldsymbol{p}}$`)`
7: **for** `each sequence` $p = (v_1^p, \ldots, v_{|p|}^p) \in S(G_{\hat{y}})$ **do**
8: **if** $|p| > 2$ **then**
9: $e = \langle v_1^p, v_{|p|}^p \rangle$
10: $c_e = \sum_{u \in E(p)} c_u$
11: $d_{v_1^p} = d_{v_1^p} + \sum_{i=2}^{|p|-1} d_{v_i^p}$
12: $V_{\hat{y}} = V_{\hat{y}} \setminus \{v_i^p \mid i \in \{2, \ldots, |p| - 1\}\}$
13: $E_{\hat{y}} = E_{\hat{y}} \setminus \{\langle i, j \rangle \in E \mid i \notin V_{\hat{y}} \vee j \notin V_{\hat{y}}\}$
14: **end if**
15: **end for**
16: $G = G[V_{\hat{y}}]$
17: `MP` $\longleftarrow$ `initialize_MP(`$\mathcal{P}_m$`)`
18: **for** `each edge` $e \in E_{\hat{y}}$ **do**
19: `add_constraint(MP,"`$x_e = 1$`")`
20: **end for**
21: `solve(`$\mathcal{P}_m, G$`)`

6.5 Computational experiments

This section starts by describing the CVRP instances we used and the data generation process. Next, we present the details about the ML phase. Finally, the results of our heuristic method on different benchmark instances are reported. All the experiments were conducted

on a Linux machine with an Xeon(R) Gold 6142 CPU @ 2.60GHz and 512GB of RAM.

6.5.1 CVRP instances

The instances used are based on the X benchmark dataset introduced by Uchoa et *al.* [89]. Each instance is characterized by the following attributes :

- **Depot position :** The possible values are **Central (C)** (i.e., the depot is positioned at the center of the grid), **Eccentric (E)** (i.e., the depot is positioned at the South-West corner $(0, 0)$ of the grid) and **Random (R)**.
- **Client positioning :** The three possibilities are **Random (R)** (i.e., the clients are randomly dispersed on the grid), **Clustered (C)** (i.e., the clients are grouped in clusters) and a combination of both, referred to as **Random-Clustered (RC)**.
- **Demand distribution :** For the client demands, there are seven options with different intervals, and all demands are drawn uniformally from each distribution. The demands can range from : **(a)** [1-10], **(b)** [5-10], **(c)** [1-100], **(d)** [50-100], **(e)** clients located in an even quadrant have demands in the range [1-50] and [50,100] for the others, **(f)** 70% to 95% of the clients have a demand in the range [1,10] and [50,100] for the remaining ones and finally, **(g)** unit demands, where all the demands are equal to 1.
- **Average route size :** This represents the average number of clients that can be visited by the same route, which is directly controlled by the vehicles capacity. This is computed as n/K_{min} where n is the number of clients and K_{min} is an estimate of the minimum number of vehicles required to service all clients.

Since we are interested in modifying the demands of the clients, we exclude the instances with unit demands. Table 6.1 describes the instances we picked for our experiments. The instance names are written in the form "X-n[n_{nodes}]-k[$n_{vehicles}$]" where n_{nodes} is the number of nodes including the depot and $n_{vehicles}$ is an estimate of the number of vehicles required to service all clients. The remaining columns report the characteristics described above.

6.5.2 Data generation

For each instance described in Table 6.1, we generate a set of modified instances by randomly changing the demands of N_c% of the clients. The new demand of each of these clients is chosen randomly in the interval $[d_i - \Delta_d, d_i + \Delta_d]$, where d_i is the original demand of the client i and Δ_d a parameter controlling the interval width. In order to analyze the impact of the demand changes on the performance of the heuristic and the predictions, we used different values of $N_c \in \{10, 20, 30\}$. As for Δ_d, since each instance has a different demand distribution, we created three classes of intervals : Small (S), Medium (M) and Large (L), controlled by the value of Δ_d as shown in Table 6.2.

TABLE 6.1 CVRP instances from the X benchmark instances.

Instance name	Depot position	Client positioning	Demand distribution	Avg. route size
X-n101-k25	R	RC	$[1-100]$	4.0
X-n106-k14	E	C	$[50-100]$	7.5
X-n110-k13	C	R	$[5-10]$	8.4
X-n125-k30	R	C	*Quadrant*	4.1
X-n129-k18	E	RC	$[1-10]$	7.1
X-n134-k13	R	C	*Quadrant*	10.2
X-n139-k10	C	R	$[5-10]$	13.8
X-n143-k07	E	R	$[1-100]$	20.3

TABLE 6.2 Δ_d values used depending on the demand distribution.

Demand	Δ_d (interval size)		
distribution	**S**	**M**	**L**
$[1-100]$	5	10	15
$[50-100]$	5	10	15
$[5-10]$	1	2	3
Quadrant	5	10	15
$[1-10]$	2	3	4

By combining the three different values of N_c and the three interval sizes, this results in nine different scenarios $\Phi = \{10S, 10M, 10L, 20S, 20M, 20L, 30S, 30M, 30L\}$. We then proceeded as follows. For each instance in Table 6.1 and each scenario $\phi \in \Phi$, 100 modified instances are generated, where 95 of them are used for the ML phase (i.e., training, parameters tuning, etc.) and the remaining 5 for the optimization phase (i.e., when the ML model is incorporated in the CG algorithm). We therefore consider the eight instances of Table 6.1 as the original instances (i.e., $\mathcal{P}_o^1, \ldots, \mathcal{P}_o^8$) and, for each scenario ϕ, we generate 100 modified versions (i.e., $(\mathcal{P}_m^i)_\phi^1, \ldots, (\mathcal{P}_m^i)_\phi^{100}, i \in \{1, 2, \ldots, 8\}, \phi \in \Phi$). One ML-model is trained for each instance and for each scenario. The idea of training a model for each instance comes from the assumption that we have a specific instance that we solve repeatedly. Therefore we want a specific model for that particular instance. This makes a total of 8 instances $\times$ 9 scenarios $\times$ 100 = 7,200 modified instances generated (and $8 \times 9 = 72$ ML models). Given the large number of instances, instead of using an exact B&P method for solving and collecting solutions, we opted to use the FILO heuristic [87] (see Section 6.3.3). For each of the 7,200 instances, we run the heuristic using 10 different random seed values for 1,000,000 iterations and the solution with the smallest cost is retained.

TABLE 6.3 Hyperparameters values of the ANN models.

Hyperparameter	**Value**
Learning rate	10^{-3}
Number of epochs	1000
Epoch size	64
Batch size	32
NN Architecture	$32 \times 32 \times 32 \times 1$
Activation function	ReLU
Output function	Sigmoid
Optimizer	Adam
Class weights	Balanced

Once the solutions of the original and modified instances are obtained, for each instance $i \in \{1, \ldots, 8\}$ and scenario $\phi \in \Phi$, the solutions are grouped in a set of tuples, i.e., $\{(\mathcal{P}_o^i, \mathcal{S}_o^i, (\mathcal{P}_m^i)_\phi^1, (\mathcal{S}_m^i)_\phi^1), \ldots, (\mathcal{P}_o^i, \mathcal{S}_o^i, (\mathcal{P}_m^i)_\phi^{100}, (\mathcal{S}_m^i)_\phi^{100})\}$ that are used to extract the edge features and labels as detailed in Section 6.4.1.

6.5.3 Machine learning phase

Before starting the training, common practices in ML are followed, starting by a pre-processing phase that consists of scaling and normalizing the data. The dataset (i.e., the data from the 95 instances) is then split into a training set, a validation set and a test set. The goal of the validation set is to tune the different hyperparameters, whereas the purpose of the test set is to compare different classification algorithms, such as logistic regression, K-nearest neighbors, random forest, artificial neural network (ANN), etc. According to the results obtained on the test set, the ANN model is the overall most robust model with accuracies ranging from 70% to 88% depending on the instance and the scenario $\phi \in \Phi$ (more detailed results about the models performance are reported in the next section). The hyperparameter values used during the training of the ANN models are described in Table 6.3.

6.5.4 Optimization phase

In this section, we present the results obtained by our edge-fixing heuristic. Since a ML model is trained for each original instance and scenario, we report the ML model performance in this section as well. Tables 6.4 to 6.6 summarize the results obtained on the test instances. Each row corresponds to the average values obtained on the 5 test instances (for each original instance), whereas the details of each individual instance can be found in Appendix A.1. There is one table for each N_c value (i.e., 10, 20 and 30 for Tables 6.4, 6.5 and 6.6, respectively).

The first column corresponds to the interval used when changing the demands (i.e., S, M and L, respectively), followed by the name of the original instance $\mathcal{P}_o$. Next, the average cost of the best solutions $\mathcal{S}_m$ obtained by the FILO heuristic (over the 10 executions of the heuristic with different seeds). In the fourth column, we report the average similarity between the solution of the original instance and the solution of the modified instances computed by the following formula :

$$sim(\mathcal{S}_o, \mathcal{S}_m) = \frac{|E(\mathcal{S}_o) \cap E(\mathcal{S}_m)|}{|E(\mathcal{S}_o)|}. \tag{6.8}$$

Notice that this similarity also matches the percentage of edges to fix (i.e., with the label 1). The next three columns summarize the performance of the ML model and show the following metrics : the True Negative Rate (**TNR**) corresponds to the percentage of edges that should not be fixed and that are predicted accurately ; the True Positive Rate (**TPR**) is equal to the percentage of edges that should be fixed and that are predicted correctly ; and the (balanced) accuracy is the mean of the two previous columns. By fixing the edges suggested by the ML model and solving the instance using VRPSolver, we obtain the results shown under the heading “Edge-fixing" : the average costs of the solutions as well as the computing times in seconds. To evaluate the quality of this solution, we report the average gap that compares the cost of the solution of our method with that of the solution $\mathcal{S}_m$ of the FILO heuristic (third column). The next three columns provide the average cost of the solutions computed by the exact algorithm in VRPSolver, the average computing time and the ratio with respect to the computing time of our method (i.e., the computing time of the exact B&P algorithm divided by the computing time of our edge-fixing method). For the exact algorithm, a time limit of five hours is set. Empty values mean that the exact B&P failed to find an optimal solution in the time limit for one or more of the 5 instances, refer to Appendix A.1 for additional details.

A natural question with respect to the proposed approach is if there is value in trying to learn the difference between solutions of slightly modified instances. Indeed, ML-based CVRP heuristics should be in a very favorable position in our computational setting because the instances are from a specific data distribution and of the same size, i.e., no generalization is needed. In order to give a tentative answer to this question, we compare the performance of our method with another ML method for the CVRP not designed for reoptimization purposes but with excellent overall performance on recent benchmarks. We chose the *Dual-Aspect Collaborative Transformer* algorithm (DACT [90]) that is considered one of the most effective methods for solving the CVRP to-date in terms of solution quality according to

TABLE 6.4 Average results for scenarios with $N_c = 10$.

Interval	$\mathcal{P}_o$	$\mathcal{S}_m$ cost	$sim(\mathcal{S}_o, \mathcal{S}_m)$	ML model metrics			Edge-fixing			Exact B&P			DACT	
				TNR	TPR	Accuracy	Cost	Time (s)	Gap	Cost	Time (s)	Ratio	Cost	Gap
Small	X-n101-k25	27637	84%	71%	70%	70%	27718	13	0.29%	27635	211	22.4	28184	1.98%
	X-n106-k14	26376	85%	89%	60%	75%	26436	25	0.23%	26376	962	61.3	26897	1.98%
	X-n110-k13	14987	93%	100%	66%	83%	14987	8	0.00%	14987	265	36.7	15159	1.15%
	X-n125-k30	55613	63%	76%	73%	75%	55683	235	0.13%	-	-	-	58581	5.34%
	X-n129-k18	28765	62%	78%	72%	75%	28982	78	0.75%	28765	3479	53.4	29697	3.24%
	X-n134-k13	10888	80%	89%	54%	72%	10917	77	0.27%	-	-	-	11284	3.64%
	X-n139-k10	13590	85%	92%	81%	86%	13599	26	0.06%	-	-	-	13846	1.88%
	X-n143-k07	15722	81%	87%	88%	88%	15726	54	0.02%	-	-	-	16245	3.32%
Average		**23886**	**79%**	**85%**	**71%**	**78%**	**24256**	**65**	**0.22%**	**-**	**-**	**-**	**24987**	**2.82%**
Medium	X-n101-k25	27606	83%	77%	63%	70%	27736	18	0.47%	27606	324	20.4	28298	2.51%
	X-n106-k14	26358	66%	95%	73%	84%	26380	14	0.08%	26358	355	23.9	26871	1.95%
	X-n110-k13	14971	87%	98%	67%	82%	14993	12	0.15%	14969	283	24.5	15137	1.11%
	X-n125-k30	55713	60%	74%	73%	74%	55758	198	0.08%	55655	5156	58.4	58735	5.42%
	X-n129-k18	28862	60%	74%	75%	75%	29124	105	0.91%	-	-	-	29801	3.26%
	X-n134-k13	10888	63%	74%	73%	74%	11024	320	1.25%	-	-	-	11304	3.82%
	X-n139-k10	13601	90%	85%	76%	81%	13608	27	0.05%	-	-	-	13863	1.92%
	X-n143-k07	15707	85%	97%	79%	88%	15710	63	0.02%	-	-	-	16200	3.14%
Average		**24213**	**74%**	**84%**	**72%**	**78%**	**24292**	**95**	**0.38%**	**-**	**-**	**-**	**25026**	**2.89%**
Large	X-n101-k25	27651	70%	63%	77%	70%	28125	122	1.71%	27648	399	7.9	28142	1.98%
	X-n106-k14	26412	65%	84%	74%	79%	26557	250	0.54%	-	-	-	26846	1.64%
	X-n110-k13	15030	75%	86%	73%	80%	15107	35	0.51%	15030	2283	60.7	15187	1.04%
	X-n125-k30	55733	55%	81%	75%	78%	55825	422	0.16%	-	-	-	58500	4.97%
	X-n129-k18	28755	62%	78%	73%	75%	28972	171	0.76%	28748	2457	22.5	29546	2.75%
	X-n134-k13	10908	69%	72%	68%	70%	10908	164	0.66%	-	-	-	11267	3.29%
	X-n139-k10	13600	83%	85%	75%	80%	13635	47	0.26%	-	-	-	13850	1.84%
	X-n143-k07	15716	88%	97%	73%	85%	15717	62	0.00%	-	-	-	16265	3.49%
Average		**24226**	**71%**	**81%**	**73%**	**77%**	**24365**	**159**	**0.58%**	**-**	**-**	**-**	**24950**	**2.62%**

the recent survey [57]. Unlike our approach, DACT is a reinforcement learning method that learns an improvement heuristic, which means that it starts with an initial solution and tries to improve it in an iterative way. In this method, the learner (i.e., the agent) learns to identify a pair of nodes on which to apply a pairwise operator (e.g., 2-opt, swap, insert) and is rewarded when a better solution is found. The authors report good results that outperform several other learning methods on the X benchmark instances (same instances we are using) in a reasonable computing time. In our case, we proceeded by using the same pretrained model that the authors used in their paper but with additional training on the 8 instances we focus on (see Table 6.1), in addition to using the same parameters of their best performing model. The results obtained are reported in the last two columns of Tables 6.4 to 6.6, representing respectively the average cost of the solutions and the gap with respect to the FILO heuristic (just like Edge-Fixing).

