POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Machine learning-driven hybrid optimization based on decision diagrams

JAIME ESTEBAN GONZALEZ JURADO
Département de mathématiques et de génie industriel

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

Août 2020

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

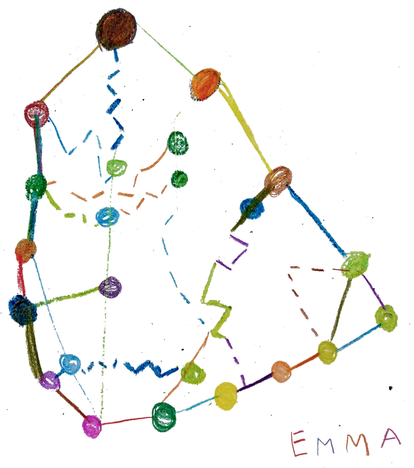
Cette thèse intitulée :

Machine learning-driven hybrid optimization based on decision diagrams

présentée par **Jaime Esteban GONZALEZ JURADO**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

DEDICATION

To Sandra, Emma, and Bianca



ACKNOWLEDGEMENTS

First, I would like to thank my advisors Dr. Louis-Martin Rousseau, Dr. Andrea Lodi and Dr. Andre Cire. This dissertation would not have been possible without your support, collaboration, and guidance. I am honored to have learned from such a group of brilliant researchers. My sincere gratitude goes to them for all the opportunities and lessons.

I am grateful to Professors Dr. Michel Gendreau, Dr. Gilles Pesant and Dr. Michela Milano for agreeing to be on my dissertation committee and for all the feedback and suggestions. I also thank Professor Dr. Andrés Medaglia for being a mentor during my years at Universidad de los Andes. His guidance and support were fundamental in my decision to pursue a Ph.D.

A huge thanks to the CERC team. It was a pleasure to have spent time with such great people and learned from the new generation of talented researchers, professionals, and professors. I also extend my gratitude to the guys and all the technical and administrative staff in CIRRELT, GERAD, Hanalog, and the CERC. I am delighted to have made great friends and memories there. A huge thank you goes to Maria, Simon, Juan, Jorge, Dídac, Camilo, Manuel, Jesus, Carlos, Daniel, Karim, Pedro, and Luciano. And to all the friends outside André-Aisenstadt for their words of encouragement and support during these years.

Quiero agradecer a mis padres y a mi hermano por su amor y apoyo en todo momento. Y, en general, a toda mi familia porque a pesar de la distancia siempre estuvieron cerca.

To my amazing wife Sandra, thanks for your love, patience, and support during this journey. We made it! To my wonderful daughter Emma, for being there from the beginning making these years more special and unique. And finally, to my second little one on her way, Bianca, who has been an extra push of motivation in the final sprint.

RÉSUMÉ

Les problèmes d'optimisation combinatoire se posent dans de nombreux domaines des mathématiques et de l'informatique, ainsi que dans des applications telles que l'ordonnancement et la planification. Malgré des décennies de développement des différentes technologies d'optimisation, certains problèmes combinatoires restent encore difficiles à résoudre. Le développement d'outils d'optimisation génériques pour résoudre ces problèmes difficiles est donc un domaine de recherche actif et continu. Dans cette thèse, nous proposons de nouveaux mécanismes d'optimisation hybrides qui exploitent les avantages complémentaires de différents paradigmes, à savoir, (i) l'optimisation basée sur les diagrammes de décision (ODD), (ii) la programmation en nombres entiers (PNE), et (iii) l'apprentissage automatique (AA) pour améliorer les méthodes d'optimisation.

Dans une première contribution, nous explorons l'utilisation de l'AA pour discriminer la difficulté des instances uniquement en fonction de caractéristiques spécifiques du problème. Nous montrons que l'AA peut effectivement révéler des patrons cachés sur le problème qui rendent sa résolution facile ou difficile par un solveur de PNE. De plus, les fonctions apprises (classificateurs) se révèlent utiles pour fournir des informations au solveur de PNE afin d'ajuster sa configuration et d'augmenter sa performance.

Deuxièmement, nous proposons une approche d'optimisation hybride en combinant l'ODD et la PNE. Nous nous appuyons sur le rôle que les diagrammes de décision (DD) de taille limitée peuvent jouer en tant qu'arbre de recherche. De plus, nous exploitons la machinerie très developpée des solveurs PNE pour concevoir de nouveaux mécanismes permettant d'explorer, de manière collaborative, l'espace de solution. En outre, l'AA est utilisé pour améliorer les performances du solveur hybride en fournissant des informations utiles lors de l'exploration. Les expériences de calcul montrent que si une structure appropriée est révélée, l'approche intégrée est supérieure aux solveurs basées soit uniquement sur les DD, soit sur la PNE.

Enfin, dans une troisième contribution, une nouvelle représentation du problème basée sur les DD est proposée pour le problème quadratique du stable maximum, une version plus difficile et non linéaire du problème du stable maximum. De plus, un algorithme hybride ODD-PNE est étendu en considérant les fonctionnalités de programmation quadratique d'un solveur PNE et une intégration plus étroite de l'AA pour guider l'exploration de l'espace de solution. L'algorithme proposé est plus performant qu'un solveur basé sur la programmation semi-définie et deux solveurs PNE commerciaux majeurs.

ABSTRACT

The discrete and finite nature of combinatorial optimization problems arises in many areas of mathematics and computer science as well as in applications such as scheduling and planning. Despite decades of development and remarkable speedups in general-purpose solvers, some combinatorial problems are still difficult to be solved. The design of generic optimization solvers to tackle such challenging problems is a continuous and active research area.

In this dissertation, we propose novel hybrid optimization mechanisms that exploit complementary strengths from different paradigms. The integrated mechanisms leverage (i) the decision diagram-based optimization (DDO) solving approach, (ii) a more mature technology such as mixed-integer programming (MIP), and, finally, (iii) the use of machine learning (ML) to enhance optimization methods.

In a first contribution, we explore the use of ML for combinatorial optimization. We employ a learning framework to discriminate instance hardness as a function of problem-specific features. We show that ML can effectively reveal hidden patterns that make the problem either easy or difficult to be solved through a MIP solver. Moreover, the trained classifiers prove useful to adjust the MIP solver configuration and boost the performance.

Second, we propose a hybrid optimization approach combining DDO and MIP. We rely on the role that limited-size DDs play as a search tree. We then exploit the mature machinery of MIP solvers and design novel mechanisms to explore, in a collaborative way, the solution space. In addition, ML is employed to enhance the hybrid solver performance by providing useful information during search. Computational experiments show that if suitable structure is revealed, the integrated approach outperforms both stand-alone DD and MIP solvers.

Finally, in a third contribution, a novel problem representation based on DDs is proposed for the quadratic stable set problem, a more difficult and nonlinear version of the maximum independent set problem. Furthermore, a hybrid DD-MIP algorithm is extended by considering the quadratic programming capabilities of a MIP solver and a more tightly integration of ML to guide the exploration of the solution space. The proposed algorithm provides state-of-the-art results when compared with a semidefinite programming-based solver and two leading commercial MIP solvers.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

BDD	Binary Decision Diagram
CP	Constraint Programming
DD	Decision Diagram
DDO	Decision Diagram-based Optimization
DP	Dynamic Programming
DT	Decision Tree
DUM	Dummy Classifier
ILP	Integer Linear Programming
IP	Integer Programming
k-NN	k-Nearest Neighbors
LR	Linear Regression
MDD	Multivalued Decision Diagram
MIP	Mixed-Integer Programming
MILP	Mixed-Integer Linear Programming
MINLP	Mixed-Integer Nonlinear Programming
MIQP	Mixed-Integer Quadratic Programming
MISP	Maximum Independent Set Problem
ML	Machine Learning
MSSP	Maximum Stable Set Problem
QP	Quadratic Programming
QSSP	Quadratic Stable Set Problem
RF	Random Forest
SAT	Boolean Satisfiability
SDP	Semidefinite Programming
SVM	Support Vector Machine

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Optimization has become ubiquitous in science and engineering. Recent advances in algorithms and computer hardware have opened up new possibilities for the design of more efficient algorithms to tackle challenging decision-making problems. In this chapter, we first motivate the importance of combinatorial optimization problems and the effort from different communities to efficiently deal with them. Next, we present introductory concepts on the central topics of this dissertation. Finally, we describe the research objectives to provide an overview of the content of this document.

1.1 Background and motivation

A combinatorial optimization problem consists of finding the best solution among a discrete, finite set of possible alternatives. The different solutions, which typically correspond to assignments or orderings, are characterized by a set of feasibility constraints and evaluated by an objective function that measures their quality. More formally, we consider a combinatorial optimization problem $\mathcal{P}$ of the form $\min\{f(x) : x \in \mathcal{S}\}$ where $x = (x_1, \ldots, x_n)$ is a tuple of variables, $\mathcal{S} \subseteq \mathbb{R}^n$ is the set of feasible solutions, and f is a function that assigns to each solution $x \in \mathbb{R}^n$ an objective function value $f(x)$.

The discrete and finite nature of combinatorial problems arises in many areas of mathematics and computer science as well as in application domains such as scheduling, timetabling, and planning. A special category is that of combinatorial optimization problems over graphs. They are an important case of problems with well-known examples such as the traveling salesman problem and the maximum clique problem. In general, combinatorial optimization problems are NP-hard and the number of feasible solutions grows exponentially with the problem size. Thus, solving such problems through an exhaustive enumeration of the solution set is intractable. Given their relevance, combinatorial problems have received significant attention from the theory and algorithm design communities over the years [1].

Different optimization paradigms coming from *operations research* (OR) and *computer science* have been proposed for solving combinatorial optimization problems. To mention some of such paradigms, mixed-integer programming (MIP), constraint programming (CP) and Boolean satisfiability (SAT) are among the most mature ones with several years of development. However, despite recent significant progress on modern generic solvers (e.g., in MIP [2]), many combinatorial optimization problems remain difficult to be solved.

Among the main reasons why optimization solvers struggle, when tackling some challenging problems, we find (i) scalability issues, (ii) the lack of mechanisms to exploit the problem structure, and (iii) the absence of systematic ways to identify such a structure. Thus, the design of hybrid optimization technology integrating complementary strengths of different paradigms can be an effective strategy to solve and achieve the highest performance [3]. For instance, the integration of artificial intelligence and OR has been one of the most successful cases of integrated optimization methods [4, 5]. Along these lines, in this dissertation, we focus on designing hybrid optimization mechanisms based on both MIP and decision diagrams, the latter are the basis of a much more recent technology in the OR community.

Decision diagrams (DDs) for optimization [6] is a problem-solving paradigm which is built from elements of dynamic programming, mathematical programming, and computer science. We investigate hybrid optimization mechanisms that rely heavily on DD features. Moreover, we explore how *machine learning* (ML) can be useful to both understand hidden patterns when tackling combinatorial problems and enhance optimization methods. Next, we present basic notions on mixed-integer programming, decision diagrams for optimization, as well as general concepts in ML.

1.1.1 Mixed-integer programming problems

Mixed-integer programming is an area of mathematical optimization which comprises several decades of algorithmic development in the OR community [2]. It has become a strong, reliable, and well-known solving paradigm due to its generic-purpose nature and sophisticated algorithms. The MIP technology has then allowed the modeling and efficient resolution of a broad number of complex real-world applications [7, 8].

When using the MIP paradigm, a combinatorial optimization problem ($\mathcal{P}$) is represented as a general mixed-integer program of the form:

$$\begin{aligned} \min_{x} \quad & f(x) && (1.1)\\ \text{subject to,} \quad & Ax \geq b, && (1.2)\\ & x_i \in \mathbb{Z}, \qquad \forall i \in I, && (1.3) \end{aligned}$$

where variables $x \in \mathbb{R}^n$ and $I \subseteq N = \{1, \ldots, n\}$ is the index set of the variables required to be integer. The problem constraints (1.2) are modeled by a system of linear inequalities where $A \in \mathbb{R}^{m \times n}$ is the matrix of constraints coefficients and $b \in \mathbb{R}^m$ the right-hand sides. A solution x is called feasible if it satisfies constraints (1.2) and (1.3). Finally, a function $f : \mathbb{R}^n \to \mathbb{R}$ is minimized to find the optimal solution x^* with optimal objective value $f(x^*) = z^*$.

MIP formulations can be classified according to the properties of the objective function (1.1). The most studied case is that of a linear objective function which, in the OR community, is indifferently referred to as a Mixed-Integer Linear Programming (MILP) formulation and takes the form

$$\min \{c^T x : Ax \geq b, x_i \in \mathbb{Z} \ \ \forall i \in I\}, \tag{1.4}$$

where $c \in \mathbb{R}^n$ is a vector of objective coefficients. We also remark that when all variables are required to be integer, i.e., $|I| = n$, the resulting problem can be referred to as a (pure) integer program (IP) or integer linear programming (ILP) model for the case of a linear objective. In this dissertation, although we mainly experiment with models which are pure IPs, we use as well the general term MIP.

Since MIP is NP-hard, a fundamental concept to solve MIP problems is its linear programming (LP) relaxation that has the form

$$\min \{c^T x : Ax \geq b\}, \tag{1.5}$$

i.e., it is the resulting polynomially solvable problem where the integrality requeriments are dropped. Let x^*_{LP} be the optimal solution of (1.5) and $f(x^*_{LP}) = z^*_{LP}$ its optimal solution value. Then, z^*_{LP} provides a lower bound on the optimal solution value of (1.4), i.e., $z^*_{LP} \leq z^*$.

For our purposes, another case of special interest is Mixed-Integer Quadratic Programming (MIQP),

$$\min \left\{ \frac{1}{2} x^T Q x + c^T x : Ax \geq b, x_i \in \mathbb{Z} \ \ \forall i \in I \right\}, \tag{1.6}$$

in which we minimize a quadratic objective function, where $Q \in \mathbb{R}^{n \times n}$ is a symmetric matrix. It is well known that problem (1.6) is NP-hard too, as it contains problem (1.4) as a special case.

Currently, state-of-the-art MIP solvers tackle an optimization problem by using a machinery composed of different significant building blocks. The main component is a branch-and-bound search tree, i.e., a divide-and-conquer approach where all possible solutions of an optimization problem are implicitly enumerated to search for the optimal one. In the case of MILP, the standard solving method is the so-called *branch-and-cut* scheme. The latter mainly consists of a LP-based branch-and-bound tree enhanced with cutting planes, presolving, and primal heuristic procedures.

The implicit exploration is achieved by storing partial solutions and partioning the solution space (i.e., *branching*) into smaller subproblems in a tree-structure fashion. In parallel, the scheme computes and updates global lower and upper bounds on the optimal solution value. Therefore, at each node of the tree, the LP relaxation provides information on the node's optimal solution value so it is possible to prune provable suboptimal regions of the search space (i.e., *bounding*). In case that a node subproblem cannot be discarded, its LP optimal solution also provides information about which variables are still fractional and can be selected to branch on and partition further the solution space.

On the other hand, in the case of solving a MIQP problem, a MIP solver has different options depending on the properties of (1.6). In general, modern MIP solver engines have also incorporated mechanisms to directly solve a MIQP model by means of a nonlinear programming-based branch-and-bound search tree. In such a scheme, a node subproblem corresponds to a quadratic programming (QP) relaxation, i.e., a problem similar to (1.6) but removing the integrality constraints $x_i \in \mathbb{Z} \; \forall i \in I$, following branching and bounding procedures. Alternatively, state-of-the-art MIP solvers can also linearize the MIQP model and exploit the more mature MILP machinery with the trade-off of a resulting larger problem.

In any case, several decisions become critical within a branch-and-bound scheme in MIP solvers. We mention among others the *node selection* which refers to the choice of the subproblem to look at next, and the *variable selection*, i.e., the strategy to determine the next variable to branch on and create new subproblems. We remark that many of such decisions are usually guided by heuristic mechanisms. Thus, an area with growing attention is the design of more systematic and automatic ways of improving the search scheme by making better decisions related to it.

1.1.2 Decision diagram-based optimization

Decision diagrams are data structures initially used to manipulate Boolean functions and introduced for applications in circuit design, verification, and model checking [9, 10, 11]. They have been widely studied in computer science; and a survey can be found in [12] and [13, Chapter 3]. For the discrete optimization community a DD can also be interpreted as a compact graphical representation of the solution set of a combinatorial problem. Thus, they have been introduced for discrete optimization and constraint programming [14, 6].

Although DDs were proposed in the end of 1950s, DD-based optimization (DDO) is relatively new in the OR field. The DDO modeling framework is strongly connected to the recursive formulations in dynamic programming (DP) [15], however, DDO also incorporates mathematical optimization features such as bounding procedures to mitigate the typical DP's curse of dimensionality. A DD can be interpreted as a compact graphical representation of the solution set $\mathcal{S}$ of a discrete optimization problem $\mathcal{P}$ where, for our purposes, the variables $x_1, \ldots, x_n$ have finite domains $D_1, \ldots, D_n$, respectively.

Specifically, an ordered decision diagram representation of $\mathcal{P}$ is a layered directed acyclic graph whose nodes are partitioned into $(n+1)$ layers. The first and terminal layers are made up by only one node, the root node r and the terminal node t, respectively. In its most general version, arcs only connect nodes of consecutive layers and there is a one-to-one correspondence between the variables and the layers' outgoing arcs. Such arcs then represent value assignments of the corresponding decision variable. In addition, we can associate weights with the arcs of a DD to model the objective function $f(x)$. In that case, each path from r to t represents a feasible solution of S; and consequently, the shortest path from r to t corresponds to the optimal solution of P.