According to the results reported in Tables 6.4 to 6.6, we can notice that on average the similarity decreases by increasing the number of changes we make on the demands (controlled by the parameter N_c and the intervals), which is expected. A high similarity of 79% is noticed on average on the instances with 10% change and small interval, while the instances with 30% change and large interval have an average similarity of 63%. At the instance level, it

TABLE 6.5 Average results for scenarios with $N_c = 20$.

Interval	$\mathcal{P}_o$	$\mathcal{S}_m$ cost	$sim(\mathcal{S}_o, \mathcal{S}_m)$	ML model metrics			Edge-fixing			Exact B&P			DACT	
				TNR	TPR	Accuracy	Cost	Time (s)	Gap	Cost	Time (s)	Ratio	Cost	Gap
Small	X-n101-k25	27486	83%	73%	75%	74%	27628	29	0.51%	27486	231	8.6	28220	2.67%
	X-n106-k14	26338	72%	90%	71%	81%	26401	30	0.24%	26336	1147	88.5	26845	1.93%
	X-n110-k13	14980	77%	86%	74%	80%	15024	9	0.29%	14980	429	55.5	15152	1.15%
	X-n125-k30	55493	63%	85%	68%	77%	55509	233	0.03%	-	-	-	58372	5.19%
	X-n129-k18	29020	56%	71%	78%	74%	29408	143	1.33%	29009	5791	76.5	29703	2.35%
	X-n134-k13	10909	73%	85%	69%	77%	10943	150	0.32%	-	-	-	11323	3.80%
	X-n139-k10	13613	87%	91%	81%	86%	13616	103	0.02%	-	-	-	13870	1.89%
	X-n143-k07	15715	86%	90%	86%	88%	15748	30	0.21%	-	-	-	16261	3.47%
Average		**24194**	**75%**	**84%**	**75%**	**80%**	**24284**	**91**	**0.37%**	**-**	**-**	**-**	**24968**	**2.81%**
Medium	X-n101-k25	27512	68%	74%	72%	73%	27694	32	0.66%	27512	518	19.4	28076	2.05%
	X-n106-k14	26306	64%	87%	71%	79%	26400	20	0.36%	26306	5257	238.2	26785	1.72%
	X-n110-k13	14983	73%	86%	77%	82%	15078	27	0.64%	14983	548	30.3	15217	1.56%
	X-n125-k30	55503	54%	82%	74%	78%	55579	233	0.14%	-	-	-	58528	5.45%
	X-n129-k18	29171	57%	73%	76%	74%	29655	315	1.66%	-	-	-	30001	2.85%
	X-n134-k13	10877	55%	82%	80%	81%	10956	127	0.73%	-	-	-	11258	3.50%
	X-n139-k10	13605	72%	87%	80%	84%	13631	205	0.20%	-	-	-	13855	1.84%
	X-n143-k07	15708	81%	80%	83%	82%	15745	49	0.24%	-	-	-	16233	3.35%
Average		**24208**	**65%**	**81%**	**77%**	**79%**	**24342**	**126**	**0.58%**	**-**	**-**	**-**	**24991**	**2.79%**
Large	X-n101-k25	27645	59%	70%	73%	72%	27871	56	0.81%	27645	466	30.8	28244	2.17%
	X-n106-k14	26413	61%	79%	74%	77%	26555	100	0.54%	-	-	-	26834	1.59%
	X-n110-k13	15034	68%	81%	79%	80%	15161	23	0.84%	15034	487	25.4	15216	1.21%
	X-n125-k30	55958	58%	79%	70%	74%	56168	433	0.37%	-	-	-	58629	4.77%
	X-n129-k18	29123	55%	77%	74%	75%	29462	386	1.16%	29092	3106	16.0	29848	2.49%
	X-n134-k13	10909	55%	82%	80%	81%	11051	254	1.31%	-	-	-	11317	3.75%
	X-n139-k10	13585	73%	88%	77%	83%	13619	139	0.25%	-	-	-	13788	1.50%
	X-n143-k07	15743	84%	79%	79%	79%	15775	41	0.20%	-	-	-	16332	3.74%
Average		**24301**	**64%**	**79%**	**76%**	**78%**	**24458**	**179**	**0.69%**	**-**	**-**	**-**	**25026**	**2.65%**

seems that the dispersion of the clients as well as the length of the routes can have an impact on the similarity of the solutions. For example, the instances with randomly dispersed clients and medium to long routes (e.g., X-n110-k13, X-n139-k10, X-n143-k07) tend to have a higher similarity compared to instances with clustered clients (e.g., X-n106-k14 and X-n125-k30).

Before evaluating the performance of the model, we would like to highlight the impact that a bad prediction may have in the obtained solution. If some edges are not fixed when they should be (false negatives), it can affect the computing time but the solver can still include them in the final solution. However, the edges with label 0 can have a more serious impact, since they can affect the quality of the solution if predicted inaccurately (false positives), which potentially leads to a higher gap. As previously mentioned, the similarity is also the percentage of edges to be fixed. A high similarity means that most of the edges in the solution of the original instance are labeled 1, which also implies that the ML model tends to make fewer “significant" errors since there are not many edges with label 0. If we focus on the solution quality, it is possible to give a higher weight to the edges with label 0, which can lead to a high TNR (thus reducing the false positives) and probably a lower TPR, meaning that less edges will be fixed resulting in a higher computing time. Conversely, if we do the opposite and assign a higher weight to the edges with label 1, we will likely fix more edges

TABLE 6.6 Average results for scenarios with $N_c = 30$.

Interval	$\mathcal{P}_o$	$\mathcal{S}_m$ cost	$sim(\mathcal{S}_o, \mathcal{S}_m)$	ML model metrics			Edge-fixing			Exact B&P			DACT	
				TNR	TPR	Accuracy	Cost	Time (s)	Gap	Cost	Time (s)	Ratio	Cost	Gap
Small	X-n101-k25	27562	75%	80%	70%	75%	27669	37	0.39%	27562	178	5.9	28219	2.38%
	X-n106-k14	26383	65%	89%	69%	79%	26438	148	0.21%	26378	1577	25.9	26833	1.70%
	X-n110-k13	15005	74%	92%	77%	84%	15083	32	0.52%	15005	555	21.8	15157	1.01%
	X-n125-k30	55776	53%	77%	73%	75%	55834	259	0.10%	-	-	-	58601	5.06%
	X-n129-k18	29414	58%	80%	74%	77%	29778	237	1.24%	29405	3742	24.8	30086	2.28%
	X-n134-k13	10925	77%	90%	62%	76%	10952	140	0.25%	-	-	-	11361	4.00%
	X-n139-k10	13622	83%	91%	73%	82%	13692	154	0.51%	-	-	-	13829	1.52%
	X-n143-k07	15754	83%	79%	87%	83%	15822	56	0.43%	-	-	-	16328	3.65%
Average		**24305**	**71%**	**85%**	**73%**	**79%**	**24409**	**133**	**0.46%**	**-**	**-**	**-**	**25052**	**2.70%**
Medium	X-n101-k25	27654	63%	68%	73%	71%	27804	51	0.54%	27651	243	24.8	28277	2.25%
	X-n106-k14	26437	59%	75%	70%	73%	26615	247	0.67%	-	-	-	26780	1.30%
	X-n110-k13	15058	77%	87%	72%	79%	15095	14	0.24%	15058	269	33.6	15292	1.55%
	X-n125-k30	55925	50%	82%	78%	80%	56009	451	0.15%	-	-	-	58859	5.25%
	X-n129-k18	29221	54%	79%	75%	77%	29698	137	1.63%	29215	3574	34.8	29849	2.16%
	X-n134-k13	10926	56%	83%	80%	81%	11048	129	1.12%	-	-	-	11301	3.43%
	X-n139-k10	13640	75%	89%	78%	83%	13708	256	0.49%	-	-	-	13926	2.09%
	X-n143-k07	15782	80%	78%	85%	82%	15870	49	0.55%	-	-	-	16423	4.06%
Average		**24330**	**64%**	**80%**	**76%**	**78%**	**24481**	**167**	**0.67%**	**-**	**-**	**-**	**25088**	**2.76%**
Large	X-n101-k25	27617	66%	76%	72%	74%	27929	185	1.12%	27617	339	19.4	28207	2.13%
	X-n106-k14	26440	54%	79%	68%	74%	26637	275	0.75%	-	-	-	26913	1.79%
	X-n110-k13	15090	66%	82%	72%	77%	15211	88	0.80%	15090	1120	18.7	15318	1.51%
	X-n125-k30	56228	52%	84%	71%	78%	56221	475	-0.01%	-	-	-	58725	4.44%
	X-n129-k18	29674	58%	75%	73%	74%	30179	75	1.69%	29670	4914	69.9	30554	2.97%
	X-n134-k13	10950	53%	83%	82%	82%	11029	145	0.72%	-	-	-	11423	4.31%
	X-n139-k10	13616	73%	91%	76%	84%	13635	128	0.14%	-	-	-	13868	1.85%
	X-n143-k07	15884	79%	82%	83%	83%	15941	313	0.36%	-	-	-	16326	2.79%
Average		**24437**	**63%**	**82%**	**75%**	**78%**	**24598**	**211**	**0.70%**	**-**	**-**	**-**	**25167**	**2.72%**

and achieve a lower computing time but at the expense of bad-quality solutions. In our case, we use a balanced weight between the two classes as shown in Table 6.3. Depending on the desired goal, one seeks to find a compromise between quality and computing time.

From the ML metrics obtained we can observe an average accuracy ranging from 78% to 80%, but if we look closer at the individual instances, we can notice some variance and a slight positive correlation between the accuracy and the similarity. By fixing the edges proposed by the model and comparing the solution with the one of the FILO heuristic, we obtain an average gap ranging from 0.22% to 0.70%. We can also notice that the gap follows the same tendency as the similarity, i.e., more changes of the demands imply less similarity, more possible misclassifications and thus a higher gap. Therefore, the instances with high similarity (e.g., instances with dispersed clients and long routes) generally have a lower gap compared to the others. The detailed results in Appendix A.1 indicate that the gap of the individual instances can vary between 0% and 1.71%. A perfect solution is obtained (i.e., same solution as the FILO heuristic, thus a 0% gap) when the TNR is 100%. In a few rare cases, the solver was able to find a better solution than the heuristic even after fixing parts of the solution, such as X-n125-k30_10M_4 in Table A.2 and X-n125-k30_30L_3 in Table A.9. Generally, the FILO heuristic finds very good solutions (especially when considering

10 runs) : it succeeds in finding the optimal solution in many cases when compared to the available exact B&P results. We believe that the gap reported between the edge-fixing and the FILO heuristic must not be far from the one with the exact B&P algorithm.
In terms of computing time, the edge fixing method takes only a few minutes or even a few seconds in some cases to find the optimal solution of the modified problem (that with fixed variables according to the ML prediction), whereas the exact B&P algorithm that solves the (original) problem from scratch can take hours of computation to find an optimal solution, and in many cases no solution is found after the time limit especially for large instances. Apparently, the current version of VRPSolver does not have strong heuristics to produce feasible solutions. On the one side, this is not an issue for our computational investigation because we use VRPSolver as well (and we are definitely not using it for comparison). On the other side, this is somehow reinforcing the fact that, after our proposed fixing, the instances are much easier and even without good heuristics VRPSolver is very effective on them. Note that in our experiments, the FILO heuristic with 1,000,000 iterations takes about 8 to 13 minutes per execution depending on the instance size, not to forget that we executed the heuristic 10 different times with different seeds and retained the best solution. Overall, the computing time of the edge-fixing heuristic remains lower than a single execution of FILO and way lower than an exact B&P algorithm.
Concerning the DACT method, we can notice an average gap of 2.7%. The method performs better on some instances than on others with gaps ranging from 0.03% up to 5.72% according to the detailed results in Appendix A.1. Since the method is iterative and performs a certain number of steps at the inference phase (more precisely 10,000 steps as described in [90]), its computing time is not negligible and can range from 10 to 15 minutes on average depending on the instance size. Therefore, the edge-fixing method outperforms DACT both in terms of computing time and solution quality.

6.6 Conclusion

In this paper, we proposed a ML-based heuristic for the reoptimization of repeatedly solved problems after minor changes in the problem data. More precisely, we put ourselves in the context of a company that solves CVRPs on a daily basis where the locations of the clients are the same but slightly different demands occur. Given that there can be a great similarity between the solutions, the goal was to exploit the ones obtained from previous executions (not necessarily optimal ones) in order to speed up the reoptimization of future instances. The aim was to predict and fix the edges that have a high chance of remaining in the solution after a change of the demands.
Following a supervised learning approach, we trained neural network models on the data

collected from a recent heuristic for the CVRP called FILO. The models achieved an average accuracy of 78% on the test instances. By incorporating the predictions in our edge-fixing method, we were able to find solutions in reasonable computing times with gaps ranging from 0% to 1.71% (with an average of 0.51%) when compared to the FILO heuristic we learned from. These gaps are also lower compared to other recent ML methods proposed in the literature such as the DACT method used to compute the entire CVRP solution. We also obtained an acceleration effect, considerably reducing the size of the network and allowing an even faster reoptimization.

Future work can seek some improvements, for example by the exploration of more complex learning models that can take advantage of the graph structure of the problem or even sequence models since we are dealing with routes. This may potentially lead to a better accuracy and therefore better gaps and lower computing times.

CHAPITRE 7 DISCUSSION GÉNÉRALE

Dans le cadre de cette thèse, nous présentons trois approches qui exploitent l'apprentissage machine pour accélérer la méthode de GC. Les techniques décrites sont indépendantes les unes des autres et peuvent être combinées pour obtenir des réductions plus importantes en termes de calcul, que ça soit du côté du PMR ou bien du SP. Les méthodes ont aussi un fort potentiel d'être appliquées à d'autres problèmes d'optimisation.

Pour autant que nous sachions, il n'y a eu que très peu de travaux spécifiquement dédiés à la sélection de colonnes dans la littérature : par exemple, l'ajout de colonnes en se basant sur le coût réduit, ou encore la diversification des colonnes à ajouter (e.g., blocs disjoints, voir section 2.1.6). La méthode présentée dans le premier sujet de cette thèse constitue une contribution novatrice à cet égard. Nous proposons une stratégie de sélection qui prend en considération plusieurs caractéristiques à la fois, afin de fournir une sélection de colonnes plus efficace. L'utilisation d'un modèle GNN est parfaitement adaptée à cette tâche, puisque le problème de prédiction est modélisé sur un graphe biparti où la structure colonnes-contraintes est exploitée. Ceci permet de capturer la relation entre les colonnes nouvellement générées et les colonnes et contraintes du PMR. De plus, les prédictions sont obtenues en un temps négligeable.

La performance de la GC dépend beaucoup de l'efficacité de résolution du SP. Pour les problèmes définis sur un graphe et impliquant un SPPRC comme SP, réduire la taille du réseau est sans aucun doute une façon efficace pour accélérer la résolution. Certaines stratégies de ce type ont été proposées dans la littérature cherchant à réduire le nombre d'arcs du réseau. Parmi les stratégies les plus efficaces, on retrouve la sélection basée sur les coûts réduits des arcs (RedCost-S comme décrit dans le chapitre 5). Cependant cette stratégie n'est applicable que dans le cas où chaque arc est associé à au moins une tâche (i.e., contribue à une contrainte du PM). Dans le deuxième sujet, nous proposons une méthode plus générale, qui exploite les données historiques qui peuvent être collectées durant les exécutions précédentes. En effet, contrairement au premier sujet où nous devons résoudre un MILP pour identifier les colonnes à sélectionner, dans le deuxième sujet, les données des arcs sont prêtes à être collectées à chaque fois que le SPPRC (ou sa variante) est résolu, et ce sans calculs supplémentaires. La méthode offre également une certaine flexibilité. Dans le cas où le réseau sous-jacent du problème comprend plusieurs types d'arcs, il est possible de limiter la sélection à seulement un sous-ensemble des arcs, comme il a été démontré dans le cas du VCSP.

La troisième contribution de cette thèse consiste à proposer une méthode dédiée aux pro-

blèmes d'optimisation qui sont résolus d'une façon répétitive. Dans un tel scénario, on peut s'attendre à ce que les solutions ne changent que peu lorsque de légères modifications sont apportées aux instances précédemment résolues. Les solutions historiques contiennent donc de riches informations qui peuvent être exploitées en utilisant du ML. En fournissant une solution partielle au solveur, le temps de résolution peut être considérablement réduit, tel que démontré dans le troisième sujet. De plus, dans le cas du CVRP, nous avons également proposé une stratégie d'agrégation, permettant de réduire la taille du réseau en agrégeant les séquences d'arêtes fixées par le modèle de prédiction.

CHAPITRE 8 CONCLUSION ET RECOMMANDATIONS

La recherche effectuée dans le cadre de cette thèse fournit des exemples réussis de l'intégration de l'apprentissage machine et la GC. Nous avons proposé trois approches différentes visant à réduire les temps de calcul. Les articles contribuent à la littérature en proposant de nouvelles idées applicables à une variété de problèmes d'optimisation.

8.1 Synthèse des travaux

Dans le premier sujet, nous proposons une méthode de sélection de colonnes adaptée aux problèmes qui prennent un temps considérable dans la réoptimisation des PMRs. L'objectif est de sélectionner les colonnes les plus prometteuses à chaque itération à l'aide d'un algorithme d'apprentissage supervisé. Le problème d'apprentissage a été réduit à un problème de classification de nœuds sur un graphe biparti en utilisant un modèle GNN. Les résultats ont montré la capacité du modèle à généraliser à des instances de tailles différentes de celles utilisées durant l'apprentissage, ainsi que des réductions en temps de calcul atteignant 30%.

Toujours dans le contexte de la GC, nous développons dans le deuxième sujet une heuristique basée sur l'apprentissage machine pour la résolution du SPPRC ou une de ses variantes. La méthode consiste à réduire la taille du réseau, en ne conservant que les arcs qui ont de grandes chances de faire partie des bonnes colonnes. Les arcs sélectionnés par le modèle sont utilisés pour construire un réseau réduit, qui est utilisé à chaque itération pour générer de nouvelles colonnes. Lorsque le réseau réduit ne parvient pas à trouver un nombre satisfaisant de colonnes, le réseau complet est utilisé à la place, jusqu'à l'atteinte d'une solution optimale. Les résultats ont montré des réductions en temps de calcul allant jusqu'à 27% pour le VCSP, et 40% pour le VRPTW. Les modèles entraînés ont également montré une capacité à généraliser à de nouvelles instances non rencontrées pendant la phase de l'apprentissage.

Dans le troisième sujet, nous proposons une heuristique pour la réoptimisation des problèmes résolus de manière répétitive après un changement mineur des données. Plus précisément, nous nous plaçons dans le contexte d'une compagnie qui résout des CVRPs de façon quotidienne où les clients sont les mêmes mais avec des demandes légèrement différentes. Étant donné qu'il peut y avoir une grande similarité entre les solutions, le but est de prédire puis de fixer les arêtes qui ont une grande probabilité de demeurer dans une solution après un changement de demandes. En ajustant les paramètres du modèle de prédiction, il est possible de trouver un bon compromis entre la qualité des solutions et les temps de calcul. Les résultats numériques obtenus en incorporant les prédictions dans un algorithme B&P ont permis

l'obtention de solutions avec des écarts d'optimalité variant entre 0% et 1,71%, et ce en des temps de calcul très raisonnables.

8.2 Limitations et améliorations futures

Dans cette section, nous décrivons quelques limitations des méthodes proposées ainsi que quelques améliorations potentielles.

Il est connu que l'apprentissage machine nécessite un grand ensemble de données. Dans nos expérimentations, nous nous sommes limités à l'utilisation d'instances académiques. Des générateurs d'instances ont été utilisés pour produire assez de données pour l'entraînement. Comme étape suivante, il serait intéressant de tester les méthodes proposées sur des instances industrielles, et éventuellement les intégrer dans des logiciels commerciaux. Pour ce qui est des applications, il serait également intéressant de tester les approches proposées sur d'autres types de problèmes. Entre autres, la sélection de colonnes peut être testée sur des problèmes de découpes ou de coloration de graphe par exemple.

Comme précédemment mentionné, la méthode de sélection de colonnes est plus adaptée aux problèmes qui prennent plus de temps de calcul dans la réoptimisation des PMRs que dans la résolution des SPs, ce qui n'est pas le cas pour plusieurs problèmes d'optimisation. De plus, dans la phase de collecte de données, il est nécessaire de résoudre un MILP à chaque itération, ce qui est assez coûteux et pas pratique.

Dans les expérimentations conduites dans les deux premiers sujets, nous nous sommes concentrés uniquement sur la relaxation linéaire des problèmes, et ceci pour deux raisons : i) pour se focaliser sur la méthode de GC et ii) pour comparer plus facilement les algorithmes. En effet, la recherche de solutions entières ajoute un peu d'aléatoire dans le processus (e.g., branchement), ce qui peut influencer et rendre la comparaison plus difficile entre les différentes méthodes. De toute façon, fixer des variables durant le B&B ne doit pas affecter l'applicabilité des méthodes proposées.

Parmi les améliorations qui peuvent être apportées au niveau de la sélection de colonnes, nous envisageons la définition d'un algorithme "expert" différent qui peut prendre en considération plusieurs itérations futures et non seulement la suivante. De plus, il existe d'autres méthodes récentes qui ont le potentiel d'améliorer la performance du GNN, comme par exemple l'ajout du mécanisme d'attention, ou l'utilisation d'une autre architecture. Ceci s'applique également aux deuxième et troisième sujets. Les algorithmes d'apprentissage utilisés dans ces derniers (RF, ANN) considèrent chaque arc/arête individuellement. Il est possible d'expérimenter avec des modèles d'apprentissage plus complexes qui peuvent tirer parti de la structure des

graphes, ou même des modèles traitant des séquences puisque nous considérons des chemins dans nos applications. De telles améliorations peuvent conduire à de meilleures prédictions par les modèles, et donc de meilleurs résultats.

RÉFÉRENCES

[1] G. Desaulniers, J. Desrosiers et M. M. Solomon, *(eds.) Column generation.* New York : Springer, 2005.

[2] P. Gilmore et R. E. Gomory, "A linear programming approach to the cutting stock problem," *Operations Research*, vol. 9, p. 849–859, 1961.

[3] D. R. Morrison, S. H. Jacobson, J. J. Sauppe et E. C. Sewell, "Branch-and-bound algorithms : A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, p. 79–102, 2016.

[4] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh et P. H. Vance, "Branch-and-price : Column generation for solving huge integer programs," *Operations Research*, vol. 46, p. 316–329, 1996.

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

[6] F. Vanderbeck, "Implementing mixed integer column generation," dans *Column Generation*, G. Desaulniers, J. Desrosiers et M. M. Solomon, édit. Boston, MA : Springer US, 2005, p. 331–358.

[7] O. du Merle, D. Villeneuve, J. Desrosiers et P. Hansen, "Stabilized column generation," *Discrete Mathematics*, vol. 194, n°. 1, p. 229–237, 1999.

[8] H. M. Ben Amor, J. Desrosiers et A. Frangioni, "On the choice of explicit stabilizing terms in column generation," *Discrete Applied Mathematics*, vol. 157, n°. 6, p. 1167–1184, 2009.

[9] A. Pessoa, R. Sadykov, E. Uchoa et F. Vanderbeck, "Automation and combination of linear-programming based stabilization techniques in column generation," *INFORMS Journal on Computing*, vol. 30, n°. 2, p. 339–360, 2018.

[10] L. M. Rousseau, M. Gendreau et D. Feillet, "Interior point stabilization for column generation," *Operations Research Letters*, vol. 35, n°. 5, p. 660–668, 2007.

[11] I. Elhallaoui, D. Villeneuve, F. Soumis et G. Desaulniers, "Dynamic aggregation of set-partitioning constraints in column generation," *Operations Research*, vol. 53, n°. 4, p. 632–645, 2005.

[12] I. Elhallaoui, A. Metrane, F. Soumis et G. Desaulniers, "Multi-phase dynamic constraint aggregation for set partitioning type problems," *Mathematical Programming*, vol. 123, n°. 2, p. 345–370, 2010.

[13] I. Elhallaoui, A. Metrane, G. Desaulniers et F. Soumis, “An improved primal simplex algorithm for degenerate linear programs,” *INFORMS Journal on Computing*, vol. 23, n°. 4, p. 569–577, 2011.

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

[15] A. Zaghrouti, I. El Hallaoui et F. Soumis, “Improved integral simplex using decomposition for the set partitioning problem,” *EURO Journal on Computational Optimization*, vol. 6, n°. 2, p. 185–206, Jun 2018.

[16] A. Tahir, G. Desaulniers et I. El Hallaoui, “Integral column generation for the set partitioning problem,” *EURO Journal on Transportation and Logistics*, vol. 8, n°. 5, p. 713–744, 2019.

[17] ——, “Integral column generation for set partitioning problems with side constraints,” *INFORMS Journal on Computing*, vol. 34, n°. 4, p. 2313–2331, 2022.

[18] S. Irnich et G. Desaulniers, “Shortest path problems with resource constraints,” dans *Column Generation*, G. Desaulniers, J. Desrosiers et M. M. Solomon, édit. Boston, MA : Springer US, 2005, p. 33–65.

[19] S. Irnich, “Resource extension functions : properties, inversion, and generalization to segments,” *OR Spectrum*, vol. 30, n°. 1, p. 113–148, Jan 2008.

[20] G. Righini et M. Salani, “Symmetry helps : Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints,” *Discrete Optimization*, vol. 3, n°. 3, p. 255–273, 2006.

[21] D. Feillet, P. Dejax, M. Gendreau et C. Gueguen, “An exact algorithm for the elementary shortest path problem with resource constraints : Application to some vehicle routing problems,” *Networks*, vol. 44, n°. 3, p. 216–229, 2004.

[22] S. Irnich et D. Villeneuve, “The shortest-path problem with resource constraints and k-cycle elimination for k $\geq$ 3,” *INFORMS Journal on Computing*, vol. 18, n°. 3, p. 391–406, 2006.

[23] G. Desaulniers, F. Lessard et A. Hadjar, “Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows,” *Transportation Science*, vol. 42, n°. 3, p. 387–404, 2008.

[24] R. Baldacci, A. Mingozzi et R. Roberti, “New route relaxation and pricing strategies for the vehicle routing problem,” *Operations Research*, vol. 59, n°. 5, p. 1269–1283, 2011.

[25] G. Desaulniers et D. Villeneuve, “The shortest path problem with time windows and linear waiting costs,” *Transportation Science*, vol. 34, n°. 3, p. 312–319, 2000.

[26] C. Contardo, G. Desaulniers et F. Lessard, “Reaching the elementary lower bound in the vehicle routing problem with time windows,” *Networks*, vol. 65, n°. 1, p. 88–99, 2015.

[27] L. Costa, C. Contardo et G. Desaulniers, “Exact branch-price-and-cut algorithms for vehicle routing,” *Transportation Science*, vol. 53, n°. 4, p. 946–985, 2019.

[28] A. Pessoa, R. Sadykov, E. Uchoa et F. Vanderbeck, “A generic exact solver for vehicle routing and related problems,” *Mathematical Programming*, vol. 183, n°. 1, p. 483–523, Sep 2020.

[29] D. Pecin, C. Contardo, G. Desaulniers et E. Uchoa, “New enhancements for the exact solution of the vehicle routing problem with time windows,” *INFORMS Journal on Computing*, vol. 29, n°. 3, p. 489–502, 2017.