Moreover, if every domain D_i, $i = 1, \ldots, n$, contains no more than two values, it implies that every node in the DD has at most two outgoing arcs and such DD is then referred to as a *binary* decision diagram (BDD). On the other hand, if there exists at least one variable domain D_i containing more than two values (i.e., at least one node in the DD with more than two outgoing arcs), the associated DD is referred to as a *multivalued* decision diagram (MDD).

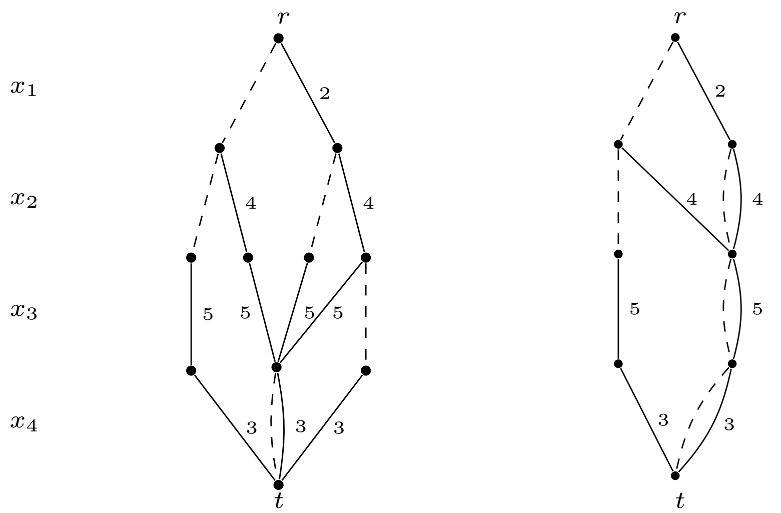
For an illustrative example, we describe a DD representation for a combinatorial optimization problem represented by the following MIP formulation:

$$
\begin{aligned}
\min_{x} \quad & 2x_1 + 4x_2 + 5x_3 + 3x_4 && (1.7)\\
\text{subject to,} \quad & x_1 + x_3 \geq 1, && (1.8)\\
& x_2 + x_3 \geq 1, && (1.9)\\
& x_3 + x_4 \geq 1, && (1.10)\\
& x_1 + x_2 + x_4 \geq 1, && (1.11)\\
& x_i \in \{0,1\}, \quad i = 1, ..., 4. && (1.12)
\end{aligned}
$$

Figure 1.1(a) shows a weighted binary decision diagram B representation for the model defined by (1.7)-(1.12). The nodes of B are partitioned into 5 layers, where layers 1 and 5 contain the root node r and the terminal node t, respectively. The dashed arcs represent that the associated variable is assigned to 0, and solid arcs model assignments to 1. For instance, the dashed and solid arcs going out from the root node r represent $x_1 = 0$ and

$x_1 = 1$, respectively. Thus, every path p from r to t represents an assignment to the variables x_1, x_2, x_3, x_4, and we denote this assignment x^p. In addition, costs are assigned to arcs of B to represent the objective function. For example, a weight 2 is assigned to the solid arc going out from r because any path crossing such an arc implies that the coefficient of variable x_1 in the objective function (i.e., 2) is collected. Conversely, every dashed arc has weight equals to 0. For our purposes, a relevant feature of a DD is its *width*. The width of a DD layer is the number of nodes in the layer, then the width $\omega(B)$ of a decision diagram B is the maximum width among all its layers. Considering again the BDD B in the Figure 1.1(a), $\omega(B) = 4$.

Note that the 8 feasible solutions (x_1, x_2, x_3, x_4) of problem (1.7)-(1.12), namely, $(0, 0, 1, 1)$, $(0, 1, 1, 0)$, $(0, 1, 1, 1)$, $(1, 0, 1, 0)$, $(1, 0, 1, 1)$, $(1, 1, 0, 1)$, $(1, 1, 1, 0)$, and $(1, 1, 1, 1)$; are exactly represented by the 8 paths from r to t in the BDD of Figure 1.1(a). In addition, as the length of a $r - t$ path p in B is equal to the objective value $f(x^p)$, this implies that there is a complete correspondence between $r - t$ paths in B and the feasible solutions in the problem. We then say that B is an exact DD for the model (1.7)-(1.12). In this manner, an exact DD representation reduces a discrete optimization problem to a shortest-path computation. For the illustrative example, note that the path $x = (x_1, x_2, x_3, x_4) = (1, 0, 1, 0)$ corresponds to the shortest path with length 7, which is the optimal solution value of the problem.



(a) Exact DD

(b) Relaxed DD

Figure 1.1 DD representations for the illustrative example (1.7)-(1.12)

Although an exact DD reduces the optimization problem to a shortest-path computation, such DDs can grow exponentially with the problem size. To overcome this challenge, Andersen et al. [14] introduced the concept of *relaxed* DDs which are graphs of controllable size that represent an over-approximation of the feasible solution space. In this case, the size of the DD is kept under control by defining a maximum width (W) such that $\omega(B) \leq W$, which is observed when compiling the DD representation. In a relaxed DD, there is a path associated with each feasible solution, however, there are other paths that are not necessarily feasible solutions of the problem. Hence these limited-size decision diagrams become a discrete relaxation of the problem because the combinatorial (graph) representation is relaxed and can be used to obtain bounds on the optimal solution value. Figure 1.1(b) presents a relaxed BDD of maximum width 2 for the problem (1.7)-(1.12). Note that all the $r-t$ paths in Figure 1.1(a) are included in the relaxed BDD of Figure 1.1(b), however, there are also present paths that correspond to infeasible solutions. Nevertheless, while the shortest path defined by the path $(1,0,0,0)$ is infeasible because of constraints (1.8) and (1.10), it provides a lower bound of 2 on the optimal solution value.

Another type of limited-size decision diagrams are the so-called *restricted* DDs which were introduced in [16]. A restricted DD is an under-approximation of the solution set of a problem and is obtained by heuristically removing nodes from a layer once the maximum width W is reached. In such a procedure, we remove feasible solutions and possibly the optimal one too. However, a shortest path computation on a restricted DD provides a feasible solution and then an upper bound on the optimal solution value of a minimization problem.

In addition of being a useful bounding technique, DDs can also be perceived as the search tree within a branch-and-bound procedure. Such a branching scheme brings up a different way of exploring the solution space of a problem. The DD-based search consists of recursively branching on decision diagram nodes (i.e., set of partial solutions) rather than branching either on a fractional variable which must be integer like in traditional linear-programming-based branch and bound, or on a variable-value pair like in a CP search tree. Since DD-based technology provides the main components of an optimization solver, i.e., a modeling framework, bounding procedures, and a branching scheme; [17] introduce a stand-alone DD method for solving discrete optimization problems based exclusively on limited-size decision diagrams. Such a paradigm has shown to be highly competitive with generic optimization solvers and is extensively exploited in this dissertation.

1.1.3 Machine learning concepts

Machine learning (ML) is the discipline focused on using statistical techniques to design computer systems that automatically improve through experience [18]. The recent explosion and success of ML is mainly attributed to both the development of new learning algorithms and the availability of data and computational power [19]. Consequently, ML algorithms and methodologies have appeared in several contexts and applications, and the field of combinatorial optimization is no exception.

Broadly speaking, ML algorithms learn from (training) data to infer a function that predicts an output (label). They can be divided into three main categories, namely, *supervised*, *unsupervised*, and *reinforcement* learning [20]. Such categories are defined based on the objective of the learning task and the characteristics of the provided data from which the algorithm aims to learn. In supervised learning, an algorithm learns from labeled data. In unsupervised learning, the training data has no target label and the goal is more related to find similarities (i.e., clustering) or distribution of data within the input space. Finally, in reinforcement learning, the algorithm learns to take actions (i.e., make a sequence of decisions) in a given game-like situation in order to maximize a reward. In this case, the algorithm is not provided with examples of the right output, instead it must discover them by a trial-and-error process.

We are particularly interested in supervised learning, where a ML model is trained with pairs of input-output examples to learn a function that maps an unseen input to a predicted output. The input is usually referred to as features, while the output is a special feature denoted by label. More formally, consider a set of training data $\mathcal{D} = \{(\mathbf{x}^i, y^i)\}_{i=1}^m$ with m examples, where the i^{th} example is given by a n-dimensional vector of features values $\mathbf{x}^i$ and a label y^i. Let X and Y be the input and output space, respectively, such that $\mathbf{x}^i \in X$ and $y^i \in Y$ for $i = 1, \dots, m$. The goal of a supervised ML algorithm is to learn a function $h : X \to Y$ that minimizes a loss function which measures how good h is when used as a predictor. The function h is typically called a *classifier* if the output is discrete and a *regression function* if the output is continuous.

We focus on supervised classification tasks with two class labels which are commonly referred to as binary classification problems. Supervised learning algorithms for classification can be categorized based on the similarity of their mapping functions and how they work. We mention, among others, tree-based methods, kernel-based methods, neural-network-based methods, instance-based learning methods, and ensemble methods. Nevertheless, such categories are not exclusive and a ML algorithm can belong to different ones. Moreover, when tackling a classification problem, a relevant question is related to the selection of the ML model to learn a classifier. Each ML algorithm has its advantages and limitations depending on the dataset and application; therefore, in this dissertation, we will follow a typical approach in supervised learning. Namely, we assess different algorithms tuning their specific parameters to obtain the best model according to pertinent performance metrics. We will employ some of the most well-known ML algorithms, specifically, logistic regression, support vector machine, decision tree, k-nearest neighbors, and random forest. We refer the reader to

[20] and [21] for a comprehensive description of the mentioned ML algorithms. Nevertheless, we provide a short description of such ML algorithms for completeness.

- *Logistic Regression* (LR) [22]. This method is an extension of the linear regression model for the case of classification. In LR, instead of using a linear equation to explain the relationship between a dependent variable (label) with the independent variables (features), a logistic sigmoid function is employed to model probabilities and make the output binary. One important advantage of LR is that it provides explicit probabilities of classification rather than just the label information. However, as its main limitation, LR performs poorly in non-linear problems and requires a good selection of relevant features.

- *k-Nearest Neighbors* (k-NNs) [23]. This is an instance-based learning algorithm that uses the nearest neighbors of a data point for prediction and can be used for regression and classification. In the classication case, k-NNs algorithm assigns as prediction the most common label of the nearest neighbors of an example. k-NNs algorithm is also a non-parametric method, this means that there are no parameters to be learned and it is difficult to get some interpretability from the classification (i.e., degree of human understanding of the cause of a model's result).

- *Decision Tree* (DT) [24]. In this tree-based method, the data is split multiple times based on threshold values in the features. From this procedure, different subsets are generated in such a way that each data example belongs to one subset. The final subsets are denoted leaf nodes, each associated with a class label. Paths from the root node of the tree to a leaf node denote the classification rules which are used to make a prediction. The main advantage of this method is its simplicity and interpretability due to the classification rules and easy computation of importance scores for the features, i.e., their relevance for the prediction. However, the main disadvantage is that, usually, DT has high risk of overfitting, i.e., it does not generalize well on unseen examples.

- *Support Vector Machine* (SVM) [25]. In its simplest version, a SVM is a linear model for classification. Given a set of training labeled data points, a SVM-algorithm first generates a feature space which represents a finite dimensional vector space where each dimension corresponds to a particular feature. The objective of a SVM is to find a hyperplane (i.e., the decision boundary) that separates the data points into regions (i.e., the classes) in which they belong. A major advantage of SVM is that it also works in non-linearly separable data. The latter is achieved by using the so-called kernel trick, i.e., by mapping the original feature space into a higher-dimensional space. The SVM is widely used due to its good performance on nonlinear problems, however, it is often difficult to interpret because it does not provide a direct probabilistic interpretation for the classification.

- *Random Forest* (RF) [26]. This is an ensemble learning method, i.e., it combines one or several base models to produce a better predictive performance. In the case of RF, the base model corresponds to the described above decision tree. A RF algorithm generates several decision trees from randomly selected subsets of the training data. The algorithm then considers the predicted class labels of the set of DTs and aggregates them to make a final prediction based on the mode of the labels, i.e., by taking the majority vote. As one of its main advantages, a RF tends to correct the overfitting problem in DT, keeping the interpretability. However, a RF is difficult to train, requiring more computational resources than other algorithms previously described.

Due to its nature, combinatorial optimization has become an important source of applications where ML can be leveraged. The mechanisms of SAT, CP, and MIP solvers can be enhanced through the integration of ML-based decisions. For instance, algorithmic questions, that arise while solving an optimization problem with MIP solvers, have been recently tackled using machine learning approaches. Finally, as ML for combinatorial optimization is a component further explored in this dissertation, we will extend the discussion on recent literature in Chapter 2.

1.2 Research objectives

The primary objective of this dissertation is the design of novel hybrid optimization mechanisms that mainly integrate DD-based optimization with more mature paradigms from classical operations research such as MIP. We aim at identifying complementary strengths from paradigms to devise hybrid strategies that exploit, in a collaborative way, both different search schemes and problem structure. Moreover, we investigate how machine learning can discriminate suitable problem structure for a stand-alone optimization solver. Our main goal is to design optimization technology that efficiently solve difficult discrete optimization problems by exploiting the integration of DD-based optimization, MIP, and ML.

We rely on limited-size DDs by further emphasizing their role as approximations of the search tree of a problem. In addition, we explore how a mature technology such as MIP could be further exploited within a DD-based search scheme. Next, we explore how machine learning techniques can become an important component to understand problem hardness and be incorporated alongside hybrid optimization mechanisms.

Finally, we evaluate the proposed optimization mechanisms on combinatorial optimization problems and, specially, problems over graphs. In particular, we select two challenging case studies, the maximum independent set problem and the quadratic stable set problem. Such well-known problems have been studied from different optimization paradigms and they can be used to properly compare different solving methods. Finally, we develop extensive computational experiments to validate the competitiveness of the mechanisms proposed in this dissertation.

1.3 Thesis outline

The remainder of this document is organized as follows. In Chapter 2, we discuss related works on integrated optimization methods, DD-based methodologies, and the role of machine learning for understanding solvers' performance and designing discrete optimization algorithms. Chapter 3 describes the structure of this document and the contributions of the main chapters of this dissertation. Chapter 4 investigates the use of supervised learning to discriminate problem-specific instance hardness when using a MIP solver. Chapter 5 describes a novel hybrid algorithm based on the integration of decision diagrams and mixed-integer programming which is enhanced with machine learning-based decisions. Chapter 6 tackles a relevant problem in quadratic programming by providing a novel representation and adapting a hybrid approach that combines DDs, quadratic programming capabilities of MIP, and ML. In Chapter 7, we provide a general discussion on the contributions of this thesis from a unified perspective. Finally, in Chapter 8, we conclude and layout the future work in the area resulted from this dissertation.

CHAPTER 2 RELATED WORK

Although the main chapters in this dissertation are self-contained, we provide here a brief discussion on related literature to the topics covered in this work. In order, we focus on (i) integrated optimization methods, (ii) approaches that incorporate DD technology for solving optimization problems, and (iii) the use of ML for discrete optimization.

2.1 Integrated optimization methods

The main building blocks of an optimization solver are a modeling framework, a relaxation technique, inference procedures, primal heuristics, and a search scheme. The significant progress made by individual disciplines has allowed to understand and identify in which components their main advantages lie. To give some examples, mathematical programming has made significant contributions in relaxation methods and concepts such as duality theory. Constraint programming, on the other hand, has provided strong inference techniques and a more flexible modeling framework, while dynamic programming has been good at modeling recursive structure. Thus, it is natural to think about optimization methods that can handle several of these advantages in a unified approach.

Integrated (hybrid) optimization methods is not a new area in the OR community [3]. In recent years, much emphasis has been given to the integration of techniques coming from different areas such as CP, SAT, and MIP (see, e.g., [27]). Two main forms of integration can be identified. First, in a simpler case, the integration is employed to enhance only small components of an independent solving technology, e.g., the use of CP-based techniques to strengthen the bounds on variables during presolving in a MIP solver. In a second place, a tighter integration can lead to the development of strong novel algorithms whose hybrid mechanisms are based on the independent paradigms.

Several optimization problems can be efficiently tackled by leveraging multiple solution approaches with complementary strengths. For instance, the integration of MIP and CP is one the most successful cases [4, 5]. Related to tightly integrated techniques, these methods commonly involve a problem decomposition into two or more subproblems, which defines in advance the component that is more suitable for existing techniques.

We can mention as a significant example of such a-priori decomposition, the case of logic-based Benders decomposition [28] which has been successful in different contexts such as machine scheduling [29]. The main idea of this algorithm is to decompose the problem into a master problem and a set of subproblems providing a natural framework for the integration of different disciplines. The different problems are then tackled by using different paradigms with an important inference component to communicate information between them.

In the same sense, other classical decomposition techniques in MIP, such as column generation, can profit from the complementary strengths from other technologies. In a pure column generation procedure, a problem is decomposed in a restricted master problem with as few variables as possible whereas an auxiliary problem generates interesting variables for the master problem at each iteration. In the case of hybridization, successful applications have appeared in the literature in the context of CP-based branch and price. In this case, a column generation procedure is embedded at each node of a branch-and-bound tree where the subproblems are more efficiently solved by using CP [30, 31].

Given the nature of its mechanisms, decision diagram-based optimization is closely involved with the concepts behind integrated methods. Since their introduction in the OR and CP communities [32, 33, 14] and after proving an effective stand-alone optimization method [34, 16, 17, 6], decision diagrams have grown in interest, becoming a fruitful research area. In general, for this discussion on related literature, most of the works integrate in some way the DD paradigm within MIP or CP approaches to boost the performance of optimization algorithms.

2.2 Decision diagrams integration

The constraint programming community has been fundamental for the development of DD-based techniques for combinatorial problems. One of the main reasons is the seminal work in [14] introducing the concept of relaxed DDs for CP. The authors use limited-size DDs as a more flexible alternative to replace the traditional constraint store (usually based solely on the variable domains) and show a very effective DD-based filtering procedure for `AllDifferent` constraints. Since then several works have been developed using DD-based constraint propagation and filtering [35, 36, 37].

Also in the context of CP, and particularly related to solving sequencing and scheduling optimization problems, multivalued DDs have been embedded as global constraints in CP models [38, 39] to exploit such a graphical representation of the problem. More recently, for a variant of the traveling salesman problem, Castro et al. [40] propose a MDD-based method that leverages the DD representation within a Lagrangian method [41]. The approach profits from information of both the DD and the linear programming representation.

Cutting-plane is a fundamental component of state-of-the-art MIP solvers and DD-based cut generation is a research trend that has been reinforced in recent years, starting with the work of Becker et al. [32]. Several methods have been proposed to use limited-size DDs to exploit the problem structure and generate cuts. In this setting, [33] made significant contributions (i) connecting BDD for generating cuts for 0-1 integer programming models as well as (ii) proposing a network-flow reformulation of a BDD which has been exploited in several BDD-based cutting-plane procedures. Tjandraatmadja and van Hoeve [42] extract target cuts from relaxed DDs by providing a connection between DDs and polar sets. An extension of the previous work and the one in [33] is proposed in [43]. The authors use DDs to derive valid inequalities for non-linear constraints by deriving a DD representation for only specific substructures rather than the entire solution set.

DD-based optimization has been also employed as a stand-alone solving technique for particular components of classical decomposition methods in MIP. In particular, Morrison et al. [44] use DDs to solve the pricing problem in a branch and price scheme when tackling the graph coloring problem. In such a decomposition, each branching node of a traditional linear programming-based branch-and-bound tree is solved through a column generation procedure. Within the column generation, the pricing problem corresponds to the maximum independent set problem which is solved through DD-based technology. In this procedure, the DD-based solver is repeatedly used to tackle the same general structure in a regular basis, highlighting the importance of improving the solver's performance.

DD-based decomposition [45] is another interesting way in which DDs have been used to represent and solve optimization problems. The main idea is to represent components of a problem through different decision diagrams and then employ a lifting procedure based on a network-flow reformulation such that the paths in the DDs are linked to the original problem variables through constraints. For nonlinear optimization, DDs have been used to strengthen MIP models through their equivalent linear reformulation as shortest paths [46] and also for modelling problems with quadratic constraints [47].

In a final category, DD-based optimization has been used for stochastic optimization and particularly in the context of two-stage stochastic programming, an effective approach when tackling decision-making problems under uncertainty. Lozano and Smith [48] derive Benders-like cuts by modelling the second-stage decisions through DDs which consider first-stage information. Moreover, Serra et al. [49] model both the first and second stage decisions using DDs. In this case, the DDs representations are linked through channeling constraints that are optimized by using an IP formulation.

We observe an extensive body of research exploiting DD-based features within different optimization paradigms and traditional methods. As an alternative research direction, in this dissertation, we explore hybrid optimization mechanisms where the DD-based branch and bound [17] is a core component in the integrated method. Moreover, we aim at designing collaborative and complementary mechanisms that profit from a more mature paradigm such as MIP. Finally, since we explore ML to design automatic mechanisms to identify complementarity between different solving paradigms, we proceed to discuss literature related with the role of ML and OR.

2.3 Machine learning for optimization

Another significant component of this thesis is the idea of using ML techniques for enhancing combinatorial optimization approaches. Recently, different works have proven the success and yet promising character of the *ML-for-OR* research paradigm. Bengio et al. [50] present a comprehensive survey that categorizes how the machine learning framework can be exploited for optimization algorithms and places different contributions in the literature within different types of integration.

We focus on two different trends when using ML, namely, (i) to improve optimization algorithms prior to its execution and (ii) to tightly integrate it alongside a solving algorithm [50]. In the improvement of optimization algorithms, the objective is to use a learning framework to provide additional information to the combinatorial optimization algorithm and, accordingly, adjust it before the resolution process starts. On the other hand, in a tighter integration, a ML model is systematically and repeatedly employed through the resolution process of the combinatorial optimization algorithm. Such an integration implies that the optimization algorithm's performance relies heavily on the assistance provided by the ML-based decisions. Along these lines, ML-based approaches have been used to make a wide range of decisions in configuration, selection, and design of optimization solvers.

In algorithm configuration, MIP solvers are relevant because, despite their exact nature, many heuristic decisions are made during the branch-and-cut scheme. Several ML-based approaches to strengthen MIP solvers have been investigated in the literature, such as in variable selection [51, 52], and running heuristics [53] in a branch-and-bound tree. Remarkably, as another example within the DDO paradigm, Cappart et al. [54] propose a ML framework to improve the dual bounds obtained from relaxed decision diagrams which are a main component in DD-based solvers.

Another relevant area where ML has been used for discrete optimization is that of algorithm design. In [1], both reinforcement learning and graph embedding are employed to learn heuristics for solving optimization problems over graphs such as the minimum vertex cover, maximum cut, and traveling salesman problem (TSP). In the same category, Kruber et al. [55] define a ML framework to determine whether or not a MIP model should be decomposed. In case several reformulations are possible, the ML approach also decides which one should be selected.

We also highlight ML applications for algorithm selection [56]. Xu et al. [57] introduce the portfolio-based algorithm selection approach for SAT solvers. Given a set of SAT solvers, the authors use ML to predict an approximate solver runtime for a given problem based on features of the instance and the algorithms' previous performances. On another case, Bonami et al. [58] propose a ML application to make a relevant decision when tackling MIQP models with CPLEX, a state-of-the-art MIP solver. The authors learn a classifier to predict whether to linearize a quadratic programming (QP) model. Such a linearize/not-linearize decision is analogous to algorithm selection because it ultimately determines the algorithm to be employed by the solver for the resolution process.

Finally, ML has been also useful to predict algorithm performance. Hutter et al. [59] use machine learning techniques to build regression models that predict algorithm runtime for SAT and MIP solvers. The prediction is based on features coming from the algorithm partial progression and the generic problem representation. The authors present an extensive empirical analysis on different types of instances with several ML techniques. In a more specific approach for MIP, Fischetti et al. [60] define a binary classification framework and use ML methods to predict whether or not a generic mixed-integer linear programming instance will be solved to optimality within a given time limit. In this case, MIP-generic features are defined and collected from the partial progression of the branch-and-bound tree search.

Throughout the remaining chapters of this thesis, we will discuss how different contributions converge to a unified framework. Finally, we will also discuss how such contributions fit into the literature and fill different gaps.

CHAPTER 3 THESIS ORGANIZATION

In the following chapters, we develop hybrid optimization mechanisms that tightly integrate decision diagrams and mixed-integer programming for solving challenging combinatorial optimization problems. Moreover, the use of ML to either improve or automatize elements of an optimization solver is also a significant component of this dissertation.

Chapter 4 focuses on the ML-for-OR approach. We use supervised machine learning to discriminate problem-specific instance hardness when using a MIP solver. In the classification approach, data examples correspond to problem instances while the MIP solver performance when tackling such instances is used to define a target label for each data point as either easy or hard. We then aim to learn a function (classifier) that for unseen instances discriminates their hardness for a MIP solver depending solely on problem-specific instance features related to the problem at hand.

As case studies, we select challenging combinatorial optimization problems over graphs, namely, the maximum independent set problem (MISP) and the quadratic stable set problem (QSSP). We get insights from the trained classifiers on the graph properties that impact the instance hardness for the MIP solver the most. In particular for the QSSP, we answer an important algorithm-selection question when tackling mixed-integer quadratic programming models. In this case, the classification label is defined by the linearization or not of the QSSP model when tackled with a state-of-the-art MIP solver. We then build a predictive model that determines whether to linearize or not the QP model based only on QSSP-specific instance features. In addition, when comparing against the out-of-the-box classifier that decides on the linearization of a MIP solver, we show that our predictive model is useful for algorithm configuration, improving the solver's performance on QSSP instances.

In Chapter 5, we introduce a novel hybrid optimization algorithm that mainly integrates decision diagrams and integer linear programming to explore, in a collaborative way, the solution space. The hybrid solver strongly relies on a DD-based branching scheme where an approximate DD plays the role of the search tree. Instead of branching on fractional variables like in traditional linear programming-based branch and bound, we branch on sets of partial solutions (nodes of relaxed DDs). The mechanisms in the hybrid solver allow to obtain primal and dual solutions from the two different representations triggering novel pruning strategies when recursively generating limited-size DDs. The hybrid framework raises an algorithm-selection question which is cast as a machine learning problem. ML-based procedures are employed to guide the exploration of the solution space as the hybrid algorithm progresses.

First, trained classifiers are invoked in the root DD node and then the relevant features, revealed during the training phase with a tree-based ML-model, are used to define a simple decision tree to guide the exploration in the remaining DD nodes. We use as case study the MISP. The numerical experiments show that, in presence of suitable problem structure, the integrated DD-ILP approach exploits complementary strengths and improves upon the performance of both a stand-alone DD solver and a MIP solver.

Chapter 6 focuses on the nonlinear and more challenging QSSP. We present a novel problem representation for the QSSP through BDDs which is based on a proposed dynamic programming formulation for the problem. Furthermore, we adapt the hybrid approach that combines DDs and MIP by highlighting the modeling flexibility offered by decision diagrams to handle a nonlinear problem. Machine learning also becomes a significant component within the method for the QSSP. We train offline a support vector machine model to include the trained classifier alongside the hybrid method and guide, on-the-fly, the search mechanisms at each BDD subproblem. In the computational experiments, the hybrid approach shows to be superior, by at least one order of magnitude, to two leading commercial MIP solvers with quadratic programming capabilities and also a semidefinite programming (SDP)-based branch-and-bound solver.

Finally, Chapter 7 presents a synthesis of the different contributions from a unified perspective. In Chapter 8, we present the main conclusions along with a discussion of the limitations and future research.

CHAPTER 4 LEARNING TO DISCRIMINATE PROBLEM-SPECIFIC INSTANCE HARDNESS WHEN USING A MIP SOLVER

Authors: Jaime E. González, Andre A. Cire, Andrea Lodi, Louis-Martin Rousseau
To be submitted.

Abstract: In this chapter, we present a machine learning (ML) application to predict problem-specific instance hardness when using a Mixed-Integer Programming (MIP) solver. We cast this prediction as a binary classification task based on the performance of a MIP solver when tackling the problem at hand. In particular, traditional supervised ML methods are used to learn a classifier that predicts instance easiness/hardness for a MIP solver as a function of problem-specific instance features. We focus on the Maximum Stable Set Problem and the Quadratic Stable Set Problem as case studies. Learning experiments show that the learned classifiers capture a statistical pattern on the problem-specific structure, which in turn could be relevant for algorithm design and selection.

Keywords: Supervised machine learning, Mixed-integer programming, Instance hardness, algorithm performance prediction

4.1 Introduction

The integration of machine learning (ML) and operations research has established itself as a productive and growing research area (see, e.g., [50] for a recent survey). ML-based methodologies have been used to make a wide range of decisions in configuration, selection, and design of optimization solvers. Two different approaches have been discussed on the ML-for-OR research trend, namely, (i) when ML improves optimization algorithms prior to its execution, and (ii) when it is tightly integrated alongside a solving algorithm.

In the improvement of optimization algorithms, the objective is to use a learning framework to provide additional information to the combinatorial optimization algorithm. Then, the configuration of the algorithm is adjusted accordingly before the resolution process starts. For instance, in [55, 58, 59], ML-models guide decisions related to the choice of the decomposition or configuration that the optimization algorithm should use to tackle the problem at hand.

On the other hand, in a closer integration, a ML model is repeatedly employed within the combinatorial optimization algorithm. In this case, the optimization algorithm's performance relies heavily on the assistance provided by the ML-based decisions. A relevant case is the branch-and-cut scheme of MIP solvers, where several works have proposed the use of ML models to make decisions related to the branching and search strategies [51, 52].

For our purposes, we are particularly interested in successful ML applications for per-instance algorithm selection problems. In such a context, given a set of algorithms that can tackle a computational problem and a specific instance that has to be solved, the problem consists of determining which algorithm is expected to perform best on that particular instance. In the context of SAT, Xu et al. [57] define a portfolio solver by using ML algorithms to predict the solvers' performances which in turn are based on problem features of the instance. Similarly, Bonami et al. [58] develop a ML application to decide on the linearization of a MIQP model when tackled by a state-of-the-art MIP solver. The linearize/not-linearize decision is analogous to algorithm selection because it determines the algorithm used by CPLEX to tackle an instance problem.

Partially related to algorithm selection, ML has been also employed to predict algorithm performance. In Hutter et al. [59], ML is employed to build regression models and predict algorithm runtime for SAT and MIP solvers. The learned function is based on features coming from both the problem representation and the algorithms partial progression. In the MIP context, Fischetti et al. [60] design a learning framework to classify whether or not a MILP instance will be solved to optimality by a general-purpose MIP solver within a given time limit. The prediction is based on MIP-generic features that are either static or dynamic, the latter, collected from the partial progression of the branch-and-cut scheme.

Understanding the problem-specific instance structure of a combinatorial problem and how such a structure affects the solution performance is relevant for algorithm design. This chapter points in that direction and is partially related with ML approaches for algorithm performance and selection. As prediction models for algorithm performance could be useful for gaining insights on the hardness of an instance [59], our work explores the use of ML to predict instance hardness through an assessment of the MIP solver performance. As case studies, we tackle two classical combinatorial optimization problems over graphs such as the maximum stable set problem (MSSP) and the quadratic stable set problem (QSSP). In addition, related with [58] and specifically for the QSSP, we use the learning framework to understand the problem-specific instance features that better discriminate the decision on the linearization of the QP formulation when tackled by a MIP solver.

Instead of using MIP-generic features (such as in [59, 60]) as the input of the predictive models, we focus on problem-specific characteristics for each case study. We contribute with a supervised learning framework which aims to understand instance hardness for a MIP solver by performing a binary classification task. Learning experiments show that problem-specific features are worth to be considered when predicting instance hardness in MIP technology, and also that automated predictive models can capture problem structure to make a meaningful discrimination. Finally, getting insights on the characteristics that affect problem solvability using a MIP solver can contribute in several ways. For instance, (i) in the design of new optimization methods which could consider MIP technology as a component, (ii) in the development of fair benchmark datasets [61], and (iii) to support the configuration of a MIP solver for tackling a problem based on its instance structure.

The remainder of this chapter is organized as follows. In Section 4.2, we describe both the supervised learning application and the case studies. Section 4.3 presents details on the methodology and experimental results when learning a classifier to discriminate instance hardness. Next, Section 4.4 describes a second learning task associated with gaining insights on instance hardness based on the possible linearization of the QSSP. Finally, we conclude in Section 4.5.

4.2 Learning methodology for predicting instance hardness

Supervised ML is the process of learning a function from a given set of examples (i.e., the training dataset) and assigning a discrete class label to an unseen instance. Each data example is described by a set of features (attributes) and an associated class. The learned function (classifier) then maps the features of new instances to the available classes revealing a possible hidden structure within the training dataset.

We aim at investigating problem-specific instance hardness for MIP solvers by formulating such algorithmic question as classification tasks. We will then address those tasks using ML techniques. In this way, predictive models based on both problem-specific features as well as the MIP solver performance could provide insights about the instance hardness of a combinatorial optimization problem. Such insights, in turn, are useful in both algorithm design and solving the problem.

One of the main distinctions with previous studies in ML for algorithm selection and performance (e.g., [59, 58, 60]) is that our defined features are solely based on specific properties of the problem instances. We hypothesize that this type of features could be useful to get an understanding about instance hardness and that a trained classifier can capture possible patterns. Thus, we do not rely on MIP-generic *probing/dynamic* features [59]. A probing feature is the one computed by briefly running the algorithm for the given problem instance to extract characteristics from the algorithm's partial progression.

We define two different classification tasks and select as case studies the Maximum Stable Set Problem and its quadratic version, the Quadratic Stable Set Problem. In a first learning framework, we aim at classifying a problem-specific instance as either hard or easy when tackled by a MIP solver for the MSSP and QSSP. Second, we explore another algorithmic question formulated as a classification task specifically for the QSSP. We seek to learn a classifier that predicts whether to linearize or not the quadratic programming (QP) model representation of the QSSP when tackled with a MIP solver but relying only on problem-specific features.