[30] R. Sadykov, E. Uchoa et A. Pessoa, “A bucket graph–based labeling algorithm with application to vehicle routing,” *Transportation Science*, vol. 55, n°. 1, p. 4–28, 2021.

[31] K. Haase, G. Desaulniers et J. Desrosiers, “Simultaneous vehicle and crew scheduling in urban mass transit systems,” *Transportation Science*, vol. 35, n°. 3, p. 286–303, 2001.

[32] G. Desaulniers, J. Desrosiers et M. M. Solomon, “Accelerating strategies for column generation methods in vehicle routing and crew scheduling problems,” dans *Essays and Surveys in Metaheuristics*, C. C. Ribeiro et P. Hansen, édit. Norwell, MA : Kluwer, 2002, p. 309–324.

[33] R. Fukasawa, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa et R. F. Werneck, “Robust branch-and-cut-and-price for the capacitated vehicle routing problem,” *Mathematical Programming*, vol. 106, p. 491–511, 2006.

[34] T. Hastie, R. Tibshirani et J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA : Springer New York Inc., 2001.

[35] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York : Springer, 2006.

[36] I. Goodfellow, Y. Bengio et A. Courville, *Deep Learning*. MIT Press, 2016.

[37] W. S. McCulloch et W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, n°. 4, p. 115–133, Dec 1943.

[38] D. E. Rumelhart, G. E. Hinton et R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, n°. 6088, p. 533–536, Oct 1986.

[39] M. Gori, G. Monfardini et F. Scarselli, “A new model for learning in graph domains,” dans *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, July 2005, p. 729–734 vol. 2.

[40] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner et G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks*, vol. 20, n°. 1, p. 61–80, 2009.

[41] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li et R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv :1806.01261*, 2018.

[42] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang et P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv :1901.00596*, 2019.

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

[44] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris et P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," *arXiv preprint*, 2021.

[45] R. Václavík, A. Novák, P. Šůcha et Z. Hanzálek, "Accelerating the branch-and-price algorithm using machine learning," *European Journal of Operational Research*, vol. 271, nº. 3, p. 1055–1069, 2018.

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

[47] A. Tahir, F. Quesnel, G. Desaulniers, I. El Hallaoui et Y. Yaakoubi, "An improved integral column generation algorithm using machine learning for aircrew pairing," *Transportation Science*, vol. 55, nº. 6, p. 1411–1429, 2021.

[48] F. Quesnel, A. Wu, G. Desaulniers et F. Soumis, "Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering," *Computers & Operations Research*, vol. 138, p. 105554, 2022.

[49] E. B. Khalil, P. L. Bodic, L. Song, G. Nemhauser et B. Dilkina, "Learning to branch in mixed integer programming," dans *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 724–731.

[50] A. Alvarez, Q. Louveaux et L. Wehenkel, "A machine learning-based approximation of strong branching," *INFORMS Journal on Computing*, vol. 29, p. 185–195, 01 2017.

[51] M. Gasse, D. Chételat, N. Ferroni, L. Charlin et A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks." arXiv preprint arXiv :1906.01629, 2019.

[52] H. He, H. Daume III et J. M. Eisner, "Learning to search in branch and bound algorithms," dans *Advances in Neural Information Processing Systems*, Z. Ghahramani,

M. Welling, C. Cortes, N. Lawrence et K. Weinberger, édit., vol. 27. Curran Associates, Inc., 2014.

[53] Y. Tang, S. Agrawal et Y. Faenza, "Reinforcement learning for integer programming : Learning to cut," dans *Proceedings of the 37th International Conference on Machine Learning*, vol. 119. PMLR, 2020, p. 9367–9376.

[54] C. K. Joshi, T. Laurent et X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem." arXiv preprint arXiv :1906.01227, 2019.

[55] I. Bello, H. Pham, Q. V. Le, M. Norouzi et S. Bengio, "Neural combinatorial optimization with reinforcement learning." arXiv preprint arXiv :1611.09940, 2016.

[56] W. Kool, H. van Hoof et M. Welling, "Attention, learn to solve routing problems !" arXiv preprint arXiv :1803.08475, 2018.

[57] A. Bogyrbayeva, M. Meraliyev, T. Mustakhov et B. Dauletbayev, "Learning to solve vehicle routing problems : A survey." arXiv preprint arXiv :2205.02453, 2022.

[58] A. S. Xavier, F. Qiu et S. Ahmed, "Learning to solve large-scale security-constrained unit commitment problems," *INFORMS Journal on Computing*, vol. 33, n°. 2, p. 739–756, 2021.

[59] A. Lodi, L. Mossina et E. Rachelson, "Learning to handle parameter perturbations in combinatorial optimization : An application to facility location," *EURO Journal on Transportation and Logistics*, vol. 9, n°. 4, p. 100023, 2020.

[60] J. Gondzio, P. González-Brevis et P. Munari, "Large-scale optimization with the primal-dual column generation method," *Mathematical Programming Computation*, vol. 8, n°. 1, p. 47–82, 2016.

[61] I. Elhallaoui, G. Desaulniers, A. Metrane et F. Soumis, "Bi-dynamic constraint aggregation and subproblem reduction," *Computers & Operations Research*, vol. 35, n°. 5, p. 1713–1724, 2008.

[62] A. Lodi et G. Zarpellon, "On learning and branching : a survey," *TOP*, vol. 25, n°. 2, p. 207–236, Jul 2017.

[63] M. Ball, L. Bodin et R. Dial, "A matching based heuristic for scheduling mass transit crews and vehicles," *Transportation Science*, vol. 17, n°. 1, p. 4–31, 1983.

[64] R. Freling, A. P. M. Wagelmans et J. M. P. Paixão, "An overview of models and techniques for integrating vehicle and crew scheduling," dans *Computer-Aided Transit Scheduling*, N. H. M. Wilson, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 1999, p. 441–460.

[65] R. Freling, D. Huisman et A. P. M. Wagelmans, "Models and algorithms for integration of vehicle and crew scheduling," *Journal of Scheduling*, vol. 6, n°. 1, p. 63–85, 2003.

[66] D. Huisman, R. Freling et A. P. M. Wagelmans, "Multiple-depot integrated vehicle and crew scheduling," *Transportation Science*, vol. 39, n°. 4, p. 491–502, 2005.

[67] M. Mesquita et A. Paias, "Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem," *Computers & Operations Research*, vol. 35, n°. 5, p. 1562–1575, 2008.

[68] S. W. de Groot et D. Huisman, "Vehicle and crew scheduling : Solving large real-world instances with an integrated approach," dans *Computer-aided Systems in Public Transport*, M. Hickman, P. Mirchandani et S. Voß, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, p. 43–56.

[69] M. Desrochers, J. Desrosiers et M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Operations Research*, vol. 40, n°. 2, p. 342–354, 1992.

[70] N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon et F. Soumis, "2-path cuts for the vehicle routing problem with time windows," *Transportation Science*, vol. 33, n°. 1, p. 101–116, 1999.

[71] M. Jepsen, B. Petersen, S. Spoorendonk et D. Pisinger, "Subset-row inequalities applied to the vehicle-routing problem with time windows," *Operations Research*, vol. 56, n°. 2, p. 497–511, 2008.

[72] G. Desaulniers, O. B. G. Madsen et S. Ropke, "The vehicle routing problem with time windows," dans *Vehicle Routing : Problems, Methods and Applications*, 2e éd., P. Toth et D. Vigo, édit. Philadelphia, PA : Society for Industrial and Applied Mathematics, 2014, ch. 5, p. 119–159.

[73] J. Homberger et H. Gehring, "A two-phase hybrid metaheuristic for the vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 162, n°. 1, p. 220–238, 2005.

[74] M. Desrochers et F. Soumis, "A generalized permanent labeling algorithm for the shortest path problem with time windows," *Information Systems and Operations Research*, vol. 26, n°. 3, p. 191–212, 1988.

[75] G. Desaulniers, J. Desrosiers, I. loachim, M. M. Solomon, F. Soumis et D. Villeneuve, *A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems*. Boston, MA : Springer US, 1998, p. 57–93.

[76] M. Gamache, F. Soumis, G. Marquis et J. Desrosiers, “A column generation approach for large-scale aircrew rostering problems,” *Operations Research*, vol. 47, n°. 2, p. 247–263, 1999.

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

[78] J. Kotary, F. Fioretto, P. Van Hentenryck et B. Wilder, “End-to-end constrained optimization learning : A survey.” arXiv preprint arXiv :2103.16378, 2021.

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser et I. Polosukhin, “Attention is all you need.” arXiv preprint arXiv :1706.03762, 2017.

[80] M. Nazari, A. Oroojlooy, L. Snyder et M. Takac, “Reinforcement learning for solving the vehicle routing problem,” dans *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi et R. Garnett, édit., vol. 31. Curran Associates, Inc., 2018.

[81] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina et L. Song, “Learning combinatorial optimization algorithms over graphs.” arXiv preprint arXiv :1704.01665, 2017.

[82] Z. Li, Q. Chen et V. Koltun, “Combinatorial optimization with graph convolutional networks and guided tree search.” arXiv preprint arXiv :1810.10659, 2018.

[83] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu et A. Singh, “Learning heuristics over large graphs via deep reinforcement learning.” arXiv preprint arXiv :1903.03332, 2019.

[84] G. Zarpellon, J. Jo, A. Lodi et Y. Bengio, “Parameterizing branch-and-bound search trees to learn branching policies.” arXiv preprint arXiv :2002.05120, 2020.

[85] M. Kruber, M. E. Lübbecke et A. Parmentier, “Learning when to use a decomposition,” dans *Integration of AI and OR Techniques in Constraint Programming*, D. Salvagnin et M. Lombardi, édit. Cham : Springer International Publishing, 2017, p. 202–210.

[86] P. Bonami, A. Lodi et G. Zarpellon, “A classifier to decide on the linearization of mixed-integer quadratic problems in cplex,” *Operations Research*, vol. 0, n°. 0, p. 0, 2022.

[87] L. Accorsi et D. Vigo, “A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems,” *Transportation Science*, vol. 55, n°. 4, p. 832–856, 2021.

[88] G. Clarke et J. W. Wright, “Scheduling of vehicles from a central depot to a number of delivery points,” *Operations Research*, vol. 12, n°. 4, p. 568–581, 1964.

[89] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal et A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, nº. 3, p. 845–858, 2017.

[90] Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen et J. Tang, "Learning to iteratively solve routing problems with dual-aspect collaborative transformer." arXiv preprint arXiv :2110.02544, 2021.

ANNEXE A

A.1 Detailed results of the experiments

This section provides detailed computational results that are complementary to those presented in Section 6.5. The results are grouped by scenario (N_c and interval) and reported in Tables A.1-A.9. The same information as in Tables 6.4 to 6.6 is reported, with the only difference that each row represents an individual instance. Also, an additional column has been added to indicate the name suffix of the modified instance $\mathcal{P}_m$ (second column).

TABLE A.1 Results for instances with $N_c = 10$ and Small (S) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ cost	$sim(\mathcal{S}_o, \mathcal{S}_m)$	ML model metrics			Edge-fixing			Exact B&P			DACT	
				TNR	TPR	Accuracy	Cost	Time (s)	Gap	Cost	Time (s)	Ratio	Cost	Gap
	_10S_1	27538	84%	55%	67%	61%	27733	26	0.71%	27538	226	8.7	27943	1.47%
	_10S_2	27775	71%	61%	73%	67%	27849	14	0.27%	27764	199	14.2	28306	1.91%
X-n101-k25	**_10S_3**	27591	100%	100%	63%	82%	27591	7	0.00%	27591	180	25.7	28287	2.52%
	_10S_4	27703	73%	60%	76%	68%	27811	14	0.39%	27703	276	19.7	27966	0.95%
	_10S_5	27580	91%	77%	70%	74%	27605	4	0.09%	27580	175	43.8	28417	3.03%
Average	-	**27637**	**84%**	**71%**	**70%**	**70%**	**27718**	**13**	**0.29%**	**27635**	**211**	**22.4**	**28184**	**1.98%**
	_10S_1	26403	77%	74%	64%	69%	26600	58	0.75%	26403	1567	27.0	26977	2.17%
	_10S_2	26398	85%	85%	59%	72%	26478	46	0.30%	26398	1246	27.1	26860	1.75%
X-n106-k14	**_10S_3**	26365	94%	100%	56%	78%	26365	8	0.00%	26365	589	73.6	26957	2.25%
	_10S_4	26359	89%	92%	60%	76%	26366	8	0.03%	26359	1250	156.3	26904	2.07%
	_10S_5	26353	82%	95%	63%	79%	26369	7	0.06%	26353	157	22.4	26789	1.65%
Average	-	**26376**	**85%**	**89%**	**60%**	**75%**	**26436**	**25**	**0.23%**	**26376**	**962**	**61.3**	**26897**	**1.98%**
	_10S_1	14971	95%	100%	64%	82%	14971	11	0.00%	14971	141	12.8	15102	0.88%
	_10S_2	15006	95%	100%	64%	82%	15006	6	0.00%	15006	241	40.2	15275	1.79%
X-n110-k13	**_10S_3**	14996	92%	100%	68%	84%	14996	6	0.00%	14996	338	56.3	15212	1.44%
	_10S_4	14965	87%	100%	70%	85%	14965	7	0.00%	14965	215	30.7	15076	0.74%
	_10S_5	14996	96%	100%	66%	83%	14996	9	0.00%	14996	389	43.2	15130	0.89%
Average	-	**14987**	**93%**	**100%**	**66%**	**83%**	**14987**	**8**	**0.00%**	**14987**	**265**	**36.7**	**15159**	**1.15%**
	_10S_1	55573	67%	68%	68%	68%	55715	71	0.26%	55573	1044	14.7	58563	5.38%
	_10S_2	55784	57%	77%	76%	77%	55786	788	0.004%	55817	>5h	-	58883	5.56%
X-n125-k30	**_10S_3**	55653	65%	80%	74%	77%	55654	169	0.002%	55638	3161	18.7	58609	5.31%
	_10S_4	55487	69%	79%	70%	75%	55661	84	0.31%	55487	3243	38.6	58473	5.38%
	_10S_5	55567	57%	76%	77%	77%	55599	64	0.06%	55530	1927	30.1	58379	5.06%
Average	-	**55613**	**63%**	**76%**	**73%**	**75%**	**55683**	**235**	**0.13%**	**-**	**-**	**-**	**58581**	**5.34%**
	_10S_1	28981	59%	77%	75%	76%	29472	129	1.69%	28981	2793	21.7	30015	3.57%
	_10S_2	29061	62%	81%	74%	78%	29257	81	0.67%	29061	1034	12.8	30055	3.42%
X-n129-k18	**_10S_3**	28644	58%	70%	68%	69%	28781	74	0.48%	28643	3124	42.2	29630	3.44%
	_10S_4	28598	70%	81%	67%	74%	28716	56	0.41%	28598	9179	163.9	29265	2.33%
	_10S_5	28541	60%	82%	77%	80%	28685	48	0.50%	28541	1263	26.3	29518	3.42%
Average	-	**28765**	**62%**	**78%**	**72%**	**75%**	**28982**	**78**	**0.75%**	**28765**	**3479**	**53.4**	**29697**	**3.24%**
	_10S_1	10906	89%	87%	52%	70%	10913	37	0.06%	-	>5h	-	11416	4.68%
	_10S_2	10916	100%	100%	45%	73%	10916	50	0.00%	-	>5h	-	11274	3.28%
X-n134-k13	**_10S_3**	10809	54%	75%	62%	69%	10929	190	1.11%	-	>5h	-	11205	3.66%
	_10S_4	10892	56%	85%	67%	76%	10913	72	0.19%	-	>5h	-	11293	3.68%
	_10S_5	10916	100%	100%	42%	71%	10916	37	0.00%	10950	>5h	-	11234	2.91%
Average	-	**10888**	**80%**	**89%**	**54%**	**72%**	**10917**	**77**	**0.27%**	**-**	**-**	**-**	**11284**	**3.64%**
	_10S_1	13586	93%	100%	76%	88%	13586	34	0.00%	13586	10804	317.8	13753	1.23%
	_10S_2	13605	89%	100%	81%	91%	13605	17	0.00%	13609	>5h	-	13869	1.94%
X-n139-k10	**_10S_3**	13607	85%	100%	81%	91%	13607	35	0.00%	-	>5h	-	13945	2.48%
	_10S_4	13611	91%	100%	80%	90%	13611	19	0.00%	13611	17189	904.7	13820	1.54%
	_10S_5	13542	67%	60%	86%	73%	13586	24	0.32%	-	>5h	-	13841	2.21%
Average	-	**13590**	**85%**	**92%**	**81%**	**86%**	**13599**	**26**	**0.06%**	**-**	**-**	**-**	**13846**	**1.88%**
	_10S_1	15716	85%	100%	88%	94%	15716	21	0.00%	-	>5h	-	16347	4.02%
	_10S_2	15726	85%	85%	85%	85%	15738	41	0.08%	-	>5h	-	16237	3.25%
X-n143-k07	**_10S_3**	15733	83%	87%	86%	87%	15736	117	0.02%	-	>5h	-	16063	2.10%
	_10S_4	15730	68%	62%	93%	78%	15734	50	0.03%	-	>5h	-	16243	3.26%
	_10S_5	15707	83%	100%	89%	95%	15707	41	0.00%	-	>5h	-	16334	3.99%
Average	-	**15722**	**81%**	**87%**	**88%**	**88%**	**15726**	**54**	**0.02%**	**-**	**-**	**-**	**16245**	**3.32%**