We propose to use, in an offline fashion, traditional supervised learning techniques to build models for both classification tasks. The general supervised learning methodology corresponds to five main steps: (i) problem representation and instance generation, (ii) feature design from the problem-specific representation, (iii) the label definition for the learning task, (iv) dataset definition and composition, and finally, (v) the learning experiments.

We define below the combinatorial problems selected as case studies for the ML application. In addition, we present the mathematical formulations used in the experiments with the MIP solver.

4.2.1 Stable set problems

We use two classical combinatorial problems, namely, the Maximum Stable Set Problem and the Quadratic Stable Set Problem. Such problems are well-known in the discrete optimization community with several applications domains such as telecommunications, scheduling, and social networks. Moreover, these problems have appeared as auxiliary problems in different decomposition approaches [62, 63] and have been employed as benchmarks for optimization solvers [17].

Gaining insights on instance easiness/hardness for these combinatorial problems could be relevant for several applications. For instance, if the combinatorial problem is present as a subproblem in a decomposition approach, then it is likely solved several times on a regular basis keeping the same general structure and only varying its data. Then, the study of instance hardness is relevant to efficiently tackle the problems, design new optimization methods, and even, to extend over the ideas presented here for other combinatorial problems over graphs.

Maximum Stable Set Problem (MSSP)

Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, a subset of vertices $\mathcal{S} \subseteq \mathcal{V}$ is called a stable set (also referred to as independent set or vertex packing) of $\mathcal{G}$ iff there is no edge in $\mathcal{E}$ with its two endpoints in $\mathcal{S}$. Moreover, let w_i be a weight (or profit) associated with each vertex $i \in \mathcal{V}$ that is collected if vertex i is included in the stable set $\mathcal{S}$. In the so-called unweighted version, i.e., $w_i = 1$ for $i = 1, \ldots, |\mathcal{V}|$, the maximum stable set problem (MSSP) consists of finding the stable set $\mathcal{S}$ of maximum cardinality [64]. It models application in scheduling, biomedical engineering, and information retrieval, to name a few [65]. As an illustrative example, the optimal MSSP solution to the graph in Figure 4.1 is $\{1, 2, 5\}$.

A classical MIP representation to address the MSSP is the so-called *edge formulation* [64]. Let x_i be a binary variable that takes the value of 1 if vertex i belongs to the maximum stable set and 0 otherwise. A MIP model with $|\mathcal{E}|$ constraints is given by $\max\left\{\sum_{i\in\mathcal{V}} w_i x_i \ : \ x_i + x_j \leq 1, \forall (i,j) \in \mathcal{E}; x \in \{0,1\}^{|\mathcal{V}|}\right\}$.

For this study, we use a stronger integer linear programming formulation referred to as the *clique formulation*. The latter is obtained by partitioning the vertices of $\mathcal{G}$ into cliques [64], i.e., subsets of $\mathcal{V}$ where the vertices in each subset are pairwise adjacent. Let x_i be a binary variable that takes value 1 if vertex i belongs to the maximum stable set and 0 otherwise. Let $\mathcal{K}$ be a collection of (inclusion-wise) maximal cliques that cover $\mathcal{V}$. If $\mathcal{K}$ spans all edges in $\mathcal{E}$, the resulting mathematical formulation is as follows:

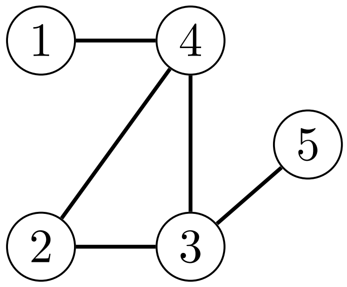


Figure 4.1 Illustrative example of a MSSP instance

$$\begin{aligned}
\max \quad & \sum_{i \in \mathcal{V}} w_i x_i && && (4.1)\\
\text{subject to} \quad & \sum_{i \in K} x_i \leq 1, && \forall K \in \mathcal{K}, && (4.2)\\
& x_i \in \{0,1\}, && \forall i \in \mathcal{V}. && (4.3)
\end{aligned}$$

Each maximal clique K can be computed in a greedy fashion following, e.g., the procedure in [6]. We initially select the vertex with highest degree and then include iteratively adjacent vertices (also with highest degree) to every vertex so far in K until no more additions are possible. Next, we add clique K to $\mathcal{K}$, remove from $\mathcal{G}$ all the edges belonging to the added clique, update the vertices degrees, and repeat the procedure.

Quadratic Stable Set Problem (QSSP)

The QSSP is a difficult variant of the maximum stable set problem where additional profits are associated with pairs of vertices. The QSSP is a key component of modern applications (e.g., protein structure prediction [66] and marketing [67]), and its quadratic nature takes the problem to a more challenging level in comparison to classical stable set problems. In addition, given the relationship between stable sets and cliques (i.e., induced complete subgraphs) in a graph, the study of the QSSP can also contribute to approaches for solving clique related problems.

Consider again an undirected graph $\mathcal{G}$. Moreover, we consider a symmetric matrix $Q = \{q_{ij}\}_{i,j=1,\ldots,n} \in \mathbb{R}^{n \times n}$ (here not restricted to be positive semidefinite), where the profit q_{ij} for each pair of vertices $i, j \in \mathcal{V}$ is collected if both vertices are included in $\mathcal{S}$. The QSSP asks for the stable set in $\mathcal{G}$ that maximizes the total profit.

Let x_i be a binary variable that takes value of 1 if vertex $i \in \mathcal{V}$ belongs to the stable set $\mathcal{S}$ and takes value 0 otherwise. Similarly to its linear case, a *clique cover*-based formulation for the QSSP is obtained by profiting from the fact that at most one vertex in any maximal clique $K \in \mathcal{K}$ can be part of the stable set. The QSSP can be formulated as the following binary quadratic program (BQP):

$$\begin{aligned} \max_{x} \quad & \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2q_{ij} x_i x_j && && (4.4) \\ \text{subject to,} \quad & \sum_{i \in K} x_i \leq 1, && \forall K \in \mathcal{K}, && (4.5) \\ & x_i \in \{0, 1\}, && \forall i \in \mathcal{V}. && (4.6) \end{aligned}$$

Note that $x_i^2 = x_i$ for all i and the linear profits w_i can be easily obtained by adjusting the terms q_{ii} in Q, i.e., the main diagonal of Q. In fact the QSSP reduces to the the MSSP when $q_{ij} = 0$ for all $i \neq j$.

4.3 Learning to classify problem-specific easy/hard instances for a MIP solver

In this section, we present the learning methodology to classify problem-specific instance hardness for a MIP solver. Before describing the learning approach for each case study, i.e., the MSSP and QSSP, we first present a simple motivating experiment to illustrate that problem structure could hide interesting patterns to be revealed.

Motivating example. We generate 500 MSSP instances based on random Erdös-Rényi graphs with the same number of nodes ($n = 175$) and graph density ($p = 10\%$). We solve each instance 5 times (changing the random seed to deal with performance variability issues [68]) using CPLEX as MIP solver. The solving time for each instance is the average CPU time of the runs. Figure 4.2 presents the distribution of computational times for all the instances (i.e., each data point corresponds to the solving time of an instance).

By observing formulation (4.1)-(4.3), it is expected that the number of vertices and the graph density have a significant impact in the MIP solver performance due to their direct relationship with the number of variables and constraints in the MIP model. Thus, as every instance has the same type of graph, size, and density, we would expect little to no variation in the solving times in this motivating experiment. However, note that the CPU time (y-axis) ranges between 41.11 and 830.48 seconds, i.e., instances which are very similar in structure present dramatic differences in MIP performance. We hence conjecture that, in addition to size and density, more characteristics of the problem instance reveal the reasons for the CPU time differences and provide insights about MSSP-instance hardness.

We use a ML methodology to learn a classifier which discriminates when a problem-specific instance could be efficiently solved by a MIP solver. We frame such a discrimination as a

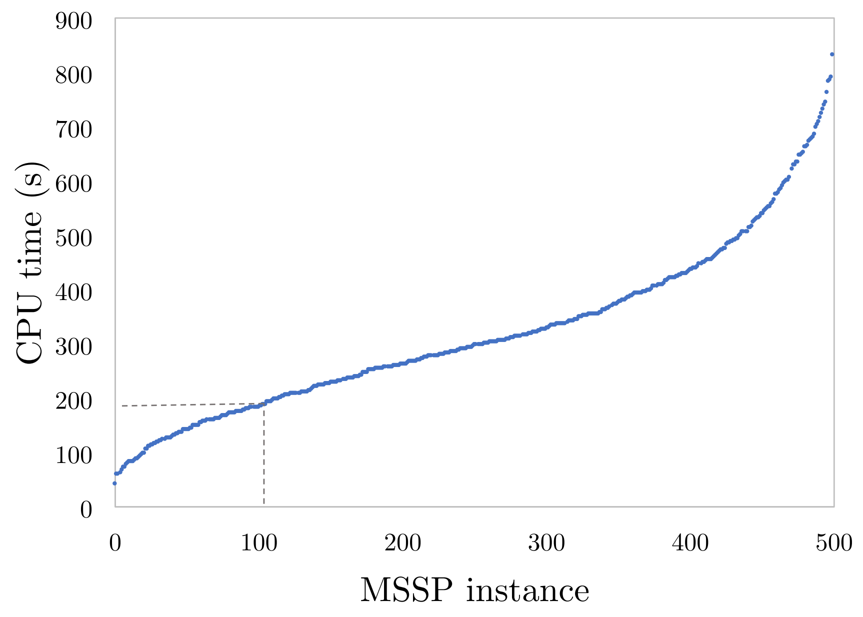


Figure 4.2 Distribution of solving times for 500 MSSP instances with the same number of vertices (175) and density value (10%)

classification task and rely on the MIP performance to label an instance as either easy or hard. Therefore, a data point consists of a problem instance, its vector of features is defined by problem-specific properties, and its label is defined according to the MIP solver performance. We present details on the different learning steps and experiments for the MSSP and QSSP in Section 4.3.1 and 4.3.2, respectively.

4.3.1 Classifying MSSP instance hardness

We now present methodological details of the learning task of classifying a MSSP instance as either easy or hard. As a data point corresponds to a MSSP instance, we define how MSSP instances are generated, the specific feature design for this problem, how each instance is labeled, and the dataset selection and composition. Finally, we extend on the experimental results for this learning task.

i. MSSP instances generation

We consider a set of randomly defined test cases to train our classifiers. Since a MSSP instance is mainly described by an undirected graph, we generate instances defined from random graphs according to different graph generator schemes, namely, the Erdös-Rényi (ER) [69], Watts-Strogatz (WS) [70], Barabási-Albert (BA) [71], and Holme-Kim (HK) [72] models. The instances are generated varying the number of nodes (n) and the graph density (p). In addition, we only consider unweighted MSSP instances, i.e., the weight for every vertex is 1.

ii. Feature design for the MSSP

As mentioned above, feature design is one of the main distinctions in this study. We only define and extract problem-specific features from graph metrics and other properties of the instance. In addition, this type of features could be particularly useful in portfolio-based algorithm selection. Suppose we aim at discriminating instance hardness when tackling a problem with different optimization paradigms (i.e., when the problem representation is not the same), in such a setting, we could rely only on problem-specific instance features.

Since a MSSP instance corresponds to an undirected graph $\mathcal{G}$, we select 17 specific features based on graph properties and metrics. We note that as we experiment with the unweighted version of the MSSP, we do not extract features from the vertex weights w_i. We now provide a more detailed description of all the considered features.

- *Number of vertices.* Number of nodes in the undirected graph $\mathcal{G}$, i.e., $n = |\mathcal{V}|$.
- *Number of edges.* Number of edges in $\mathcal{G}$, i.e., $|\mathcal{E}|$.
- *Density.* Denoted by p, it is the ratio of the number of edges and the number of possible edges. For undirected graphs, the density is defined as $p = \frac{2|\mathcal{E}|}{|\mathcal{V}|(|\mathcal{V}|-1)}$. In such a way, a dense graph has a number of edges which is close to the number of possible edges (i.e., p closer to 1), whereas a sparse graph has only few edges (p closer to 0).
- *Vertex degree.* The degree of a vertex (also called the valency) is the number of adjacent vertices, i.e., the number of edges that are incident to the vertex. Having the degrees for all the vertices in $\mathcal{G}$, we derive seven features by computing their mean, median, standard deviation (SD), minimum, maximum, the interquartile range (IR), and the variability score (SD/mean).
- *Graph assortativity.* Defined as the assortativity of a graph by degree [73]. It measures the tendency for vertices of being connected to other vertices that are similar in terms of the vertex degree.
- *Vertex connectivity.* It computes the minimum number of vertices that must be removed to disconnect $\mathcal{G}$ or make it trivial [74]. In general, algorithms to compute vertex connectivity are based on max-flow subroutines.

- *Edge connectivity.* Similarly to vertex connectivity, it computes the minimum number of edges that must be removed such that $\mathcal{G}$ is disconnected or trivial [74].

- *Graph transitivity.* The fraction of all possible triangles in $\mathcal{G}$. The graph transitivity relies on triads which are defined when two edges shared a vertex. The transitivity of a graph is then computed as the ratio of the number of triangles and the number of triads.

- *Average clustering coefficient.* The local clustering of a vertex is the fraction of triangles over the number of possible triangles in its neighborhood. The average clustering coefficient of $\mathcal{G}$ is then the mean of local clusterings. This feature computes an approximate average clustering coefficient considering a number of trials in the procedure as defined in [75].

- *Number of triangles.* For each vertex, we compute the number of triangles that include it as one of the vertices. We derive two features by calculating the mean and the maximum among the number of triangles of all the vertices.

iii. Label definition for classifying `E/H` MSSP instances

As we cast the learning task as a binary classification problem, we define a procedure to binarize the label. Each MSSP instance is solved with CPLEX version `12.8` (5 times with a different random seed each) to get the MIP solving time MIP_{time}. In addition, a threshold value t is defined to binarize the label. For each instance, we assign either the label `E` (*easy*) if $MIP_{time} \leq t$ or `H` (*hard*) otherwise (i.e., $MIP_{time} > t$). Such a threshold t could, for example, correspond to how much time we are allowed to use, within an iterative approach, to solve an instance.

iv. Dataset definition and composition

As in any learning process approach, datasets should be meaningful for the classification task. We evaluate the distribution of solving times and binarization thresholds to generate balanced and purposeful datasets. We define two of them, namely, `MSSP-EH-1` and `MSSP-EH-2`. `MSSP-EH-1` contains a wider range of instances in terms of size (n) and graph density (p) and is formed by $17,000$ MSSP instances with number of nodes $n \in \{100, 125, 150, 175, 200\}$, graph density (in percentage) $p \in \{10, 20, 30, 50, 70\}$, and threshold $t = 10$ seconds for label binarization. On the other hand, `MSSP-EH-2` has $4,000$ MSSP instances with $n \in \{200\}$, $p \in \{20, 30\}$, and $t = 40$ seconds. Note that, in the latter, we fix n and slightly vary p in order to understand which are the features that could better discriminate MSSP hardness, when n and p should present low impact in the dataset composition. Table 4.1 shows, for each dataset, the percentage of `E` and `H` instances.

Table 4.1 Label distribution in each dataset for the `E`/`H` learning experiments for the MSSP

Dataset	`E`(%)	`H`(%)	Total
`MSSP-EH-1`	8590 (50.52)	8410 (49.48)	**17000**
`MSSP-EH-2`	2479 (61.77)	1529 (38.23)	**4000**

v. Learning experiments when classifying `E`/`H` MSSP instances

For each dataset, we construct the classifier using traditional supervised learning methodologies. We randomly split the dataset into training (75%) and test set (25%). To obtain features in the same range, we apply feature scaling and mean normalization so, each feature is normalized to have mean 0 and standard deviation 1. Each experiment consists of a training phase with 5-fold cross validation and grid search method for hyperparameter tuning, and a test phase on the neutral test set. We test Logistic Regression (LR), k-Nearest Neighbors (k-NN), Support Vector Machine (SVM) with RBF kernel [25], and Random Forests (RF) [26], an ensemble method based on decision trees. As a measure of baseline performance, we compare both methods versus a dummy classifier (DUM) which makes random guessing using simple rules. In our case, we follow a stratified strategy, i.e., DUM makes predictions respecting the class distribution of the training dataset.

We implemented this methodology using Python with Scikit-learn [76] and used NetworkX [77] for the feature extraction. Table 4.2 presents the standard performance measures of binary classification, namely, accuracy, precision, recall and f1-score, for dataset `MSSP-EH-1`. Similarly, Table 4.4 shows the performance metrics for dataset `MSSP-EH-2`.

In the learning experiments, we use RF due to its interpretability. We then retrieve the importance scores which rank features based on their importance for the prediction using RF. In Table 4.3 and Table 4.5, we report the top-5 features of the learning experiment for datasets `MSSP-EH-1` and `MSSP-EH-2`, respectively.

We remark that both SVM and RF achieve high performance metrics. We conclude that both methods are able to capture enough about the MIP solver performance on training set to make meaningful predictions on test set. That is, most of the times, easy (`E`) instances are predicted as easy and, hard (`H`) instances tend to be predicted as hard. This corroborates that there is a statistical pattern to be learned when classifying easy and hard MSSP instances

Table 4.2 Performance measures for the three classifiers when predicting `E\H` for dataset `MSSP-EH-1`

	DUM	LR	k-NN	SVM	RF
Accuracy	0.495	0.928	0.947	0.957	0.959
Precision	0.509	0.925	0.957	0.967	0.971
Recall	0.504	0.937	0.939	0.949	0.948
F1-score	0.507	0.931	0.948	0.958	0.960

Table 4.3 Top-5 features ranked by importance score from RF for the `E\H` classifier for MSSP in dataset `MSSP-EH-1`

Feature	Score
Number of nodes	0.267
Number of edges	0.116
Avg. clustering	0.081
Graph transitivity	0.076
Degree SD	0.067

Table 4.4 Performance measures for the three classifiers when predicting `E\H` for dataset `MSSP-EH-2`

	DUM	LR	k-NN	SVM	RF
Accuracy	0.532	0.991	0.987	0.991	0.991
Precision	0.608	0.988	0.988	0.988	0.988
Recall	0.637	0.996	0.990	0.996	0.996
F1-score	0.622	0.992	0.989	0.992	0.992

Table 4.5 Top-5 features ranked by importance score from RF for the `E\H` classifier for MSSP in dataset `MSSP-EH-2`

Feature	Score
Avg. clustering	0.368
Graph transitivity	0.163
Avg. number of triangles	0.101
Degree SD	0.092
Degree var. score	0.059

for a MIP solver and the selected problem-specific features can capture such discrimination.

Additionally, from the feature importance scores, we observe that the importance score of the number of nodes is, by far, the most relevant feature in the discrimination of easy/hard instances for the MIP solver when we consider MSSP instances with a high variation in the number of nodes (n) which is the case of dataset `MSSP-EH-1`. However, for dataset `MSSP-EH-2`, the most important feature is the average clustering, indicating that such a graph property has a high incidence in the MSSP-instance hardness when we analyze instances with little variation in number of vertices and density.

We remark that the type of model (ER, WS, HK, etc) used to generate the random graph of the different instances is not a feature in the learning task. We also note that more random graph generator schemes exist (e.g., the $\hat{p}$–generator [78]), so more type of random graphs could be considered for the training data. Nevertheless, the performances of the classifiers suggest that the selected features successfully capture the differences in degree distributions and particular structures and also their relations with the MIP solver performance.

4.3.2 Classifying QSSP instance hardness

Similarly to the previous section, we develop the learning task of classifying a problem instance as either easy or hard, this time, for the QSSP. We describe details and results of the learning methodology for this problem.

i. QSSP instances generation

A QSSP instance mainly corresponds to an undirected graph plus a symmetric matrix of quadratic profits. We randomly generate instances to train the classifier. QSSP instances are generated following the Erdös-Rényi (ER) [69] model for the undirected graphs. In addition, each linear profit w_i is an integer uniformly distributed between -100 and 100. Finally, each profit term q_{ij} is uniformly distributed between -50 and 50 and a percentage of positive coefficients (v) in matrix Q is defined when generating the symmetric matrix.

ii. Feature design for the QSSP

We rely on the 17 graph features defined for the MSSP in Section 4.3.1 for undirected graphs. In addition, as noted in the instance generation, we define extra features associated with the quadratic profits. An important property which is reported in [79] when tackling the QSSP is the percentage of positive coefficients in Q. Moreover, we obtain features associated with the linear profits w_i. For the latter, we define the mean, median, standard deviation (SD), minimum and maximum, for a total of 23 features.

iii. Label binarization for classifying `E/H` QSSP instances

The label is defined following the same criteria used in the previous section for the MSSP. Each QSSP instance is solved with CPLEX version `12.8` using its nonlinear programming-based branch and bound. The MIP solving time MIP_{time} for each instance is the average of 5 runs with a different random seed. Similarly, a threshold value t is defined to binarize the label, i.e., we assign the label `E` if $MIP_{time} \leq t$ or `H`, otherwise.

iv. Dataset definition and composition

We assess the distribution of solving times and binarization threshold to generate again a meaningful dataset which is named `QSSP-EH-1`. The dataset is composed of 9,900 QSSP instances where $n \in \{50, 60, 70, 80, 90, 100, 150\}$, $p \in \{10, 15, 20, 25, 30, 40, 45, 50, 60, 65, 70, 75\}$, $v \in \{25, 50, 75\}$, and binarization threshold $t = 10$. Table 4.6 shows the percentage of `E` and `H` instances in the dataset.

Table 4.6 Label distribution in each dataset for the `E/H` learning experiments for the QSSP

Dataset	`E`(%)	`H`(%)	Total
`QSSP-EH-1`	4311 (43.55)	5589 (56.45)	**9900**

v. Learning experiments when classifying `E/H` QSSP instances

We follow the same experimental setup from Section 4.3.1. We use the same ML techniques to build the classifiers, define the same training-test ratio, and follow the same good learning practices. Table 4.7 presents the standard performance metrics for SVM and RF and also the baseline DUM. In particular, we also obtain the importance scores from RF and report in Table 4.8 the features appearing in the top-5 of the experiment.

From the experiments, we can observe that the classifiers exhibit good performance. We conclude after comparing with the baseline performance that the classifier does learn from the problem-specific features to make meaningful predictions on the hardness of new instances. From the feature importance scores, the number of nodes and percentage of positive coefficients in matrix Q appear as the most significant attributes when making predictions for this dataset.

Table 4.7 Performance measures for the classifiers when predicting `E\H` for the QSSP

	DUM	LR	k-NN	SVM	RF
Accuracy	0.501	0.934	0.842	0.942	0.958
Precision	0.430	0.915	0.859	0.918	0.966
Recall	0.431	0.938	0.764	0.953	0.938
F1-score	0.430	0.927	0.808	0.936	0.952

Table 4.8 Top-5 features ranked by importance score from RF classifier when discriminating `E\H` for the QSSP

Feature	Score
Number of nodes	0.184
Perc. of positive coeff. in Q	0.168
Number of edges	0.082
Graph density	0.071
Avg. clustering	0.070

4.4 Learning to linearize/not-linearize the QSSP for a MIP solver based on QSSP-specific features

When solving a quadratic programming (QP) model with a state-of-the-art solver as CPLEX, the linearization or not of the QP model determines which algorithm will be used by the MIP solver. In case that the QP is not linearized, CPLEX uses a nonlinear programming-based branch and bound where a QP relaxation is solved at each node. On the other hand, if the QP is linearized, a reformulated linear MIP model is solved with a standard branch-and-cut scheme. This algorithmic question can be crucial in the resulting solver performance. We refer the reader to [58] for more details about how a QP model is tackled by a state-of-the-art solver as CPLEX. In addition, Bonami et al. [58] present a ML framework to make a decision on this linearization alternative and a learned classifier is incorporated in the most recent version (`12.10`) of CPLEX. The classifier is based on static and dynamic MIP-specific features and then generic for any MIQP model.

In this section, we propose to use the QSSP-specific features of the learning task developed in Section 4.3.2 with the algorithmic question and learning framework proposed in [58]. In this way, we build a predictive model to classify whether to linearize or not the QSSP model (clique formulation) based only on QSSP-specific instance features. Such learned classifiers could then provide insights about QSSP structure that impacts the linearization decision in a MIP solver and then its hardness. As in previous sections, we now detail some methodological aspects of the learning methodology.

The generation of QSSP instances and the feature design is defined exactly as in Section 4.3.2 (so subsections i. and ii. are omitted). We proceed to describe the label definition for this specific task, the dataset composition, and the learning experiments.

iii. Label definition for deciding on linearization for the QSSP

The binary labeling scheme is based on the MIP performance when it linearizes and does not linearize the QSSP as proposed in [58]. Along these lines, each QSSP instance is solved twice, i.e., linearizing and not-linearizing the QP. Such decision is defined in CPLEX through the parameter `QToLin`. For each alternative, every instance is solved 3 times with a different random seed to get the solving time linearizing (resp., not-linearizing) LIN_{time} (resp. $NLIN_{time}$) as the average time of the runs. Finally, for each instance, we assign a label either `LIN` or `NLIN` according to the minimum solution time between both solving times.

iv. Dataset composition

The dataset named `QSSP-LIN-1` comprises 3600 QSSP instances with different values for the number of nodes (n), graph density (p), and percentage of positive coefficients in Q (v). Specifically, $n \in \{50, 60, 70, 80, 90, 100, 110, 120, 130, 140\}$, $p \in \{20, 25, 30, 35, 40, 50, 60, 70, 80\}$, and $v \in \{25, 50, 75\}$. Table 4.9 shows the percentage of `LIN` and `NLIN` instances in the dataset.

Table 4.9 Label distribution in each dataset for the `LIN`/`NLIN` learning experiments for the QSSP

Dataset	`NLIN`(%)	`LIN`(%)	Total
`QSSP-LIN-1`	3318 (92.16)	282 (7.84)	**3600**

We observe that we deal with a highly imbalanced dataset. In this case, most of the instances are labeled as not-linearize (`NLIN`) and the analysis on the performance metrics will have to consider this fact.

v. Learning experiments

Once again, we follow the learning setup from previous sections. We then use the same ML techniques to build the classifiers (DUM, SVM, and RF), define the same training-test ratio, and follow the same good learning practices when training the classifiers. Table 4.10 presents the standard performance metrics. Again, we obtain the importance scores from RF and report them in Table 4.11 for features appearing in the top-5 of the experiment.

As we tackle an imbalanced classification task, the accuracy is not a suitable performance metric for evaluating the model performance. We observe in Table 4.11 that even the dummy classifier presents a good accuracy because it relies on the class distribution of the training

Table 4.10 Performance measures for the classifiers when predicting `LIN\NLIN` for the QSSP

	DUM	LR	k-NN	SVM	RF
Accuracy	0.838	0.981	0.917	0.975	0.986
Precision	0.058	0.884	0.456	0.863	0.892
Recall	0.071	0.871	0.371	0.814	0.942
F1-score	0.064	0.877	0.409	0.838	0.916

Table 4.11 Top-5 features ranked by importance score from RF classifier for the QSSP when classifying `LIN\NLIN`

Feature	Score
Perc. of positive coeff. in Q	0.362
Graph density	0.099
Graph transitivity	0.085
Avg. clustering	0.074
Avg. number of triangles	0.074

set. Conversely, we search for a model which performs well at finding the relevant cases within the dataset, i.e., the ones where the solver should linearize the QP model by reformulating it as a MILP model. In this case, the precision and recall are better metrics, and we can observe that the trained classifiers present a high performance in those. The models find most of the QSSP instances of interest within the dataset and the high precision indicates the models' ability to rightly classify the QSSP instances to linearize.

Furthermore, we observe in Table 4.11 that the feature importance scores reveal that the percentage of positive coefficients in Q is by far the most important feature involved when using the random forest classifier.

4.4.1 Comparison with the automated classifier in CPLEX 12.10

In this experiment we compare the trained `LIN\NLIN` classifier versus the automated out-of-the-box classifier incorporated in CPLEX version 12.10. We take into production the `LIN\NLIN` SVM classifier and implement the QSSP in C++ using the Concert Technology Library of CPLEX version `12.10`. All experiments are run on a Linux machine, Intel(R) Xeon(R) Gold 6142 CPU @ 2.60GHz and 512 GB of RAM.

For the benchmark, we generate a challenging testbed focused on QSSP instances of intermediate size $n = \{80, 100, 120\}$, sparse graphs $p = \{20, 25, 30\}$, and wide range of percentage of positive coefficients $v = \{25, 50, 75\}$. We generate 10 instances for each combination (group) (n, p, v) for a total of 270 instances (27 groups).

Table 4.12 reports the computational experiments. Column 1 corresponds to the group id. Columns 2, 3, and 4 present, for each group, the number of nodes, density, and percentage of positive coefficients in Q. For each case, i.e., CPLEX `12.10` with both automated classifier and `LIN\NLIN` classifier, we report two values. In the first column, the number of instances classified as linearized (`LIN`) and not linearized (`NLIN`) using and exponent with such a quantity. In the second column, we present the average performance for each group where the base number corresponds to the average computational time employed for solving the ten instances. If an exponent appears, it indicates how many of those instances could not be solved to optimality within the time limit of $7,200$ seconds. Columns 5 and 6 correspond to CPLEX with its automated out-of-the-box classifier. Columns 7 and 8 are associated with CPLEX but deciding on the linearization based on the trained classifier presented in Section 4.4.

As we previously highlighted, the main difference between the two classifiers is that the one presented in this chapter is based only on problem-specific features of the QSSP. The automated classifier used in CPLEX `12.10` ([58]) is evidently a generic predictor suitable for any type of QP model tackled by the solver. Nevertheless, it is worth evaluating if the problem-specific features can capture relevant structure about the instance when deciding on whether CPLEX should linearize or not.

We observe from Table 4.12 that the automated out-of-the-box classifier in CPLEX decides that no QSSP instance in the testbed should be linearized. On the other hand, the trained `LIN\NLIN` classifier predicts instances on some groups as `LIN`. We can observe that the linearization can really payoff in some cases such as in groups 21 and 24. For such groups, linearizing the QP translates not only in reducing computational time but also in increasing the number of solved instances. The classifier then predicts an effective strategy on the linearization of the QP model when the QSSP is tackled by CPLEX. This experiment remarks the importance of problem-specific features to discriminate instance hardness.

4.5 Conclusions

We present a machine learning application to identify problem-specific instance hardness when using a MIP solver. We formulate the discrimination on instance hardness through classification tasks and use two classical combinatorial optimization problems as case studies. In a first learning framework, we classify a combinatorial problem as either easy or hard for a MIP solver based on problem-specific instance features and the MIP performance. Next, in a second classification task, we gain insights on instance hardness when classifying the best algorithm (configuration) for solving a QSSP model by CPLEX, i.e., if it should be either linearized or not to be solved.

Learning experiments show that the learned classifiers do capture a statistical pattern based

Table 4.12 Comparison between CPLEX 12.10 with its automated classifier and CPLEX 12.10 with the trained classifier of Section 4.4

Instances				**CPLEX 12.10 out-of-the-box classifier**		**CPLEX 12.10 with trained classifier**	
Group	n	p	v	Prediction	CPU time (s)	Prediction	CPU time (s)
1	80	20	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	23.76	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	24.59
2	80	20	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	31.15	$\texttt{LIN}^1 - \texttt{NLIN}^{9}$	65.31
3	80	20	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	184.75	$\texttt{LIN}^{10} - \texttt{NLIN}^{0}$	308.94
4	80	25	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	16.21	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	17.27
5	80	25	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	18.88	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	19.78
6	80	25	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	124.31	$\texttt{LIN}^{10} - \texttt{NLIN}^{0}$	228.03
7	80	30	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	8.99	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	9.47
8	80	30	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	13.65	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	14.70
9	80	30	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	59.81	$\texttt{LIN}^4 - \texttt{NLIN}^{6}$	93.81
10	100	20	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	190.45	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	194.20
11	100	20	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	303.46	$\texttt{LIN}^1 - \texttt{NLIN}^{9}$	412.19
12	100	20	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	2625.39	$\texttt{LIN}^{10} - \texttt{NLIN}^{0}$	2180.12
13	100	25	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	111.10	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	113.53
14	100	25	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	142.34	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	147.89
15	100	25	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	1089.56	$\texttt{LIN}^{10} - \texttt{NLIN}^{0}$	922.20
16	100	30	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	57.16	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	57.92
17	100	30	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	90.31	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	90.83
18	100	30	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	443.36	$\texttt{LIN}^5 - \texttt{NLIN}^{5}$	555.07
19	120	20	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	1399.85	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	1417.22
20	120	20	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	2792.59	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	2713.06
21	120	20	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	$7200.00^{(10)}$	$\texttt{LIN}^{10} - \texttt{NLIN}^{0}$	$6741.91^{(7)}$
22	120	25	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	653.57	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	662.27
23	120	25	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	1172.76	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	1205.27
24	120	25	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	$6948.45^{(7)}$	$\texttt{LIN}^{10} - \texttt{NLIN}^{0}$	$5656.36^{(2)}$
25	120	30	25	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	276.41	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	281.35
26	120	30	50	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	398.16	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	403.77
27	120	30	75	$\texttt{LIN}^0 - \texttt{NLIN}^{10}$	2598.94	$\texttt{LIN}^5 - \texttt{NLIN}^{5}$	2267.15
Geom. mean					234.44		257.24

only on problem-specific features. Hence, the predictive models allow to gain insights on instance hardness for specific problems. In addition, computational experiments show that such problem-specific features are relevant to discriminate performance when tackling MIQP models by comparing the trained classifier with an automated out-of-the-box classification feature in CPLEX.

Insights on problem-specific instance hardness for an optimization solver can be useful in several applications such as algorithm design and selection, as well as to generate harder benchmarks. In particular, we envision that the insights and learned classifiers in this work could be very useful when designing an optimization solver that incorporates MIP technology.

We select two well-known optimization problems as case studies. These problems are general enough that it is possible that features and insights obtained here could be useful for other problems. Then, the ML application could be adapted to be used with another combinatorial optimization problem over graphs.

The learning framework used in this chapter could be extended to another optimization solver that replaces the MIP solver. Furthermore, the framework can be adapted for another setting related to an algorithm-selection decision. In that case, the problem-specific features are worth to be considered because they are solver- and representation-independent. In addition, the performances from the different solvers could be used to define the target label and answer the algorithmic question of interest.