TABLE A.2 Results for instances with $N_c = 10$ and Medium (M) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ cost	$sim(\mathcal{S}_o, \mathcal{S}_m)$	ML model metrics			Edge-fixing			Exact B&P			DACT	
				TNR	TPR	Accuracy	Cost	Time (s)	Gap	Cost	Time (s)	Ratio	Cost	Gap
	_10M_1	27722	88%	83%	63%	73%	27969	36	0.89%	27722	544	15.1	28417	2.51%
	_10M_2	27636	76%	72%	66%	69%	27841	19	0.74%	27636	246	12.9	28156	1.88%
X-n101-k25	_10M_3	27411	88%	67%	61%	64%	27482	19	0.26%	27411	370	19.5	28143	2.67%
	_10M_4	27655	69%	61%	65%	63%	27784	10	0.47%	27655	117	11.7	28286	2.28%
	_10M_5	27605	96%	100%	59%	80%	27605	8	0.00%	27605	343	42.9	28489	3.20%
Average	-	**27606**	**83%**	**77%**	**63%**	**70%**	**27736**	**18**	**0.47%**	**27606**	**324**	**20.4**	**28298**	**2.51%**
	_10M_1	26399	56%	90%	79%	85%	26433	19	0.13%	26399	684	36.0	26882	1.83%
	_10M_2	26341	63%	97%	76%	87%	26370	20	0.11%	26341	368	18.4	26898	2.11%
X-n106-k14	_10M_3	26340	61%	96%	78%	87%	26374	14	0.13%	26340	210	15.0	26853	1.95%
	_10M_4	26381	60%	93%	78%	86%	26392	13	0.04%	26381	424	32.6	26796	1.57%
	_10M_5	26330	88%	100%	55%	78%	26330	5	0.00%	26330	88	17.6	26928	2.27%
Average	-	**26358**	**66%**	**95%**	**73%**	**84%**	**26380**	**14**	**0.08%**	**26358**	**355**	**23.9**	**26871**	**1.95%**
	_10M_1	14899	82%	95%	69%	82%	14910	13	0.07%	14889	441	33.9	14972	0.49%
	_10M_2	15032	82%	95%	69%	82%	15131	14	0.66%	15032	502	35.9	15197	1.10%
X-n110-k13	_10M_3	15018	91%	100%	64%	82%	15018	15	0.00%	15018	106	7.1	15187	1.13%
	_10M_4	14933	86%	100%	68%	84%	14933	11	0.00%	14933	131	11.9	15223	1.94%
	_10M_5	14971	95%	100%	63%	82%	14971	7	0.00%	14971	237	33.9	15107	0.91%
Average	-	**14971**	**87%**	**98%**	**67%**	**82%**	**14993**	**12**	**0.15%**	**14969**	**283**	**24.5**	**15137**	**1.11%**
	_10M_1	55529	70%	79%	68%	74%	55608	79	0.14%	55529	2284	28.9	58614	5.56%
	_10M_2	55735	53%	82%	82%	82%	55807	75	0.13%	55620	7479	99.7	58916	5.71%
X-n125-k30	_10M_3	55828	68%	65%	64%	65%	55858	56	0.05%	55828	6486	115.8	58461	4.72%
	_10M_4	55773	51%	72%	78%	75%	55690	23	-0.15%	55647	834	36.3	58579	5.03%
	_10M_5	55700	57%	74%	71%	73%	55828	755	0.23%	55650	8696	11.5	59103	6.11%
Average	-	**55713**	**60%**	**74%**	**73%**	**74%**	**55758**	**198**	**0.08%**	**55655**	**5156**	**58.4**	**58735**	**5.42%**
	_10M_1	28661	54%	72%	77%	75%	29332	67	2.34%	28661	356	5.3	29734	3.74%
	_10M_2	29035	61%	70%	74%	72%	29203	27	0.58%	29031	3817	141.4	30139	3.80%
X-n129-k18	_10M_3	29283	58%	73%	73%	73%	29559	115	0.94%	29460	>5h	-	30142	2.93%
	_10M_4	28920	64%	79%	76%	78%	28931	30	0.04%	28910	2674	89.1	29751	2.87%
	_10M_5	28409	62%	77%	75%	76%	28594	286	0.65%	28395	1290	4.5	29240	2.93%
Average	-	**28862**	**60%**	**74%**	**75%**	**75%**	**29124**	**105**	**0.91%**	**-**	**-**	**-**	**29801**	**3.26%**
	_10M_1	10891	59%	79%	77%	78%	11083	554	1.76%	-	>5h	-	11265	3.43%
	_10M_2	10952	90%	60%	55%	58%	11095	645	1.31%	-	>5h	-	11387	3.97%
X-n134-k13	_10M_3	10792	54%	71%	76%	74%	10910	20	1.09%	-	>5h	-	11254	4.28%
	_10M_4	10891	57%	79%	77%	78%	10979	165	0.81%	-	>5h	-	11265	3.43%
	_10M_5	10915	55%	80%	81%	81%	11055	214	1.28%	-	>5h	-	11349	3.98%
Average	-	**10888**	**63%**	**74%**	**73%**	**74%**	**11024**	**320**	**1.25%**	**-**	**-**	**-**	**11304**	**3.82%**
	_10M_1	13584	97%	100%	71%	86%	13584	48	0.00%	13584	>5h	-	13727	1.05%
	_10M_2	13605	89%	93%	78%	86%	13621	18	0.12%	-	>5h	-	14027	3.10%
X-n139-k10	_10M_3	13606	81%	74%	82%	78%	13610	28	0.03%	13649	>5h	-	13775	1.24%
	_10M_4	13611	91%	100%	77%	89%	13611	12	0.00%	-	>5h	-	13894	2.08%
	_10M_5	13600	93%	60%	72%	66%	13613	28	0.10%	-	>5h	-	13890	2.13%
Average	-	**13601**	**90%**	**85%**	**76%**	**81%**	**13608**	**27**	**0.05%**	**-**	**-**	**-**	**13863**	**1.92%**
	_10M_1	15724	81%	96%	84%	90%	15739	156	0.10%	-	>5h	-	16151	2.72%
	_10M_2	15747	87%	100%	80%	90%	15747	41	0.00%	-	>5h	-	16370	3.96%
X-n143-k07	_10M_3	15667	83%	100%	80%	90%	15667	43	0.00%	-	>5h	-	16150	3.08%
	_10M_4	15691	87%	100%	75%	88%	15691	41	0.00%	-	>5h	-	16193	3.20%
	_10M_5	15706	85%	90%	78%	84%	15706	36	0.00%	-	>5h	-	16136	2.74%
Average	-	**15707**	**85%**	**97%**	**79%**	**88%**	**15710**	**63**	**0.02%**	**-**	**-**	**-**	**16200**	**3.14%**

TABLE A.3 Results for instances with $N_c = 10$ and Large (L) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ **cost**	$sim(\mathcal{S}_o, \mathcal{S}_m)$	**ML model metrics**			**Edge-fixing**			**Exact B&P**			**DACT**	
				TNR	**TPR**	**Accuracy**	**Cost**	**Time (s)**	**Gap**	**Cost**	**Time (s)**	**Ratio**	**Cost**	**Gap**
	_10L_1	27781	83%	64%	70%	67%	28176	85	1.42%	27781	123	1.4	28420	2.30%
	_10L_2	27493	61%	66%	86%	76%	27786	15	1.07%	27493	154	10.3	27812	1.16%
X-n101-k25	**_10L_3**	27490	80%	78%	77%	78%	27526	15	0.13%	27490	245	16.3	28297	2.94%
	_10L_4	27775	64%	60%	78%	69%	28665	424	3.20%	27759	788	1.9	28248	1.70%
	_10L_5	27717	64%	45%	74%	60%	28472	71	2.72%	27717	683	9.6	28210	1.78%
Average	-	**27651**	**70%**	**63%**	**77%**	**70%**	**28125**	**122**	**1.71%**	**27648**	**399**	**7.9**	**28142**	**1.98%**
	_10L_1	26400	60%	89%	84%	87%	26472	22	0.27%	26400	454	20.6	26776	1.42%
	_10L_2	26463	68%	81%	70%	76%	26705	408	0.91%	26463	10289	25.2	26934	1.78%
X-n106-k14	**_10L_3**	26354	59%	88%	85%	87%	26423	146	0.26%	26354	662	4.5	26860	1.92%
	_10L_4	26515	50%	64%	72%	68%	26825	667	1.17%	26549	>5h	-	26974	1.73%
	_10L_5	26330	88%	96%	60%	78%	26358	9	0.11%	26330	95	10.6	26686	1.35%
Average	-	**26412**	**65%**	**84%**	**74%**	**79%**	**26557**	**250**	**0.54%**	**-**	**-**	**-**	**26846**	**1.64%**
	_10L_1	14945	86%	91%	66%	79%	14961	9	0.11%	14945	529	58.8	14950	0.03%
	_10L_2	15203	63%	83%	81%	82%	15314	82	0.73%	15203	8812	107.5	15313	0.72%
X-n110-k13	**_10L_3**	14992	72%	69%	69%	69%	15181	62	1.26%	14992	676	10.9	15169	1.18%
	_10L_4	15088	64%	89%	84%	87%	15157	15	0.46%	15088	1063	70.9	15473	2.55%
	_10L_5	14923	92%	100%	64%	82%	14923	6	0.00%	14923	333	55.5	15032	0.73%
Average	-	**15030**	**75%**	**86%**	**73%**	**80%**	**15107**	**35**	**0.51%**	**15030**	**2283**	**60.7**	**15187**	**1.04%**
	_10L_1	55480	59%	88%	74%	81%	55517	159	0.07%	55480	2091	13.2	58240	4.97%
	_10L_2	55409	58%	84%	74%	79%	55587	38	0.32%	55383	17979	473.1	57890	4.48%
X-n125-k30	**_10L_3**	56244	49%	76%	76%	76%	56250	543	0.01%	56143	11651	21.5	58517	4.04%
	_10L_4	55829	57%	80%	72%	76%	55909	118	0.14%	55717	5987	50.7	58796	5.31%
	_10L_5	55705	51%	79%	77%	78%	55861	1252	0.28%	55802	>5h	-	59059	6.02%
Average	-	**55733**	**55%**	**81%**	**75%**	**78%**	**55825**	**422**	**0.16%**	**-**	**-**	**-**	**58500**	**4.97%**
	_10L_1	28546	64%	64%	63%	64%	29015	165	1.64%	28546	2873	17.4	29354	2.83%
	_10L_2	29109	60%	89%	81%	85%	29245	69	0.47%	29109	5039	73.0	29824	2.46%
X-n129-k18	**_10L_3**	29042	60%	81%	75%	78%	29146	100	0.36%	29042	889	8.9	29992	3.27%
	_10L_4	28673	57%	80%	79%	80%	28839	269	0.58%	28641	3035	11.3	29441	2.68%
	_10L_5	28403	68%	76%	66%	71%	28615	251	0.75%	28403	451	1.8	29120	2.52%
Average	-	**28755**	**62%**	**78%**	**73%**	**75%**	**28972**	**171**	**0.76%**	**28748**	**2457**	**22.5**	**29546**	**2.75%**
	_10L_1	10935	80%	75%	62%	69%	10964	27	0.27%	-	>5h	-	11251	2.89%
	_10L_2	10941	54%	85%	88%	87%	10998	34	0.52%	-	>5h	-	11293	3.22%
X-n134-k13	**_10L_3**	10792	58%	64%	65%	65%	10963	307	1.58%	-	>5h	-	11129	3.12%
	_10L_4	10944	56%	78%	75%	77%	10989	298	0.41%	-	>5h	-	11297	3.23%
	_10L_5	10928	95%	57%	52%	55%	10984	152	0.51%	-	>5h	-	11363	3.98%
Average	-	**10908**	**69%**	**72%**	**68%**	**70%**	**10908**	**164**	**0.66%**	**-**	**-**	**-**	**11267**	**3.29%**
	_10L_1	13580	89%	86%	71%	79%	13581	73	0.01%	13580	13184	180.6	13859	2.05%
	_10L_2	13588	77%	95%	81%	88%	13674	58	0.63%	-	>5h	-	13870	2.08%
X-n139-k10	**_10L_3**	13595	82%	82%	77%	80%	13600	26	0.04%	13595	7712	296.6	13833	1.75%
	_10L_4	13625	76%	72%	78%	75%	13660	38	0.26%	13625	13273	349.3	13832	1.52%
	_10L_5	13612	91%	91%	70%	81%	13660	41	0.35%	-	>5h	-	13855	1.79%
Average	-	**13600**	**83%**	**85%**	**75%**	**80%**	**13635**	**47**	**0.26%**	**-**	**-**	**-**	**13850**	**1.84%**
	_10L_1	15735	89%	100%	72%	86%	15735	18	0.000%	-	>5h	-	16335	3.81%
	_10L_2	15734	87%	94%	75%	85%	15736	54	0.013%	-	>5h	-	16196	2.94%
X-n143-k07	**_10L_3**	15727	90%	100%	71%	86%	15727	60	0.000%	-	>5h	-	16175	2.85%
	_10L_4	15703	90%	92%	71%	82%	15703	40	0.000%	-	>5h	-	16020	2.02%
	_10L_5	15683	85%	100%	74%	87%	15683	139	0.000%	-	>5h	-	16598	5.83%
Average	-	**15716**	**88%**	**97%**	**73%**	**85%**	**15717**	**62**	**0.00%**	**-**	**-**	**-**	**16265**	**3.49%**