CHAPTER 5 ARTICLE 1: INTEGRATED INTEGER PROGRAMMING AND DECISION DIAGRAM SEARCH TREE WITH AN APPLICATION TO THE MAXIMUM INDEPENDENT SET PROBLEM

Authors: Jaime E. González, Andre A. Cire, Andrea Lodi, Louis-Martin Rousseau
Published: *Constraints*, 2020.[1]

Abstract: We propose an optimization framework which integrates decision diagrams (DDs) and integer linear programming (ILP) to solve combinatorial optimization problems. The hybrid DD-ILP approach explores the solution space based on a recursive compilation of relaxed DDs and incorporates ILP calls to solve subproblems associated with DD nodes. The selection of DD nodes to be explored by ILP technology is a significant component of the approach. We show how supervised machine learning can be useful to detect, on-the-fly, a subproblem structure for ILP technology. We use the maximum independent set problem as a case study. Computational experiments show that, in presence of suitable problem structure, the integrated DD-ILP approach can exploit complementary strengths and improve upon the performance of both a stand-alone DD solver and an ILP solver in terms of solution time and number of solved instances.

Keywords: Decision diagrams, Integrated methods, Integer linear programming, Supervised Learning.

5.1 Introduction

Several optimization problems can be addressed efficiently by leveraging multiple solution approaches with complementary strengths, such as integer linear programming (ILP) and constraint programming (CP) [3, 4]. Within this extensive area, successful hybrid techniques often involve an *a-priori* decomposition of the problem into two or more subproblems that are more amenable for existing techniques, such as in the case of logic-based Benders decomposition for machine scheduling [29] and CP-based branch and price [30, 31].

In this paper, we propose a novel hybrid methodology that combines elements from decision diagrams (DDs) (see, e.g., [9], [10], [11]) with ILPs for discrete optimization problems. While ILPs have an extensive body of work, only recently decision diagrams have been established as

[1]Available at [80]

viable data structure in optimization solution approaches [6]. Specifically, DD techniques are based on discrete approximations of the solution space denoted by *relaxed decision diagrams* (relaxed DDs) [14].

We contribute to the literature on decision diagram-based approaches that integrate different optimization paradigms. Relaxed DDs are used as a replacement of the linear programming relaxation in branch-and-bound methods [17], and also to strengthen ILP models through their equivalent linear reformulation as shortest paths [46]. In addition, DDs have been embedded as global constraints in CP models for sequencing problems [38, 39], as a cutting plane method for integer programming [42], and to solve the pricing problem in a branch and price scheme [44], to name a few. Furthermore, in the context of two-stage stochastic programming, DDs have also been used to model second-stage decisions which are parameterized by the first-stage variables [48].

Our methodology, in turn, provides an alternative way of exploiting relaxed DDs by further emphasizing their role as approximations of the branch-and-bound search tree of a problem. In particular, we perceive the nodes of a relaxed DD as subproblems that may involve some structure that is more suitable to another technology (in our case, ILPs), which is invoked to prune the node in advance. This process therefore involves a *dynamic* identification of subproblems as opposed to an a-priori one, and may both strengthen the relaxed DD approximation (by removing undesired nodes) as well as provide primal solutions that can further speed-up solution time.

We represent each subproblem at a node by a dynamic programming state, currently the standard for a DD formulation. Using such a state representation, we follow a supervised learning methodology to get insights and define criteria to detect a structure that is more efficiently solvable by ILP. Specifically, we describe an algorithm-selection problem where, given a node of a relaxed DD, we use a decision tree inferred from predictive models to determine the best method to solve its associated subproblem. This approach has similarities, e.g., with the concept of algorithm portfolios in SAT solvers [57].

We present a case study on the maximum independent set problem (MISP), which is typically used as a basis for novel DD methodologies due to its well-understood representation [81]. We show that, under particular structure, DD-ILP can dominate either the typical DD-based branch and bound or a leading commercial ILP solver. Most importantly, it suggests a way to enhance DD-based solvers when its relative effectiveness with respect to ILP can be evaluated efficiently and dynamically with Machine Learning.

The remainder of this paper is organized as follows. Section 5.2 describes preliminary concepts in decision diagrams for optimization. In Section 5.3, we introduce the proposed DD-ILP framework and the supervised learning methodology used to exploit the DD-ILP algorithm. Section 5.4 presents computational experiments on MISP instances. Finally, we conclude in Section 5.5.

5.2 Preliminaries and notation

Maximum Independent Set Problem. Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, an independent set of $\mathcal{G}$ is a subset of vertices $\mathcal{S} \subseteq \mathcal{V}$ such that no edge in $\mathcal{E}$ has its two endpoints in $\mathcal{S}$. The maximum independent set problem (MISP) asks for the independent set with the largest cardinality [64]. For instance, the optimum MISP solution to the graph in Figure 5.1 is $\{1, 2, 5\}$. We let $n := |\mathcal{V}|$ and denote by $\mathcal{S}(\mathcal{G})$ the set of independent sets of $\mathcal{G}$ (i.e., the solution set of the problem).

A classical ILP model to address the maximum independent set problem is the so-called edge formulation [64]. Let x_i be a binary variable that takes the value of 1 if vertex i belongs to the maximum independent set and 0 otherwise. An integer programming formulation with $|\mathcal{E}|$ constraints is given by $\max\left\{\sum_{i \in \mathcal{V}} x_i \ : \ x_i + x_j \leq 1, \forall (i,j) \in \mathcal{E}; x \in \{0,1\}^{|\mathcal{V}|}\right\}$. However, a stronger ILP formulation for the MISP is obtained by partitioning the vertices of $\mathcal{G}$ into cliques [64], i.e., subsets of $\mathcal{V}$ where the vertices in each subset are pairwise adjacent. Let x_i be a binary variable that takes value 1 if vertex i belongs to the maximum independent set and 0 otherwise. Let $\mathcal{K}$ be a collection of (inclusion-wise) maximal cliques that cover $\mathcal{V}$. The resulting ILP formulation is $\max\left\{\sum_{i \in \mathcal{V}} x_i \ : \ \sum_{i \in K} x_i \leq 1, \forall K \in \mathcal{K}; x \in \{0,1\}^{|\mathcal{V}|}\right\}$.

Each maximal clique K is computed in a greedy fashion (as in [6]) by initially selecting the vertex with highest degree and then adding iteratively adjacent vertices (also with highest degree) to every vertex so far in K until no more additions are possible. Then, we add clique K to $\mathcal{K}$, remove from $\mathcal{G}$ all the edges belonging to the added clique, update the vertices degrees, and repeat the procedure.

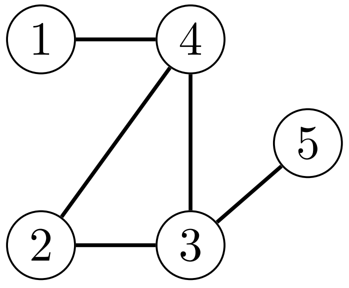
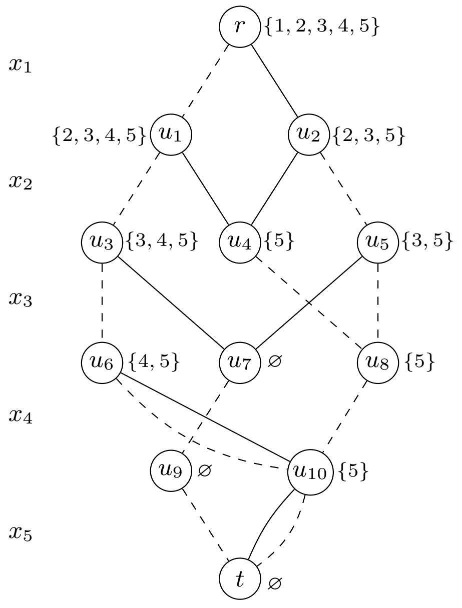


Figure 5.1 Undirected graph for the MISP



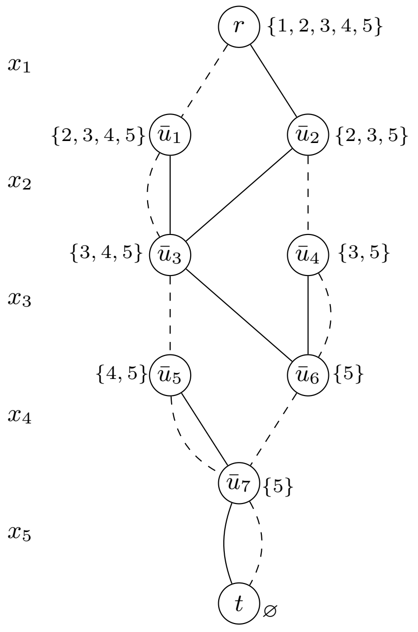


Figure 5.2 Exact DD for MISP instance of Figure 5.1

Figure 5.3 Relaxed DD for MISP instance of Figure 5.1

Exact and Approximate Decision Diagrams. A decision diagram $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ is a directed graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$ that encodes a set of solutions of an optimization problem. For the case of the MISP, the node set $\mathcal{N}$ is partitioned into $n+1$ layers $L_1, \dots, L_{n+1}$. The first and last layers are singletons and contain the root node r and the terminal node t, respectively. Arcs emanating from nodes in $L_1, \dots, L_n$ are associated with the variables $x_1, \dots, x_n$, respectively. Let $l(u)$ be the index of the layer associated with node $u \in \mathcal{N}$. Each arc $a = (u, u') \in \mathcal{A}$ only connects adjacent layers, i.e., $l(u') = l(u) + 1$, and its layer corresponds to the one of its tail node, $l(u)$. Moreover, a label $d(a) \in \{0, 1\}$ of an arc a at layer i represents the assignment $x_i = d(a)$; we denote an arc with $d(a) = 0$ and $d(a) = 1$ by 0-arc and 1-arc, respectively. An arc-specified path $(a_1, \dots, a_n)$ from the root node to the terminal node encodes the solution $(d(a_1), \dots, d(a_n))$ to the MISP. Finally, a DD is called *exact* if the set of solutions encoded in $\mathcal{D}$ is $\mathcal{S}(\mathcal{G})$.

Dynamic programming (DP) is used as the conceptual basis to compile exact DDs. We follow the same DP formulation presented in [34] for the MISP. Let $N(i)$ be the neighborhood of vertex $i \in \mathcal{V}$ including i, i.e., $N(i) = \{i' | (i, i') \in \mathcal{E}\} \cup \{i\}$. With each node u of $\mathcal{D}$ we associate a state information $s(u) \subseteq \mathcal{V}$ that represents the vertices that can be added to the partial independent set encoded by any path from the root to u. The root node's state is fixed as $s(r) = \mathcal{V}$. Each node $u \in L_i$ has an outgoing 0-arc that leads to a node u' with state

$s(u') := s(u) \setminus \{i\}$ (i.e., we exclude the vertex from the independent set). If $i \in s(u)$, then u also has an outgoing 1-arc that leads to a node u'' with state $s(u'') := s(u) \setminus N(i)$ (i.e., we include the vertex and remove its neighbourhood). No two nodes in a layer have the same state. Typically exact DDs are constructed in a top-down fashion, where layers are processed in the order $L_1, \ldots, L_n$ one at a time, adding arcs and merging nodes with the same state as required.

Exact DDs are in general of exponential size and, in practice, we prefer to manipulate approximate (controllable-size) decision diagrams such as the relaxed [17] and restricted [16] versions. A decision diagram is called *relaxed* if it over-approximates the solution set of a problem ($\mathcal{S}(\mathcal{G})$ for the MISP case). Figures 5.2 and 5.3 depict an exact and a relaxed DD for the graph in Figure 5.1, respectively, where solid arcs represent $d(a) = 1$ and dashed arcs represent $d(a) = 0$.

Relaxed DDs can be obtained similarly as exact DDs when using a top-down approach. In particular, if the number of nodes in a layer (i.e., the layer width) exceeds a given pre-specified limit W during its construction, two nodes u and u' are heuristically selected and merged into a new node u''. The state of this node is set as $s(u'') := s(u) \cup s(u')$ to ensure all valid independent sets are preserved. The strategy used to define which nodes are merged consists on selecting DD nodes u and u' with the partial longest path from the root node. Once nodes are merged, all previous incoming arcs to u and u' are directed to the new merged node. Figure 5.3, in particular, presents a relaxed DD with maximum width of 2 ($W \leq 2$). Notice that, if the length of an arc is set as $d(a)$, the longest path of an exact DD provides the optimal MISP, while the longest path value of a relaxed DD provides an upper bound to the problem.

Another type of approximate decision diagrams are the so-called restricted DDs which are under-approximations of the solution set of a problem. Restricted DDs can also be obtained in a top-down construction. Once the maximum width W is reached when constructing a layer, we heuristically remove nodes from such a layer. In this procedure, we evidently remove feasible solutions and possibly the optimal one too. However, a longest path computation on a restricted DD provides a feasible solution and a lower bound on the optimal solution value.

DD-based Branch and Bound. Relaxed DDs can play the role of a search tree in a branch-and-bound scheme [17]. The main idea is that the solution space of a problem can be divided and explored by branching recursively on decision diagram nodes as opposed to branching on a variable-value pair.

Specifically, consider a relaxed decision diagram $\bar{\mathcal{D}}$ of an optimization problem. For every pair of nodes $u, u' \in \bar{\mathcal{D}}$ such that $l(u) < l(u')$, we define $\bar{\mathcal{D}}_{uu'}$ as the decision diagram induced by all the nodes and arcs that lie on directed paths from u to u'; e.g., $\bar{\mathcal{D}}_{rt} = \bar{\mathcal{D}}$. We say that a node u in $\bar{\mathcal{D}}$ is *exact* if all $r - u$ paths lead to the same state $s(u)$, and is relaxed otherwise. A *cutset* of $\bar{\mathcal{D}}$ is a subset of nodes C such that any $r - t$ path in $\bar{\mathcal{D}}$ contains at least one node in C. In particular, C defines an exact cutset if all nodes in C are exact. Note that the removal of C from $\bar{\mathcal{D}}$ disconnects r and t. Several strategies exist for obtaining cutsets [6], such as *the frontier cutset (FC)* and *the last exact layer (LEL)*. The frontier cutset of a relaxed decision diagram $\bar{\mathcal{D}}$, FC($\bar{\mathcal{D}}$), is the set of exact nodes in $\bar{\mathcal{D}}$ such that, for each node, at least one of its outgoing arcs' heads is a relaxed node. For instance, FC($\bar{\mathcal{D}}$) in the relaxed DD of Figure 5.3 is defined by $\{\bar{u}_1, \bar{u}_4\}$. The LEL cutset LEL($\bar{\mathcal{D}}$), in turn, is defined by the last layer where all nodes are exact (i.e., where no two nodes were forcefully merged to impose the maximum width). For the case of Figure 5.3, LEL($\bar{\mathcal{D}}$) is defined by $\{\bar{u}_1, \bar{u}_2\}$.

Let C be an exact cutset obtained either by FC or LEL. DD-based branch and bound explores each node in C separately to find and prove optimal solutions. Namely, for each $u \in C$, let $v^*(u)$ be the longest-path value from r to u. If z^*_u is the optimal value of the subproblem for which its solutions are exactly encoded in $\mathcal{D}_{ut}$, then $v^*(u) + z^*_u$ is the value of the best solution across all $r - t$ paths that contain u. Since all $r - t$ paths of a decision diagram must contain some node in C, we can solve the subproblems associated with $\mathcal{D}_{ut}$ separately for all $u \in C$ to search for the optimal solution, each subproblem leading to a smaller and hence more tractable DD. Such a procedure can be applied recursively for each u if required.

5.3 A hybrid DD-ILP approach

We propose a novel strategy named *ILP-based pruning* to integrate ILP technology into DD-based branch and bound. Once we define a cutset C and select a node $u \in C$ to explore, we can solve the subproblem associated with u using ILP as opposed to recursively relaxing and exploring $\mathcal{D}_{ut}$, since only its optimal solution z^*_u is required for our purposes. Such a decision is made based on the properties of the subproblem encoded by the state $s(u)$. In particular for the MISP, a subproblem corresponds to a vertex-induced subgraph defined by the vertices in $s(u)$. In a nutshell, once an exact cutset C is defined, we explore each node of C either by recursively applying DD-based branch and bound, or by directly invoking an ILP model to prune the node in advance.

For solving the MISP instance in Figure 5.1 using the DD-ILP framework, we initially compile a relaxed decision diagram $\bar{\mathcal{D}}$ (like the one in Figure 5.3) to compute an upper bound on the optimal solution. For this case, the longest path from r to t is equal to 4 which provides a dual bound on the optimal value of the objective function. Then, we identify all DD nodes in an exact cutset which are included in a list of branching nodes to explore further e.g., $C = \{\bar{u}_1, \bar{u}_2\}$. Next, in a pure DD solver scheme, we would select a DD node to branch on from the list of open branching nodes. For instance, Figures 5.4(a) and 5.4(b) present the decision diagrams recursively obtained when branching on $\bar{u}_1$ and $\bar{u}_2$. Following a branch and bound scheme, we would update the upper and lower bounds whereas we recursively explore the solution space until we prove that the incumbent solution is optimal. Note that, for instance, the DD rooted in $\bar{u}_1$ (Fig. 5.4(a)) is relaxed and following a pure DD-based exploration, we would have to compile new relaxed decision diagrams from a selected exact cutset in $\mathcal{D}_{\bar{u}_1 t}$.