TABLE A.4 Results for instances with $N_c = 20$ and Small (S) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ **cost**	$sim(\mathcal{S}_o, \mathcal{S}_m)$	**ML model metrics**			**Edge-fixing**			**Exact B&P**			**DACT**	
				TNR	**TPR**	**Accuracy**	**Cost**	**Time (s)**	**Gap**	**Cost**	**Time (s)**	**Ratio**	**Cost**	**Gap**
	_20S_1	27524	73%	75%	82%	79%	27761	13	0.86%	27524	140	10.8	28402	3.19%
	_20S_2	27543	96%	75%	70%	73%	27543	27	0.00%	27543	328	12.1	28145	2.19%
X-n101-k25	**_20S_3**	27461	84%	87%	75%	81%	27551	26	0.33%	27461	227	8.7	28137	2.46%
	_20S_4	27522	88%	64%	69%	67%	27709	44	0.68%	27522	288	6.5	27980	1.66%
	_20S_5	27382	73%	62%	78%	70%	27575	36	0.70%	27382	171	4.8	28436	3.85%
Average	-	**27486**	**83%**	**73%**	**75%**	**74%**	**27628**	**29**	**0.51%**	**27486**	**231**	**8.6**	**28220**	**2.67%**
	_20S_1	26357	76%	92%	68%	80%	26391	11	0.13%	26357	407	37.0	26884	2.00%
	_20S_2	26318	72%	96%	72%	84%	26359	33	0.16%	26318	453	13.7	26888	2.17%
X-n106-k14	**_20S_3**	26360	74%	73%	64%	69%	26524	66	0.62%	26360	784	11.9	26897	2.04%
	_20S_4	26292	60%	91%	82%	87%	26372	31	0.30%	26292	432	13.9	26804	1.95%
	_20S_5	26361	78%	100%	69%	85%	26361	10	0.00%	26355	3660	366.0	26753	1.49%
Average	-	**26338**	**72%**	**90%**	**71%**	**81%**	**26401**	**30**	**0.24%**	**26336**	**1147**	**88.5**	**26845**	**1.93%**
	_20S_1	15039	63%	70%	77%	74%	15068	17	0.19%	15039	297	17.5	15153	0.76%
	_20S_2	14985	92%	83%	64%	74%	15033	8	0.32%	14985	208	26.0	15116	0.87%
X-n110-k13	**_20S_3**	15047	84%	100%	72%	86%	15047	7	0.00%	15047	1487	212.4	15219	1.14%
	_20S_4	14930	69%	86%	82%	84%	15024	10	0.63%	14930	114	11.4	15148	1.46%
	_20S_5	14899	79%	92%	76%	84%	14946	4	0.32%	14899	41	10.3	15122	1.50%
Average	-	**14980**	**77%**	**86%**	**74%**	**80%**	**15024**	**9**	**0.29%**	**14980**	**429**	**55.5**	**15152**	**1.15%**
	_20S_1	55528	62%	80%	67%	74%	55589	189	0.11%	-	>5h	-	58525	5.40%
	_20S_2	55486	57%	86%	76%	81%	55546	387	0.11%	55481	3074	7.9	58331	5.13%
X-n125-k30	**_20S_3**	55688	57%	81%	71%	76%	55624	282	-0.11%	55612	11840	42.0	58771	5.54%
	_20S_4	55371	74%	87%	60%	74%	55394	154	0.04%	-	>5h	-	58010	4.77%
	_20S_5	55390	65%	92%	68%	80%	55390	151	0.00%	-	>5h	-	58221	5.11%
Average	-	**55493**	**63%**	**85%**	**68%**	**77%**	**55509**	**233**	**0.03%**	**-**	**-**	**-**	**58372**	**5.19%**
	_20S_1	28574	60%	74%	78%	76%	28675	28	0.35%	28523	1670	59.6	29357	2.74%
	_20S_2	29397	55%	72%	79%	76%	29932	354	1.82%	29391	6230	17.6	30012	2.09%
X-n129-k18	**_20S_3**	29296	57%	66%	77%	72%	29868	172	1.95%	29296	1894	11.0	30056	2.59%
	_20S_4	29127	57%	76%	76%	76%	29587	108	1.58%	29126	6979	64.6	29886	2.61%
	_20S_5	28708	52%	66%	79%	73%	28977	53	0.94%	28708	12182	229.8	29204	1.73%
Average	-	**29020**	**56%**	**71%**	**78%**	**74%**	**29408**	**143**	**1.33%**	**29009**	**5791**	**76.5**	**29703**	**2.35%**
	_20S_1	10916	57%	85%	79%	82%	11023	526	0.98%	-	>5h	-	11344	3.92%
	_20S_2	10918	86%	80%	59%	70%	10934	40	0.15%	-	>5h	-	11256	3.10%
X-n134-k13	**_20S_3**	10899	63%	87%	76%	82%	10929	96	0.28%	-	>5h	-	11324	3.90%
	_20S_4	10901	65%	92%	75%	84%	10916	39	0.14%	10901	>5h	-	11315	3.80%
	_20S_5	10909	92%	81%	55%	68%	10913	49	0.04%	-	>5h	-	11377	4.29%
Average	-	**10909**	**73%**	**85%**	**69%**	**77%**	**10943**	**150**	**0.32%**	**-**	**-**	**-**	**11323**	**3.80%**
	_20S_1	13605	85%	81%	81%	81%	13605	60	0.00%	-	>5h	-	13815	1.54%
	_20S_2	13624	89%	100%	80%	90%	13624	36	0.00%	13624	6925	192.4	13819	1.43%
X-n139-k10	**_20S_3**	13602	91%	92%	78%	85%	13607	197	0.04%	13595	13074	66.4	13856	1.87%
	_20S_4	13624	86%	100%	82%	91%	13624	87	0.00%	-	>5h	-	13961	2.47%
	_20S_5	13608	83%	82%	82%	82%	13619	135	0.08%	13608	11745	87.0	13897	2.12%
Average	-	**13613**	**87%**	**91%**	**81%**	**86%**	**13616**	**103**	**0.02%**	**-**	**-**	**-**	**13870**	**1.89%**
	_20S_1	15708	87%	100%	89%	95%	15708	38	0.00%	-	>5h	-	16083	2.39%
	_20S_2	15733	87%	100%	88%	94%	15733	13	0.00%	-	>5h	-	16231	3.17%
X-n143-k07	**_20S_3**	15730	93%	100%	82%	91%	15730	26	0.00%	-	>5h	-	16158	2.72%
	_20S_4	15722	93%	100%	82%	91%	15722	56	0.00%	-	>5h	-	16386	4.22%
	_20S_5	15684	69%	50%	88%	69%	15845	16	1.03%	-	>5h	-	16449	4.88%
Average	-	**15715**	**86%**	**90%**	**86%**	**88%**	**15748**	**30**	**0.21%**	**-**	**-**	**-**	**16261**	**3.47%**

TABLE A.5 Results for instances with $N_c = 20$ and Medium (M) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ cost	$sim(\mathcal{S}_o, \mathcal{S}_m)$	ML model metrics			Edge-fixing			Exact B&P			DACT	
				TNR	TPR	Accuracy	Cost	Time (s)	Gap	Cost	Time (s)	Ratio	Cost	Gap
	_20M_1	27709	63%	77%	75%	76%	28105	47	1.43%	27709	550	11.7	28385	2.44%
	_20M_2	27440	65%	68%	73%	71%	27646	64	0.75%	27440	615	9.6	27696	0.93%
X-n101-k25	**_20M_3**	27530	70%	81%	72%	77%	27559	24	0.11%	27530	983	41.0	28193	2.41%
	_20M_4	27656	74%	70%	65%	68%	27767	11	0.40%	27656	217	19.7	28413	2.74%
	_20M_5	27226	67%	76%	75%	76%	27393	15	0.61%	27226	223	14.9	27694	1.72%
Average	-	**27512**	**68%**	**74%**	**72%**	**73%**	**27694**	**32**	**0.66%**	**27512**	**518**	**19.4**	**28076**	**2.05%**
	_20M_1	26362	69%	87%	67%	77%	26467	28	0.40%	26362	16266	580.9	26888	2.00%
	_20M_2	26345	72%	95%	68%	82%	26437	24	0.35%	26345	1522	63.4	26829	1.84%
X-n106-k14	**_20M_3**	26282	65%	87%	72%	80%	26442	17	0.61%	26282	2024	119.1	26760	1.82%
	_20M_4	26328	60%	82%	72%	77%	26353	18	0.09%	26328	4543	252.4	26644	1.20%
	_20M_5	26212	55%	82%	78%	80%	26299	11	0.33%	26212	1928	175.3	26667	1.74%
Average	-	**26306**	**64%**	**87%**	**71%**	**79%**	**26400**	**20**	**0.36%**	**26306**	**5257**	**238.2**	**26758**	**1.72%**
	_20M_1	14943	76%	74%	73%	74%	15053	8	0.74%	14943	167	20.9	15164	1.48%
	_20M_2	15046	64%	79%	80%	80%	15102	30	0.37%	15046	1616	53.9	15197	1.00%
X-n110-k13	**_20M_3**	14904	78%	96%	75%	86%	14981	6	0.52%	14904	189	31.5	15131	1.52%
	_20M_4	15018	65%	84%	84%	84%	15237	82	1.46%	15018	363	4.4	15184	1.11%
	_20M_5	15004	82%	95%	75%	85%	15018	10	0.09%	15004	407	40.7	15408	2.69%
Average	-	**14983**	**73%**	**86%**	**77%**	**82%**	**15078**	**27**	**0.64%**	**14983**	**548**	**30.3**	**15217**	**1.56%**
	_20M_1	55433	56%	85%	74%	80%	55510	635	0.14%	55668	>5h	-	58555	5.63%
	_20M_2	55593	48%	81%	80%	81%	55649	150	0.10%	55563	3534	23.6	58829	5.82%
X-n125-k30	**_20M_3**	55966	55%	81%	73%	77%	56046	87	0.14%	55918	15876	182.5	58710	4.90%
	_20M_4	55251	66%	82%	63%	73%	55288	214	0.07%	55147	12934	60.4	58736	6.31%
	_20M_5	55272	47%	80%	79%	80%	55400	80	0.23%	-	>5h	-	57810	4.59%
Average	-	**55503**	**54%**	**82%**	**74%**	**78%**	**55579**	**233**	**0.14%**	**-**	**-**	**-**	**58528**	**5.45%**
	_20M_1	29301	53%	76%	80%	78%	29697	189	1.35%	29301	2064	10.9	30306	3.43%
	_20M_2	29203	61%	75%	71%	73%	29504	325	1.03%	29169	4528	13.9	29736	1.83%
X-n129-k18	**_20M_3**	29368	60%	69%	73%	71%	29976	228	2.07%	29653	>5h	-	30326	3.26%
	_20M_4	29426	48%	74%	84%	79%	30065	705	2.17%	29412	8398	11.9	30069	2.19%
	_20M_5	28556	62%	71%	71%	71%	29035	129	1.68%	28556	4177	32.4	29568	3.54%
Average	-	**29171**	**57%**	**73%**	**76%**	**74%**	**29655**	**315**	**1.66%**	**-**	**-**	**-**	**30001**	**2.85%**
	_20M_1	10898	56%	84%	79%	82%	10939	38	0.38%	-	>5h	-	11409	4.69%
	_20M_2	10942	56%	88%	81%	85%	10974	360	0.29%	-	>5h	-	11238	2.71%
X-n134-k13	**_20M_3**	10930	56%	92%	90%	91%	11024	137	0.86%	11004	>5h	-	11280	3.20%
	_20M_4	10904	52%	73%	76%	75%	10925	30	0.19%	-	>5h	-	11398	4.53%
	_20M_5	10712	54%	72%	74%	73%	10916	72	1.90%	-	>5h	-	10967	2.38%
Average	-	**10877**	**55%**	**82%**	**80%**	**81%**	**10956**	**127**	**0.73%**	**-**	**-**	**-**	**11258**	**3.50%**
	_20M_1	13545	70%	80%	80%	80%	13584	187	0.29%	13545	12506	66.9	13811	1.96%
	_20M_2	13618	70%	90%	83%	87%	13630	107	0.09%	13618	6516	60.9	13783	1.21%
X-n139-k10	**_20M_3**	13591	66%	93%	89%	91%	13605	117	0.10%	-	>5h	-	13883	2.15%
	_20M_4	13648	72%	87%	79%	83%	13690	320	0.31%	-	>5h	-	13820	1.26%
	_20M_5	13621	82%	86%	71%	79%	13648	295	0.20%	-	>5h	-	13976	2.61%
Average	-	**13605**	**72%**	**87%**	**80%**	**84%**	**13631**	**205**	**0.20%**	**-**	**-**	**-**	**13855**	**1.84%**
	_20M_1	15722	92%	100%	77%	89%	15722	13	0.00%	-	>5h	-	16158	2.77%
	_20M_2	15747	89%	93%	81%	87%	15768	11	0.13%	-	>5h	-	16183	2.77%
X-n143-k07	**_20M_3**	15709	73%	71%	86%	79%	15774	99	0.41%	-	>5h	-	16382	4.28%
	_20M_4	15745	77%	64%	82%	73%	15745	47	0.00%	-	>5h	-	16254	3.23%
	_20M_5	15616	73%	74%	88%	81%	15717	77	0.65%	-	>5h	-	16189	3.67%
Average	-	**15708**	**81%**	**80%**	**83%**	**82%**	**15745**	**49**	**0.24%**	**-**	**-**	**-**	**16233**	**3.35%**

TABLE A.6 Results for instances with $N_c = 20$ and Large (L) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ **cost**	$sim(\mathcal{S}_o, \mathcal{S}_m)$	**ML model metrics**			**Edge-fixing**			**Exact B&P**			**DACT**	
				TNR	**TPR**	**Accuracy**	**Cost**	**Time (s)**	**Gap**	**Cost**	**Time (s)**	**Ratio**	**Cost**	**Gap**
	_20L_1	27463	67%	70%	63%	67%	27637	107	0.63%	27463	374	3.5	28221	2.76%
	_20L_2	27507	66%	69%	70%	70%	27641	93	0.49%	27507	196	2.1	28158	2.37%
X-n101-k25	**_20L_3**	27789	51%	76%	85%	81%	28245	50	1.64%	27789	100	2.0	28369	2.09%
	_20L_4	27837	53%	65%	76%	71%	28097	7	0.93%	27837	707	101.0	28276	1.58%
	_20L_5	27631	59%	70%	73%	72%	27733	21	0.37%	27631	955	45.5	28194	2.04%
Average	-	**27645**	**59%**	**70%**	**73%**	**72%**	**27871**	**56**	**0.81%**	**27645**	**466**	**30.8**	**28244**	**2.17%**
	_20L_1	26348	66%	87%	76%	82%	26420	25	0.27%	-	>5h	-	26759	1.56%
	_20L_2	26599	55%	72%	73%	73%	26792	171	0.73%	26593	12168	71.2	26907	1.15%
X-n106-k14	**_20L_3**	26405	65%	85%	74%	80%	26655	97	0.95%	26387	5847	60.3	26853	1.70%
	_20L_4	26358	57%	79%	77%	78%	26520	199	0.61%	26358	401	2.0	26871	1.95%
	_20L_5	26357	62%	74%	71%	73%	26390	8	0.13%	-	>5h	-	26782	1.61%
Average	-	**26413**	**61%**	**79%**	**74%**	**77%**	**26555**	**100**	**0.54%**	**-**	**-**	**-**	**26834**	**1.59%**
	_20L_1	15011	77%	75%	69%	72%	15035	15	0.16%	15011	231	15.4	15234	1.49%
	_20L_2	15003	76%	86%	73%	80%	15226	9	1.49%	15003	386	42.0	15160	1.05%
X-n110-k13	**_20L_3**	15062	59%	78%	83%	81%	15262	18	1.33%	15062	307	17.1	15291	1.52%
	_20L_4	15027	63%	76%	83%	80%	15150	49	0.82%	15027	351	7.2	15193	1.10%
	_20L_5	15066	65%	88%	85%	87%	15131	26	0.43%	15066	1162	44.7	15203	0.91%
Average	-	**15034**	**68%**	**81%**	**79%**	**80%**	**15161**	**23**	**0.84%**	**15034**	**487**	**25.4**	**15216**	**1.21%**
	_20L_1	55584	57%	73%	69%	71%	55824	997	0.43%	-	>5h	-	57981	4.32%
	_20L_2	56407	56%	85%	76%	81%	56870	379	0.82%	56407	2428	6.4	59339	5.20%
X-n125-k30	**_20L_3**	55903	65%	83%	68%	76%	55898	181	-0.01%	-	>5h	-	59042	5.62%
	_20L_4	56451	61%	76%	63%	70%	56680	508	0.41%	-	>5h	-	58746	4.07%
	_20L_5	55445	53%	76%	73%	75%	55567	100	0.22%	-	>5h	-	58035	4.67%
Average	-	**55958**	**58%**	**79%**	**70%**	**74%**	**56168**	**433**	**0.37%**	**-**	**-**	**-**	**58629**	**4.77%**
	_20L_1	29249	54%	81%	74%	78%	29428	286	0.61%	29149	1455	5.1	30263	3.47%
	_20L_2	29468	64%	75%	63%	69%	30008	380	1.83%	29468	1557	4.1	30198	2.48%
X-n129-k18	**_20L_3**	29761	52%	82%	81%	82%	30170	306	1.37%	29709	6152	20.1	30198	1.47%
	_20L_4	28504	50%	70%	76%	73%	28766	59	0.92%	28501	2751	46.6	29405	3.16%
	_20L_5	28635	54%	78%	74%	76%	28940	900	1.07%	28635	3614	4.0	29177	1.89%
Average	-	**29123**	**55%**	**77%**	**74%**	**75%**	**29462**	**386**	**1.16%**	**29092**	**3106**	**16.0**	**29848**	**2.49%**
	_20L_1	10934	56%	86%	83%	85%	11030	181	0.88%	-	>5h	-	11370	3.99%
	_20L_2	10922	55%	86%	85%	86%	11073	622	1.38%	-	>5h	-	11343	3.85%
X-n134-k13	**_20L_3**	10987	54%	85%	82%	84%	11132	134	1.32%	-	>5h	-	11331	3.13%
	_20L_4	10965	54%	81%	80%	81%	11048	238	0.76%	-	>5h	-	11276	2.84%
	_20L_5	10736	56%	74%	70%	72%	10971	96	2.19%	10736	12945	134.8	11264	4.92%
Average	-	**10909**	**55%**	**82%**	**80%**	**81%**	**11051**	**254**	**1.31%**	**-**	**-**	**-**	**11317**	**3.75%**
	_20L_1	13583	84%	87%	68%	78%	13597	154	0.10%	13583	16053	104.2	13798	1.58%
	_20L_2	13582	73%	87%	79%	83%	13612	36	0.22%	13580	4350	120.8	13785	1.49%
X-n139-k10	**_20L_3**	13579	81%	100%	72%	86%	13597	33	0.13%	13579	5595	169.5	13771	1.41%
	_20L_4	13565	63%	77%	81%	79%	13662	407	0.72%	13565	10501	25.8	13771	1.52%
	_20L_5	13614	65%	90%	87%	89%	13628	64	0.10%	13638	>5h	-	13814	1.47%
Average	-	**13585**	**73%**	**88%**	**77%**	**83%**	**13619**	**139**	**0.25%**	**-**	**-**	**-**	**13778**	**1.50%**
	_20L_1	15799	88%	91%	78%	85%	15814	23	0.09%	-	>5h	-	16471	4.25%
	_20L_2	15705	89%	75%	76%	76%	15715	76	0.06%	-	>5h	-	16193	3.11%
X-n143-k07	**_20L_3**	15815	71%	52%	82%	67%	15886	19	0.45%	-	>5h	-	16432	3.90%
	_20L_4	15816	82%	78%	81%	80%	15879	74	0.40%	-	>5h	-	16373	3.52%
	_20L_5	15580	89%	100%	79%	90%	15580	13	0.00%	-	>5h	-	16190	3.92%
Average	-	**15743**	**84%**	**79%**	**79%**	**79%**	**15775**	**41**	**0.20%**	**-**	**-**	**-**	**16332**	**3.74%**

TABLE A.7 Results for instances with $N_c = 30$ and Small (S) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ **cost**	$sim(\mathcal{S}_o, \mathcal{S}_m)$	**ML model metrics**			**Edge-fixing**			**Exact B&P**			**DACT**	
				TNR	**TPR**	**Accuracy**	**Cost**	**Time (s)**	**Gap**	**Cost**	**Time (s)**	**Ratio**	**Cost**	**Gap**
	_30S_1	27645	73%	79%	69%	74%	27759	40	0.41%	27645	247	6.2	28109	1.68%
	_30S_2	27709	71%	72%	72%	72%	27838	33	0.47%	27709	254	7.7	28579	3.14%
X-n101-k25	**_30S_3**	27485	78%	77%	66%	72%	27606	89	0.44%	27485	256	2.9	28104	2.25%
	_30S_4	27620	84%	85%	66%	76%	27712	8	0.33%	27620	47	5.9	28422	2.90%
	_30S_5	27350	68%	87%	79%	83%	27431	13	0.30%	27350	87	6.7	27883	1.95%
Average	-	**27562**	**75%**	**80%**	**70%**	**75%**	**27669**	**37**	**0.39%**	**27562**	**178**	**5.9**	**28219**	**2.38%**
	_30S_1	26346	64%	94%	73%	84%	26433	19	0.33%	26346	84	4.4	26900	2.10%
	_30S_2	26440	60%	80%	67%	74%	26457	239	0.06%	26419	5459	22.8	26932	1.86%
X-n106-k14	**_30S_3**	26424	60%	85%	71%	78%	26573	422	0.56%	26423	903	2.1	26787	1.37%
	_30S_4	26389	73%	88%	61%	75%	26400	53	0.04%	26386	752	14.2	26875	1.84%
	_30S_5	26317	67%	97%	73%	85%	26326	8	0.03%	26317	689	86.1	26669	1.34%
Average	-	**26383**	**65%**	**89%**	**69%**	**79%**	**26438**	**148**	**0.21%**	**26378**	**1577**	**25.9**	**26833**	**1.70%**
	_30S_1	15054	80%	87%	72%	80%	15109	61	0.37%	15054	337	5.5	15257	1.35%
	_30S_2	15002	72%	96%	78%	87%	15092	32	0.60%	15002	56	1.8	15139	0.91%
X-n110-k13	**_30S_3**	14990	71%	92%	81%	87%	15059	47	0.46%	14990	1664	35.4	15175	1.23%
	_30S_4	14949	75%	90%	77%	84%	15123	12	1.16%	14949	304	25.3	15158	1.40%
	_30S_5	15031	72%	93%	76%	85%	15032	10	0.01%	15031	412	41.2	15057	0.17%
Average	-	**15005**	**74%**	**92%**	**77%**	**84%**	**15083**	**32**	**0.52%**	**15005**	**555**	**21.8**	**15157**	**1.01%**
	_30S_1	55714	47%	75%	78%	77%	55813	309	0.18%	55643	4647	15.0	58197	4.46%
	_30S_2	56124	56%	73%	68%	71%	56119	181	-0.01%	56037	7085	39.1	59211	5.50%
X-n125-k30	**_30S_3**	55919	46%	75%	80%	78%	56014	422	0.17%	55779	1061	2.5	59158	5.79%
	_30S_4	55619	51%	79%	74%	77%	55685	297	0.12%	55614	>5h	-	58320	4.86%
	_30S_5	55504	64%	84%	66%	75%	55539	86	0.06%	55674	>5h	-	58117	4.71%
Average	-	**55776**	**53%**	**77%**	**73%**	**75%**	**55834**	**259**	**0.10%**	**-**	**-**	**-**	**58601**	**5.06%**
	_30S_1	29566	57%	76%	72%	74%	30061	213	1.67%	29533	2806	13.2	30314	2.53%
	_30S_2	29771	63%	80%	69%	75%	30035	43	0.89%	29771	1578	36.7	30305	1.79%
X-n129-k18	**_30S_3**	29588	57%	82%	75%	79%	29740	82	0.51%	29588	3659	44.6	30372	2.65%
	_30S_4	28892	61%	83%	74%	79%	29356	500	1.61%	28892	1303	2.6	29612	2.49%
	_30S_5	29253	52%	81%	80%	81%	29700	346	1.53%	29240	9362	27.1	29825	1.96%
Average	-	**29414**	**58%**	**80%**	**74%**	**77%**	**29778**	**237**	**1.24%**	**29405**	**3742**	**24.8**	**30086**	**2.28%**
	_30S_1	10925	89%	93%	53%	73%	10925	30	0.00%	-	>5h	-	11333	3.73%
	_30S_2	10962	86%	80%	53%	67%	10973	53	0.10%	-	>5h	-	11409	4.08%
X-n134-k13	**_30S_3**	10960	56%	87%	79%	83%	11012	211	0.47%	-	>5h	-	11365	3.70%
	_30S_4	10874	60%	88%	75%	82%	10949	366	0.69%	-	>5h	-	11294	3.86%
	_30S_5	10903	95%	100%	51%	76%	10903	42	0.00%	-	>5h	-	11406	4.61%
Average	-	**10925**	**77%**	**90%**	**62%**	**76%**	**10952**	**140**	**0.25%**	**-**	**-**	**-**	**11361**	**4.00%**
	_30S_1	13610	91%	91%	64%	78%	13722	127	0.82%	13610	13075	103.0	13831	1.62%
	_30S_2	13655	81%	83%	72%	78%	13835	351	1.32%	13655	3380	9.6	14047	2.87%
X-n139-k10	**_30S_3**	13623	89%	87%	67%	77%	13659	204	0.26%	13623	7294	35.8	13730	0.79%
	_30S_4	13621	77%	97%	81%	89%	13632	40	0.08%	-	>5h	-	13724	0.76%
	_30S_5	13599	79%	96%	79%	88%	13610	48	0.08%	-	>5h	-	13812	1.57%
Average	-	**13622**	**83%**	**91%**	**73%**	**82%**	**13692**	**154**	**0.51%**	**-**	**-**	**-**	**13829**	**1.52%**
	_30S_1	15738	86%	100%	89%	95%	15738	9	0.00%	-	>5h	-	16338	3.81%
	_30S_2	15822	85%	83%	87%	85%	15855	216	0.21%	-	>5h	-	16463	4.05%
X-n143-k07	**_30S_3**	15825	89%	81%	84%	83%	15972	34	0.93%	-	>5h	-	16429	3.82%
	_30S_4	15707	91%	83%	82%	83%	15712	9	0.03%	-	>5h	-	16243	3.41%
	_30S_5	15677	65%	50%	92%	71%	15833	14	1.00%	-	>5h	-	16169	3.14%
Average	-	**15754**	**83%**	**79%**	**87%**	**83%**	**15822**	**56**	**0.43%**	**-**	**-**	**-**	**16328**	**3.65%**