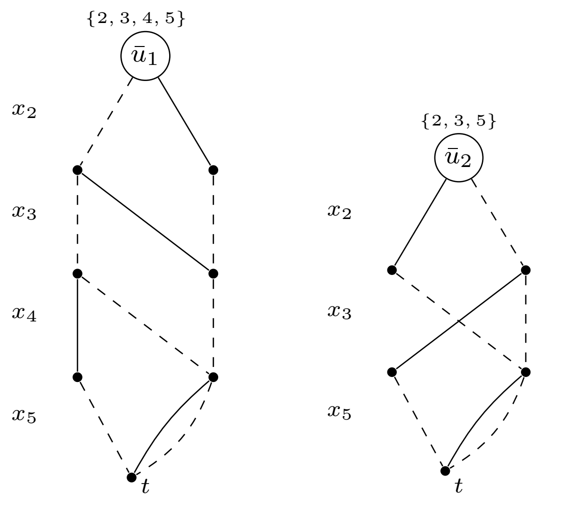
Nevertheless, in the context of our proposed hybrid DD-ILP framework, the evaluation of properties of the subproblem encoded by a DD node can lead us to determine that such a node should be pruned by solving it with ILP technology. For instance, after evaluating DD nodes properties, one possible scenario could be applying the ILP-based pruning strategy to DD node $\bar{u}_1$ but not to DD node $\bar{u}_2$. Figure 5.5 illustrates how the solution space exploration is done within the DD-ILP approach. On the one hand, we generate a relaxed DD which is rooted in $\bar{u}_2$; on the other hand, we prune $\bar{u}_1$ by solving an ILP subproblem with a classical LP-based branch and bound tree (small subtree with green and red nodes rooted in $\bar{u}_1$). Note that solving to optimality the ILP subproblem associated with $\bar{u}_1$ allows us to prune the DD node, get a feasible solution (i.e., a lower bound equals to $v^*(\bar{u}_1) + z^*_{\bar{u}_1}$), and avoid the compilation and further exploration of more relaxed DDs.

Furthermore, the DD-ILP framework can be enhanced by different strategies that combine information from the ILP and the partial solutions enumerated by the DD. For instance, assume that we define a time cut-off to solve the ILP at a node u, obtaining a lower bound $\underline{z}_u$ and an upper bound $\overline{z}_u$ as opposed to the optimal solution z^*_u. Recalling that $v^*(u)$ is the optimal longest-path value from r to u, we derive two simple methodologies:

ILP-cutoff Pruning. The value $v^*(u) + \overline{z}_u$ corresponds to an upper bound to the solutions encoded by $r - t$ paths crossing u. If such a value is lower than the current incumbent solution, the node can be pruned from the DD without losing optimality. We can also update the relaxed states of a DD after the removal of a node, which by itself can lead to additional pruning based on DD filtering methods.

ILP-cutoff Heuristic. Since u is exact, $v^*(u) + \underline{z}_u$ corresponds to a primal heuristic value to the MISP and can be used to update the current incumbent, also possibly triggering the pruning of other nodes.

When the optimal solution value z^*_u is found, we can immediately prune the node and update



(a) DD rooted in $\bar{u}_1$ (b) DD rooted in $\bar{u}_2$

Figure 5.4 Compiled DDs after branching on DD nodes in $C = \{\bar{u}_1, \bar{u}_2\}$

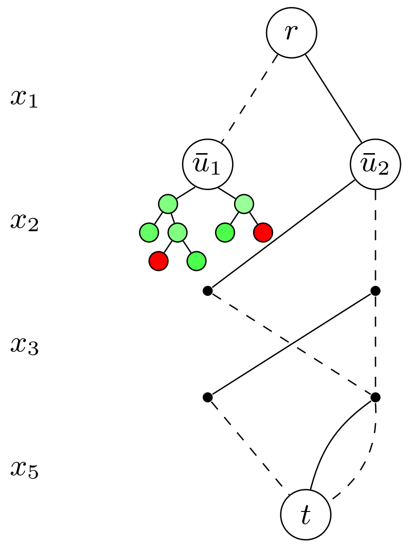


Figure 5.5 Integrated DD-ILP search tree.

the incumbent solution, if needed. Otherwise, we can use the bounds as above and proceed with DD-based branch and bound.

In addition, the complementarity offered by the hybrid DD-ILP scheme reveals an additional procedure to speed-up the subproblems' solution when calling the ILP solver. For any of the three ILP-based strategies, we can provide a global incumbent solution to the ILP solver, namely, a *lower cutoff* c. This lower cutoff indicates that the objective value of a given subproblem has to be at least c. Let LB be a global incumbent solution obtained from the hybrid DD-ILP exploration. Each time the ILP solver is called to solve a subproblem u, we include the constraint $z_u \geq c$ to the corresponding ILP model. In particular for the MISP, we include the constraint

$$\sum_{i \in s(u)} x_i \geq LB - v^*(u). \tag{5.1}$$

The constraint (5.1) takes into account that $c = LB - v^*(u)$ and $z_u = \sum_{i \in s(u)} x_i$ since the vertices considered in the subproblem are the ones defined by the state $s(u)$. As a result of this procedure, when exploring a subproblem, the ILP solver may terminate significantly earlier with the proof that no feasible solution exists, which leads to prune the DD node. Note that the ILP-based pruning strategy involves solving to optimality several ILP problems associated with DD nodes. In general, integer linear programming is NP-hard and a straightforward application of the ILP-based pruning could be computationally very expensive. Therefore, a systematic identification of subproblems that can be efficiently solved by ILP technology is key for the hybrid method to pay off. In Section 5.3.1, we define a machine learning approach useful to design the exploration mechanisms within the hybrid DD-ILP algorithm.

5.3.1 Supervised learning to identify complementarity

Recently, machine learning (ML) has served as an important tool to enhance discrete optimization solvers, see, e.g., [53, 58] and [50] for a recent survey. We propose to use ML to build predictive models which lead us to define a decision-making tool that classifies when a node should be explored by an ILP model or by a DD-based branch and bound. This approach is similar to a portfolio-based algorithm selection [57]. The distinction, however, is that our subproblems are defined dynamically during search, as driven by our DD construction. The classification is based on the node state $s(u)$, which contains all the information required to define the subproblem associated with u (i.e., $\mathcal{D}_{ut}$).

We present three different learning experiments which lead to three different classifiers. The first two classifiers aim at predicting the most suitable method (either ILP or DD solver) for tackling a particular instance. They are closely related and presented in Section 5.3.1. Next, in a third experiment, we focus only on the ILP solver performance and learn a classifier to discriminate when an instance could be efficiently solved by an ILP solver. This classifier is presented in Section 5.3.1. The methodology for all the experiments consists of four steps: dataset generation, features design, label definition, and learning experiments.

Learning to classify between ILP or DD solver

We describe, for the case of MISP, the four steps in the methodology.

1. Dataset Generation. We consider a set of randomly defined test-cases to train our classifier. Since subproblems related to DD nodes correspond to vertex-induced subgraphs, we generate a dataset composed of random graphs according to different graph generator schemes, namely, the Erdös-Rényi (ER) [69], Watts-Strogatz (WS) [70], Barabási-Albert (BA) [71], and Holme-Kim (HK) [72] models. In addition, to include even more different graph structures in the test bed, we generate additional instances as follows. We create two graphs ($\mathcal{G}_1$, $\mathcal{G}_2$) resulting from different graph generator models. We then connect both graphs by defining a number of edges e which link randomly selected nodes from $\mathcal{G}_1$ and $\mathcal{G}_2$. We combine each pair of graph models from the named four generators, giving us a total of six new graph types, that we name ERWS (i.e., combination of an ER graph and a WS graph), ERBA, ERHK, WSBA, WSHK, and BAHK. The number of nodes (n) is defined as $n = \{100, 125, 150, 175, 200\}$ and the density (p), in percentage, takes values from the set $p = \{10, 20, 30, 50, 70\}$. Finally, our dataset has $18,500$ instances.

2. Feature Design. We define and extract features that could better discriminate the performance between ILP and DD-based branch and bound. As we compare two different representations of the problem, we only define features from graph metrics and properties of the instance. Since an instance for the MISP corresponds to an undirected graph, we select 17 specific graphical features: number of nodes (n), number of edges ($|\mathcal{E}|$), density (p), graph assortativity, vertex connectivity, edge connectivity, graph transitivity, the average clustering coefficient, and the average and maximum number of triangles. We also derive seven features that come from node degrees: mean, median, standard deviation (SD), minimum, maximum, the interquartile range (IR), and the variability score (SD/mean).

3. Label Definition. We now need to establish the performance of each method based on our instance set. We solve each MISP instance with both ILP and DD solvers. For ILP solver and to deal with performance variability issues [68], we solve each instance 5 times with a different random seed using `IBM-CPLEX 12.8`. Then, the ILP solution time for each instance is the average of the 5 runs. Finally, for each instance, we assign a label (either `ILP` or `DD`) according to the minimum solution time between both methods.

4. Supervised Learning Experiments. Finally, we construct the classifier using traditional supervised learning methodologies. We randomly split the dataset into a training set and a test set, with 13875 (75%) and 4625 instances (25%), respectively. Each feature is normalized to have a mean 0 and a standard deviation 1. Each experiment consists of a training phase with 5-fold cross validation to grid search the hyper-parameters, and a test phase on the neutral test set. We test Support Vector Machines (SVM) with RBF kernel [25] and Random Forests (RF) [26]. As a measure of baseline performance, we compare both methods versus a dummy classifier (DUM), which follows a stratified strategy.

We implemented this methodology using Python with Scikit-learn [76]. Table 5.1 presents the standard performance measures for binary classification, namely, accuracy, precision, recall and f1-score. We observe high accuracy scores obtained from both SVM and RF. This corroborates that there is a statistical pattern to be learned when selecting the best method between ILP and DD solvers. In addition, the designed features can capture such discrimination. In the experiments, we use RF due to its interpretability. Scikit-learn provides scores which rank features based on their importance for the RF prediction. We report them in Table 5.2 for features appearing in the top-5 of the experiment.

We observe in Table 5.2 that features related to density (1^{st}), number of triangles (2^{nd}), average clustering (3^{rd}), graph transitivity (4^{th}) and the degree of nodes (5^{th}) significantly discriminate ILP and DD.

Table 5.1 Performance measures for the three classifiers when predicting `ILP` or `DD`

	DUM	SVM	RF
Accuracy	0.543	0.963	0.962
Precision	0.309	0.936	0.944
Recall	0.330	0.951	0.937
F1-score	0.319	0.943	0.941

Table 5.2 Top-5 features ranked by importance score from RF

Feature	Score
Density	0.2179
Avg. number of triangles	0.2147
Avg. clustering	0.1481
Graph transitivity	0.1090
Avg. degree	0.0661

In a second learning experiment, we learn a classifier that predicts whether solving an instance with an ILP solver is *significantly easier* than using a DD solver. From the previous experiment, we slightly modify the label definition based on the solving times as follows. For each instance, we assign the label `ILP` if the ILP solver is x times faster than the DD solver, otherwise we assign the label `NO-ILP`. Following the same methodology for the previous learning experiment, we obtain the performance metrics presented in Table 5.3.

Table 5.3 Performance measures for the three classifiers when predicting `ILP` or `NO-ILP`

	DUM	SVM	RF
Accuracy	0.656	0.989	0.989
Precision	0.222	0.983	0.981
Recall	0.212	0.967	0.969
F1-score	0.216	0.975	0.975

We observe that both the SVM and RF models achieve high performance metrics in the classification. In addition, such metrics are significantly better than the baseline performance obtained by the dummy classifier showing that the learning models are actually learning something useful from the data.

Learning to classify easy/hard MISP instances for an ILP solver

In a third learning experiment, we aim at classifying whether a MISP instance will be efficiently solved by an ILP solver. For this experiment, we use the same dataset and features than in subsection 5.3.1, so steps 1 and 2 of the methodology are already described. However, in this case, we redefine the label to focus on the ILP solver explaining this in steps 3 and 4.

3. Label Definition. Each MISP instance is solved with `IBM-CPLEX 12.8` (5 times with a different random seed to deal with performance variability issues) to get the ILP solving time ILP_{time}. In addition, a threshold value t is defined to binarize the label. Finally, for each instance, we assign the label `E` (*easy*, i.e., $ILP_{time} \leq t$) or `H` (*hard*, i.e., $ILP_{time} > t$).

4. Supervised Learning Experiments. We construct the classifier using the three supervised learning algorithms presented in subsection 5.3.1. In addition, we follow the same strategy for splitting the dataset, normalizing the features, and performing the training phase.

Table 5.4 presents the standard performance measures for binary classification. Similarly to the previous experiment, we remark that SVM and RF achieve high performance scores. Therefore, we conclude that both methods are able to capture enough about the ILP solver performance on training set to make meaningful predictions on test set. This is, most of the time, easy (`E`) instances are predicted as easy and, hard (`H`) instances tend to be predicted as hard.

Additionally, we report the top-5 most important features from RF in Table 5.5. We observe that the importance score of the number of nodes is, by far, the most involved feature in the discrimination of easy/hard instances for the ILP solver in this dataset.

From here, the three trained classifiers (i.e., `ILP/DD`, `ILP/NO-ILP`, and `E/H`) and the insights obtained from them are used to guide the node exploration into the hybrid method and benchmark its performance.

5.3.2 Learning to explore within a hybrid DD-ILP for MISP

The hybrid approach proposed for the MISP profits from the learning experiments in Section 5.3.1 to incorporate mechanisms to guide, on-the-fly, the exploration of the solution space. The hybrid is presented in Algorithm 1.

At the beginning, we include the initial DD root node r (which corresponds to the initial

Table 5.4 Performance measures for the three classifiers when predicting `E` or `H`

	DUM	SVM	RF
Accuracy	0.505	0.976	0.977
Precision	0.461	0.954	0.956
Recall	0.461	0.997	0.997
F1-score	0.461	0.975	0.976

Table 5.5 Top-5 features ranked by importance score from RF

Feature	Score
Number of nodes	0.483
Number of edges	0.082
Degree SD	0.069
Avg. clustering	0.046
Degree IR	0.044

Algorithm 1 Hybrid DD-ILP for MISP

```
    Input: MISP instance, ILP/NO-ILP classifier, decision tree ($n_{ILP}$, $p_{ILP}$)
    Output: $z_{opt}$
 1: initialize $L = \{r\}$, where $r$ corresponds to the DD root node (initial state)
 2: let $z_{opt} = -\infty$ and $v^*(r) = 0$
 3: extract features $ft(r)$ from subproblem associated with $r$
 4: prediction $\leftarrow$ ILP/NO-ILP classifier($ft(r)$)
 5: if prediction = ILP then
 6:     $z_r^* \leftarrow$ solve_ILP($r$)
 7:     $z_{opt} = v^*(r) + z_r^*$
 8:     return $z_{opt}$
 9: while $L \neq \emptyset$ do
10:     $u \leftarrow$ select_node($L$), $L \leftarrow L \backslash \{u\}$
11:     extract features size $n_u$ and density $p_u$
12:     if decision_tree($u$) = ILP then
13:         $z_u^* \leftarrow$ solve_ILP($u$), $v^*(\mathcal{D}_{ut}) = v^*(u) + z_u^*$
14:         if $v^*(\mathcal{D}_{ut}) > z_{opt}$ then
15:             $z_{opt} \leftarrow v^*(\mathcal{D}_{ut})$
16:     else
17:         create relaxed DD $\overline{\mathcal{D}}_{ut}$ with root $u$ and $v_r = v^*(u)$
18:         if $\overline{\mathcal{D}}_{ut}$ is exact then
19:             if $v^*(\overline{\mathcal{D}}_{ut}) > z_{opt}$ then
20:                 $z_{opt} \leftarrow v^*(\overline{\mathcal{D}}_{ut})$
21:         if $\overline{\mathcal{D}}_{ut}$ is not exact then
22:             if $v^*(\overline{\mathcal{D}}_{ut}) > z_{opt}$ then
23:                 let $C$ be an exact cutset of $\overline{\mathcal{D}}_{ut}$
24:                 for all $u'$ in $C$ do:
25:                     let $v^*(u') = v^*(u) + v^*(\overline{\mathcal{D}}_{uu'})$, add $u'$ to $L$
26:             create restricted DD $\mathcal{D}'_{ut}$ with root $u$ and $v_r = v^*(u)$
27:             if $v^*(\mathcal{D}'_{ut}) > z_{opt}$ then $z_{opt} \leftarrow v^*(\mathcal{D}'_{ut})$
28: return $z_{opt}$
```

state) on the list of open nodes L. Next, we initialize the optimal solution value z_{opt} and the longest path from the root node to itself $v^*(r)$. For such initial subproblem, we want to predict if solving it by ILP technology could be much easier than using a pure DD branch-and-bound. For this purpose we use once the trained `ILP/NO-ILP` classifier developed in Section 5.3.1. If the classifier predicts `ILP`, we simply prune the root node with ILP technology solving the problem to optimality. This automated ML-driven feature of the hybrid framework allows to avoid an unnecessary DD-based exploration.