TABLE A.8 Results for instances with $N_c = 30$ and Medium (M) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ cost	$sim(\mathcal{S}_o, \mathcal{S}_m)$	ML model metrics			Edge-fixing			Exact B&P			DACT	
				TNR	TPR	Accuracy	Cost	Time (s)	Gap	Cost	Time (s)	Ratio	Cost	Gap
	_30M_1	27849	57%	66%	75%	71%	28118	110	0.97%	27830	244	2.2	28118	0.97%
	_30M_2	27660	60%	67%	72%	70%	27964	66	1.10%	27660	91	1.4	28338	2.45%
X-n101-k25	**_30M_3**	27699	69%	69%	72%	71%	27718	8	0.07%	27699	89	11.1	28401	2.53%
	_30M_4	27724	60%	67%	71%	69%	27843	67	0.43%	27724	149	2.2	28388	2.40%
	_30M_5	27340	68%	70%	76%	73%	27375	6	0.13%	27340	642	107.0	28139	2.92%
Average	-	**27654**	**63%**	**68%**	**73%**	**71%**	**27804**	**51**	**0.54%**	**27651**	**243**	**24.8**	**28277**	**2.25%**
	_30M_1	26419	61%	66%	60%	63%	26526	76	0.41%	26419	599	7.9	26892	1.79%
	_30M_2	26550	52%	66%	71%	69%	26741	312	0.72%	26579	>5h	-	26804	0.96%
X-n106-k14	**_30M_3**	26437	63%	81%	74%	78%	26659	73	0.84%	26656	>5h	-	26805	1.39%
	_30M_4	26446	56%	73%	70%	72%	26731	769	1.08%	-	>5h	-	26621	0.66%
	_30M_5	26333	63%	89%	76%	83%	26419	7	0.33%	-	>5h	-	26779	1.69%
Average	-	**26437**	**59%**	**75%**	**70%**	**73%**	**26615**	**247**	**0.67%**	**-**	**-**	**-**	**26780**	**1.30%**
	_30M_1	15183	81%	71%	65%	68%	15228	8	0.30%	15183	339	42.4	15443	1.71%
	_30M_2	15075	83%	80%	66%	73%	15079	9	0.03%	15075	177	19.7	15502	2.83%
X-n110-k13	**_30M_3**	15129	70%	86%	78%	82%	15163	7	0.22%	15129	150	21.4	15346	1.43%
	_30M_4	14995	76%	96%	75%	86%	15093	41	0.65%	14995	291	7.1	15040	0.30%
	_30M_5	14910	77%	100%	76%	88%	14910	5	0.00%	14910	387	77.4	15130	1.48%
Average	-	**15058**	**77%**	**87%**	**72%**	**79%**	**15095**	**14**	**0.24%**	**15058**	**269**	**33.6**	**15292**	**1.55%**
	_30M_1	55574	52%	80%	77%	79%	55711	892	0.25%	55574	1425	1.6	58917	6.02%
	_30M_2	56218	45%	84%	84%	84%	56292	114	0.13%	56128	1280	11.2	59406	5.67%
X-n125-k30	**_30M_3**	56589	52%	80%	76%	78%	56691	685	0.18%	56314	3800	5.5	59519	5.18%
	_30M_4	55735	46%	79%	76%	78%	55793	86	0.10%	55656	5232	60.8	58574	5.09%
	_30M_5	55510	53%	85%	78%	82%	55558	476	0.09%	-	>5h	-	57880	4.27%
Average	-	**55925**	**50%**	**82%**	**78%**	**80%**	**56009**	**451**	**0.15%**	**-**	**-**	**-**	**58859**	**5.25%**
	_30M_1	29604	61%	82%	67%	75%	30105	324	1.69%	29604	5594	17.3	30360	2.55%
	_30M_2	29758	51%	80%	82%	81%	30408	91	2.18%	29758	3348	36.8	30256	1.67%
X-n129-k18	**_30M_3**	29813	54%	80%	72%	76%	30221	89	1.37%	29793	4689	52.7	30122	1.04%
	_30M_4	28682	55%	83%	77%	80%	29212	122	1.85%	28669	406	3.3	29629	3.30%
	_30M_5	28249	48%	72%	77%	75%	28545	60	1.05%	28249	3835	63.9	28878	2.23%
Average	-	**29221**	**54%**	**79%**	**75%**	**77%**	**29698**	**137**	**1.63%**	**29215**	**3574**	**34.8**	**29849**	**2.16%**
	_30M_1	10959	53%	83%	85%	84%	11039	36	0.73%	-	>5h	-	11168	1.91%
	_30M_2	10977	52%	80%	81%	81%	11118	108	1.28%	-	>5h	-	11360	3.49%
X-n134-k13	**_30M_3**	10998	65%	86%	73%	80%	11139	372	1.28%	-	>5h	-	11464	4.24%
	_30M_4	10942	58%	90%	80%	85%	11035	70	0.85%	-	>5h	-	11427	4.43%
	_30M_5	10752	52%	75%	81%	78%	10907	60	1.44%	-	>5h	-	11086	3.11%
Average	-	**10926**	**56%**	**83%**	**80%**	**81%**	**11048**	**129**	**1.12%**	**-**	**-**	**-**	**11301**	**3.43%**
	_30M_1	13631	80%	91%	76%	84%	13742	133	0.81%	13631	8248	62.0	13926	2.16%
	_30M_2	13795	74%	82%	76%	79%	13932	585	0.99%	-	>5h	-	14235	3.19%
X-n139-k10	**_30M_3**	13598	86%	95%	68%	82%	13598	57	0.00%	13598	10792	189.3	13834	1.74%
	_30M_4	13617	66%	90%	87%	89%	13628	322	0.08%	13617	7536	23.4	13927	2.28%
	_30M_5	13560	70%	87%	82%	85%	13638	184	0.58%	13560	16734	90.9	13710	1.11%
Average	-	**13640**	**75%**	**89%**	**78%**	**83%**	**13708**	**256**	**0.49%**	**-**	**-**	**-**	**13926**	**2.09%**
	_30M_1	15751	86%	100%	86%	93%	15751	7	0.00%	-	>5h	-	16292	3.43%
	_30M_2	16039	69%	63%	89%	76%	16346	199	1.91%	-	>5h	-	16741	4.38%
X-n143-k07	**_30M_3**	15798	84%	80%	83%	82%	15869	16	0.45%	-	>5h	-	16912	7.05%
	_30M_4	15690	91%	84%	79%	82%	15690	7	0.00%	-	>5h	-	16005	2.01%
	_30M_5	15630	70%	62%	90%	76%	15692	15	0.40%	-	>5h	-	16165	3.42%
Average	-	**15782**	**80%**	**78%**	**85%**	**82%**	**15870**	**49**	**0.55%**	**-**	**-**	**-**	**16423**	**4.06%**

TABLE A.9 Results for instances with $N_c = 30$ and Large (L) intervals.

$\mathcal{P}_o$	$\mathcal{P}_m$	$\mathcal{S}_m$ **cost**	$sim(\mathcal{S}_o, \mathcal{S}_m)$	**ML model metrics**			**Edge-fixing**			**Exact B&P**			**DACT**	
				TNR	**TPR**	**Accuracy**	**Cost**	**Time (s)**	**Gap**	**Cost**	**Time (s)**	**Ratio**	**Cost**	**Gap**
	_30L_1	27736	61%	76%	75%	76%	28298	47	2.03%	27736	555	11.8	28486	2.70%
	_30L_2	27465	72%	84%	71%	78%	27590	13	0.46%	27465	300	23.1	28045	2.11%
X-n101-k25	**_30L_3**	27809	66%	70%	70%	70%	28281	844	1.70%	27809	126	0.1	28438	2.26%
	_30L_4	27809	65%	70%	69%	70%	28156	12	1.25%	27809	348	29.0	28358	1.97%
	_30L_5	27267	65%	82%	73%	78%	27321	11	0.20%	27267	364	33.1	27709	1.62%
Average	-	**27617**	**66%**	**76%**	**72%**	**74%**	**27929**	**185**	**1.12%**	**27617**	**339**	**19.4**	**28207**	**2.13%**
	_30L_1	26623	47%	76%	71%	74%	26874	666	0.94%	-	>5h	-	26976	1.33%
	_30L_2	26502	58%	78%	62%	70%	26765	594	0.99%	26494	6491	10.9	26907	1.53%
X-n106-k14	**_30L_3**	26345	59%	82%	66%	74%	26430	9	0.32%	26345	74	8.2	26901	2.11%
	_30L_4	26438	44%	81%	79%	80%	26710	90	1.03%	26438	>5h	-	26932	1.87%
	_30L_5	26290	61%	78%	63%	71%	26406	18	0.44%	26268	2157	119.8	26851	2.13%
Average	-	**26440**	**54%**	**79%**	**68%**	**74%**	**26637**	**275**	**0.75%**	**-**	**-**	**-**	**26913**	**1.79%**
	_30L_1	15269	66%	74%	67%	71%	15347	13	0.51%	15269	795	61.2	15634	2.39%
	_30L_2	15145	54%	81%	85%	83%	15363	144	1.44%	15145	635	4.4	15297	1.00%
X-n110-k13	**_30L_3**	15076	69%	89%	71%	80%	15162	59	0.57%	15076	252	4.3	15286	1.39%
	_30L_4	14945	63%	80%	73%	77%	15050	201	0.70%	14945	3822	19.0	15123	1.19%
	_30L_5	15015	78%	86%	63%	75%	15134	21	0.79%	15015	94	4.5	15248	1.55%
Average	-	**15090**	**66%**	**82%**	**72%**	**77%**	**15211**	**88**	**0.80%**	**15090**	**1120**	**18.7**	**15318**	**1.51%**
	_30L_1	56435	49%	83%	74%	79%	56463	622	0.05%	-	>5h	-	58481	3.63%
	_30L_2	56372	58%	82%	65%	74%	56422	401	0.09%	-	>5h	-	59140	4.91%
X-n125-k30	**_30L_3**	56386	57%	88%	70%	79%	56210	312	-0.31%	56072	2397	7.7	58901	4.46%
	_30L_4	56432	50%	86%	73%	80%	56494	702	0.11%	-	>5h	-	59352	5.17%
	_30L_5	55514	46%	82%	72%	77%	55517	337	0.01%	55453	15245	45.2	57751	4.03%
Average	-	**56228**	**52%**	**84%**	**71%**	**78%**	**56221**	**475**	**-0.01%**	**-**	**-**	**-**	**58725**	**4.44%**
	_30L_1	29989	52%	74%	75%	75%	30723	136	2.45%	29985	4123	30.3	30941	3.17%
	_30L_2	30260	60%	75%	71%	73%	30736	72	1.57%	30260	3099	43.0	31252	3.28%
X-n129-k18	**_30L_3**	30007	53%	73%	75%	74%	30656	85	2.16%	30007	1201	14.1	30769	2.54%
	_30L_4	29356	55%	71%	74%	73%	29944	65	2.00%	29341	15778	242.7	30275	3.13%
	_30L_5	28757	71%	84%	71%	78%	28836	19	0.27%	28757	370	19.5	29535	2.71%
Average	-	**29674**	**58%**	**75%**	**73%**	**74%**	**30179**	**75**	**1.69%**	**29670**	**4914**	**69.9**	**30554**	**2.97%**
	_30L_1	11037	52%	79%	81%	80%	11125	337	0.80%	-	>5h	-	11627	5.35%
	_30L_2	10974	52%	88%	87%	88%	11045	111	0.65%	-	>5h	-	11498	4.77%
X-n134-k13	**_30L_3**	10945	58%	84%	77%	81%	11043	39	0.90%	10950	>5h	-	11346	3.66%
	_30L_4	10869	51%	78%	80%	79%	10940	76	0.65%	10868	9388	123.5	11231	3.33%
	_30L_5	10926	53%	84%	85%	85%	10994	161	0.62%	-	>5h	-	11412	4.45%
Average	-	**10950**	**53%**	**83%**	**82%**	**82%**	**11029**	**145**	**0.72%**	**-**	**-**	**-**	**11423**	**4.31%**
	_30L_1	13564	79%	93%	73%	83%	13583	95	0.14%	-	>5h	-	13854	2.14%
	_30L_2	13696	72%	92%	77%	85%	13720	139	0.18%	13696	9395	67.6	14082	2.82%
X-n139-k10	**_30L_3**	13648	77%	94%	75%	85%	13667	177	0.14%	13648	10780	60.9	13859	1.55%
	_30L_4	13580	61%	79%	82%	81%	13599	99	0.14%	13580	9597	96.9	13669	0.66%
	_30L_5	13593	78%	96%	75%	86%	13604	132	0.08%	-	>5h	-	13876	2.08%
Average	-	**13616**	**73%**	**91%**	**76%**	**84%**	**13635**	**128**	**0.14%**	**-**	**-**	**-**	**13868**	**1.85%**
	_30L_1	16065	78%	87%	85%	86%	16137	1297	0.45%	-	>5h	-	16566	3.12%
	_30L_2	16110	70%	64%	86%	75%	16273	181	1.01%	-	>5h	-	16474	2.26%
X-n143-k07	**_30L_3**	15856	85%	90%	80%	85%	15878	55	0.14%	-	>5h	-	16364	3.20%
	_30L_4	15712	91%	100%	77%	89%	15712	21	0.00%	-	>5h	-	16148	2.77%
	_30L_5	15676	69%	70%	89%	80%	15704	11	0.18%	-	>5h	-	16079	2.57%
Average	-	**15884**	**79%**	**82%**	**83%**	**83%**	**15941**	**313**	**0.36%**	**-**	**-**	**-**	**16326**	**2.79%**