Conversely, if the classifier predicts the root node as `NO-ILP`, we start the DD-based exploration. While open DD nodes remain in L, we select a node $u \in L$ to be explored. The search strategy used to select nodes from L follows a best-first search algorithm which is based on the upper bound obtained from the relaxed DD in which u was created. Let $v^*(\mathcal{D}_{ru}) = v^*(u)$ be the longest path from the root node r to u and $v^*(\mathcal{D}_{ut})$ the longest path from u to the terminal node t. Since u is an exact node which was identified in an exact cutset from a previously computed relaxed DD, $v^*(u)$ is known. Next, we have to determine if (a) we solve node u to optimality by finding $v^*(\mathcal{D}_{ut})$ with an ILP solver, or (b) if we create a new relaxed DD $\overline{\mathcal{D}}_{ut}$ to compute an upper bound $v^*(\overline{\mathcal{D}}_{ut})$ on $v^*(\mathcal{D}_{ut})$.

Thus, we must predict whether we invoke the ILP solver to prune the node with the ILP-based pruning strategy or not. This prediction could possibly be performed by evaluating both the `ILP/DD` and `E/H` classifiers. However, these classifications require collecting, on-the-fly, expensive graph features for every selected DD node and the computational cost does not pay off for the overall DD-ILP performance. Therefore, we propose to get a proxy for the classifiers predictions by evaluating a unique and computationally inexpensive decision tree, specifically defined from significant features identified when training the classifiers.

The feature importance scores obtained during training of the `ILP/DD` and `E/H` classifiers are useful to get insights and select the features to use in a simple decision tree. We select the most important feature during training for each classifier, i.e., the density (Table 5.2) and size (number of nodes in Table 5.5) of the graph. Thus, in the DD-ILP framework, we only compute two features, the size n_u and density p_u of the subproblem (i.e., associated vertex-induced subgraph) encoded by each selected DD node u. In addition, we define two threshold values, the maximum size (n_{ILP}) and maximum density (p_{ILP}) to fully describe the decision tree. To sum up, the evaluation, on-the-fly, of the decision tree presented in Figure 5.6 determines whether the ILP solver is invoked to prune a particular DD node.

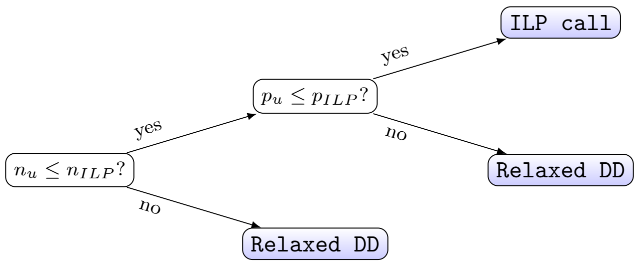


Figure 5.6 Decision tree to be evaluated at each DD node within the DD-ILP framework

In case the decision tree evaluates node u as `ILP call`, we prune the node by finding the optimal solution for such subproblem z_u^* with an ILP call. Solving the ILP subproblem associated with the DD node u provides $v^*(\mathcal{D}_{ut})$ which is used to compute z_u^* as $v^*(\mathcal{D}_{ut}) + v^*(u)$. Note that z_u^* is a lower bound on z_{opt} and no more exploration from u is needed. Otherwise, we create a relaxed DD $\overline{\mathcal{D}}_{ut}$. If $\overline{\mathcal{D}}_{ut}$ is exact (i.e., the maximum width W is never exceeded during compilation) there is no need of further branching from node u and we update the lower bound if necessary. On the contrary, if $\overline{\mathcal{D}}_{ut}$ is not exact, we compute an upper bound on the optimal solution for such subproblem u and identify an exact cutset C of $\overline{\mathcal{D}}_{ut}$ to include the nodes in C to the list L. In addition, we obtain a lower bound on the optimal solution of subproblem u by compiling a restricted DD $\mathcal{D}'_{ut}$.

5.4 Computational experiments with the DD-ILP approach

We perform experiments in order to compare the performance of the DD-ILP framework with respect to both the stand-alone DD solver proposed in [17] and a commercial ILP solver. We use `IBM-CPLEX 12.8` as ILP solver in single-thread mode with default parameter settings. We implement the DD-ILP approach in C++, solving the ILP subproblems also with `IBM-CPLEX 12.8`. All experiments are run on a Linux machine, Intel(R) Xeon(R) CPU E5-2637 v4 at 3.50GHz (16 threads) and 128 GB RAM.

Our purpose is to evaluate the DD-ILP approach performance against both the ILP solver and DD-based branch and bound, in order to verify the effectiveness of our hybrid framework. The only strategy used in the DD-ILP solver is the ILP-based pruning, i.e., we solve the subproblems without any time limit when the ILP solver is called. We also consider the lower cutoff procedure. The variable ordering and exact cut selection strategies used for all the experiments, both for the DD-ILP and for the stand-alone DD solver, are *the minimum number of states (MIN)* and *the frontier cutset (FC)*, respectively. We refer the reader to

[6] where different variable ordering heuristics and exact cutset strategies are defined. The maximum width W in both cases is 128.

From the experiments in Section 5.3.1, we determine that small-size ($n \leq 150$) instances are efficiently solved by either the DD solver (specially, dense cases) or the ILP solver (mainly in sparse cases). Instances which are very sparse ($p \leq 10\%$) represent the hardest case to be solved. On the other hand, really dense instances (where $p > 40\%$) are efficiently solved by both the ILP and, specially, the DD solver. For that reason, in this section, we focus on intermediate-size instances.

We generate random instances with $n = \{250, 300\}$ and $p = [15, 30]\%$ according to the different 10 families defined in Section 5.3.1, namely, ER, WS, BA, HK, ERWS ERBA, ERHK, WSBA, WSHK, and BAHK. We define a group as a tuple family-size-density and consider 25 instances for each of the 28 groups considered in this test bed, for a total of 700 MISP instances[2]. For example, group ER-250-20 corresponds to Erdös-Rényi instances with 250 nodes and density 20%.

All the instances are processed three times by solving them with: the ILP solver, the stand-alone DD solver, and our DD-ILP framework. Each solver run uses only one thread with a time limit of $7,200$ seconds. For the DD-ILP solver, we exactly use the Algorithm 1 (presented in Section 5.3.2). Specifically, the SVM presented in Section 5.3.1 is used as the trained `ILP/NO-ILP` classifier for the DD root node. Regarding the decision tree (Figure 5.6) evaluated at each DD node explored in the DD-based branching tree, after experimentation with the threshold values, we set $n_{ILP} = 180$ and $p_{ILP} = 20$ for all groups of instances.

Table 5.6 compares the performance of the three methods for the 25 instances of each group. Column 1 presents the group label. Columns 2 and 3 are related to the ILP solver and show, for each family, how many instances out of 25 were solved to optimality within time limit and the average solving time of the 25 instances (in seconds). Columns 4, 5, and 6 are associated with the stand-alone DD solver and present the number of instances solved to optimality, the average number of DD nodes explored, and the average solving time, respectively. Finally, columns 7, 8, 9, 10, and 11 are related to the hybrid DD-ILP solver performance for each group of instances. Column 7 presents the number of instances solved to optimality within time limit. Column 8 shows the number of instances (out of 25) where the root DD node was predicted by the classifier as `ILP`. Columns 9, 10, and 11 present the average number of DD nodes explored, the average number of ILP subproblems solved, and the average solving time.

[2]This set of instances is available at `https://github.com/jaimegonzalezj/Instances-MISP-DDILP-paper.git`

First we focus on analyzing the three solvers in terms of number of instances solved and average solving time (columns 2, 3, 4, 6, 7, and 11). For example, group ER-250-20 (i.e., Erdös-Rényi (ER) instances with 250 nodes and density 20%), ER-300-20, and BA-300-20 are the hardest groups of instances to be solved no matter the method. For all methods, the 25 instances of each group hit the time limit before being solved to optimality.

In 10 out of 28 groups, ILP outperforms both DD and DD-ILP. However, in 4 out of those 10 groups, namely BA-250-30, BA-300-30, HK-250-30, and HK-300-30, all instances are efficiently solved (on avg. in less than 8 seconds) by any of the three methods, solving to optimality the 25 instances in each of these groups.

On the other hand, in 4 out of the 28 groups (ER-250-30, ER-300-30, WS-250-30, WS-300-30), DD solver outperforms both ILP and DD-ILP solvers. Specifically, note that for group WS-300-30, the DD-ILP and the DD solvers perform almost equally and the average solving time is much better than the one obtained with the ILP solver.

Interestingly, the proposed DD-ILP approach outperforms both the ILP and the DD solvers in 11 out of 28 groups. For example, for group ERBA-300-20, the DD-ILP approach solves to optimality all 25 instances whereas the ILP solver and the DD solver solve, 18 and 19, respectively. It is even more remarkable if we observe the overall performance (700 instances), the proposed hybrid DD-ILP approach solves more instances to optimality than the other two methods with the smallest average solving time.

It is worth mentioning that we also tested the hybrid DD-ILP without providing a lower cutoff (i.e., constraint (5.1) in Section 5.3) each time the ILP solver is called. On average for all the groups, the version presented in Table 5.6 (with respect to the hybrid version without the lower cutoff) has an improvement of 11.8% on the solving time and for some groups this improvement is up to 30%.

In conclusion, for groups of instances where ILP seems much better than DD, the hybrid DD-ILP profits from the algorithm-portfolio feature incorporated through the `ILP/NO-ILP` trained classifier at the root DD node. Furthermore, the most interesting fact is that for some family of instances (e.g., ERWS, ERBA, ERHK), the algorithm-portfolio feature is not enough. Then, the hybrid mechanisms (i.e., ILP-based pruning strategy) exploit the complementary strengths when integrating two different representations to get the best performance.

Table 5.6 Comparison between ILP, DD, and Hybrid DD-ILP solvers for each group of instances

	ILP solver		DD solver			Hybrid DD-ILP				
Family group	# instances solved	Avg. CPU time (s)	# instances solved	Avg. DD nodes explored	Avg. CPU time (s)	# instances solved	# classified `ILP` at root	Avg. DD nodes explored	Avg. # of subILPs	Avg. CPU time (s)
ER-250-20	0	7200.00	0	4759587	7200.00	0	0	277653	2100	7200.00
ER-250-30	22	4598.24	25	1283024	225.30	25	0	1283024	0	233.03
ER-300-20	0	7200.00	0	3535260	7200.00	0	0	3679	322	7200.00
ER-300-30	0	7200.00	25	5880746	1235.96	25	0	5880746	0	1278.64
WS-250-20	25	1040.45	25	641324	2279.15	25	0	641324	0	2371.83
WS-250-30	25	656.78	25	47732	60.87	25	0	47732	0	62.74
WS-300-20	6	6708.84	9	711026	6464.05	9	0	711026	0	6457.05
WS-300-30	25	3878.02	25	160139	205.64	25	0	160139	0	213.75
BA-250-20	5	6610.66	0	2591519	7200.00	25	0	18125	1769	2368.19
BA-250-30	25	6.42	25	5377	6.56	25	0	4741	7	7.89
BA-300-20	0	7200.00	0	2398494	7200.00	0	0	14886	3165	7200.00
BA-300-30	25	2.62	25	4353	7.20	25	12	1925	2	4.92
HK-250-20	25	288.13	0	2020336	7200.00	25	25	1	1	322.09
HK-250-30	25	2.80	25	5165	8.11	25	19	1223	1	4.20
HK-300-20	16	4593.45	0	1787542	7200.00	6	6	76140	8368	5908.74
HK-300-30	25	1.95	25	3687	8.58	25	24	105	1	2.47
ERWS-250-20	25	1125.28	25	1424150	3543.42	25	17	8213	118	824.15
ERWS-300-21	4	6996.75	0	2208236	7200.00	25	0	361452	1562	3631.58
ERBA-250-20	25	754.79	25	1402841	1199.26	25	0	30519	304	486.39
ERBA-300-20	18	5650.41	19	6132721	5761.62	25	0	358830	1617	2706.06
ERHK-250-20	25	657.10	25	1497968	1217.93	25	3	25978	284	441.55
ERHK-300-20	22	5175.26	21	5708902	5313.47	25	0	358705	1457	2437.91
WSBA-250-14	25	158.00	24	493894	2201.70	25	25	1	1	127.64
WSBA-300-15	25	1873.90	7	508490	6290.94	20	17	157450	17	2770.48
WSHK-250-14	25	153.28	24	316572	1499.97	25	21	1822	7	127.72
WSHK-300-15	25	1508.87	12	511939	5947.28	24	21	52217	5	1603.00
BAHK-250-14	25	327.51	0	1108809	7200.00	25	25	1	1	311.65
BAHK-300-15	3	6977.28	0	1055527	7200.00	0	0	116538	10601	7200.00
Total	496	88546.79	416	48205360	108277.01	559	215	10594195	31710	63503.67
Average		3162.39			3867.04					2267.99

5.4.1 Comparison versus a traditional portfolio-based algorithm selection approach

In this experiment, we compare the hybrid DD-ILP with a classic portfolio-based algorithm approach. For this purpose, we deploy into production the `ILP/DD` trained classifier (described in Section 5.3.1) to define the portfolio algorithm as follows.

At the root node, we classify the problem either as `ILP` or `DD`. Then, we simply solve the problem using the corresponding solver according to the prediction. The comparison of the portfolio version with the hybrid approach allows us to assess the impact of the ILP-based pruning strategy within the hybrid DD-ILP algorithm.

Table 5.7 presents the performance metrics for the hybrid DD-ILP and the portfolio approach. The instances are aggregated by family instead of group as in Table 5.6. Column 1 presents the family label and column 2, the number of instances per family. Columns 3, 4, and 5 are related to the hybrid DD-ILP algorithm whereas columns 6, 7, and 8 are associated with the portfolio approach. Column 3 shows, for each family, how many instances were solved to optimality within time limit. Column 4 presents the number of instances where the root DD node was predicted as `ILP` by the `ILP/NO-ILP` classifier. Column 5 shows the average solving time for each family. Column 6 presents the number of instances solved to optimality by the portfolio approach. Column 7 shows the number of instances predicted as `ILP` by the `ILP/DD` classifier. Finally, column 8 presents the average solving time.

Table 5.7 Comparison between DD-ILP and Portfolio approach on number of solved instances and average solving time for each family

Instances		Hybrid DD-ILP			Portfolio Approach		
Family	# of instances	# solved instances	`ILP/NO-ILP`	Avg. CPU time (s)	# solved instances	`ILP/DD`	Avg. CPU time (s)
ER	100	50	0	3977.92	50	50	3975.31
WS	100	84	0	2276.34	84	2	2216.68
BA	100	75	12	2395.25	56	97	3456.25
HK	100	81	74	1559.38	93	100	1088.19
ERWS	50	50	17	2227.87	32	50	3875.8
ERBA	50	50	0	1596.23	44	48	3040.47
ERHK	50	50	3	1439.73	46	49	2914.96
WSBA	50	45	42	1449.06	50	50	734.46
WSHK	50	49	42	865.36	50	50	610.45
BAHK	50	25	25	3755.83	31	50	3532.66
Total	700	559			536		

We observe the benefits of the hybrid DD-ILP, through its ILP-based pruning strategy, mainly for families BA, ERWS, ERBA, and ERHK. The performance on these families shows that there is a significant difference between applying a classic algorithm portfolio and the proposed hybrid approach.

5.4.2 Sensitivity analysis on the ILP-based pruning strategy

We also want to observe the effect of varying the threshold values of the decision tree (n_{ILP}, p_{ILP}) on the overall hybrid DD-ILP performance. To illustrate the algorithm performance on a per-instance basis, we take the 25 instances of group ERHK-300-20 and solve them with the DD-ILP framework by using 5 different (n_{ILP}, p_{ILP}) combinations. Note that for all these instances the `ILP/NO-ILP` classifier never leads us to prune the DD root node with ILP (column 7 in Table 5.6). Thus, we can analyze a similar behavior of the hybrid DD-ILP only considering the effect of the ILP-based pruning strategy in this group.

We consider the combination ($n_{ILP} = 180$, $p_{ILP} = 20$) as the base case since those were the threshold values set in the previous experiments. Then, we evaluate four additional threshold combinations: ($n_{ILP} = 180$, $p_{ILP} = 17$), ($n_{ILP} = 180$, $p_{ILP} = 15$), ($n_{ILP} = 150$, $p_{ILP} = 20$), and ($n_{ILP} = 210$, $p_{ILP} = 20$). We choose such values to aim at evaluating the effect of small variations of n_{ILP} and p_{ILP} with respect to the base case combination. Note that the n_{ILP} threshold must be between 0 and n. On the other hand, the density of induced subgraphs does not generally differ much from the density of the original instance graph so, p_{ILP} should be close to p.

Figure 5.7 compares the solution time of the ILP solver versus the DD-ILP approach for the 25 instances of group ERHK-300-20. We only compare the running time with the ILP solver's time because, on average for this group, the ILP solver dominates the DD solver (Table 5.6). Each marker corresponds to a MISP instance where the color and shape indicates the (n_{ILP}, p_{ILP}) combination used in the DD-ILP algorithm. The marker location above the diagonal means that DD-ILP approach outperforms the ILP solver in such instance for the corresponding (n_{ILP}, p_{ILP}) combination.

In general, we can observe that the parameters (n_{ILP}, p_{ILP}) used in the decision tree do affect the DD-ILP framework due to the significant differences in performances between the five combinations. We remark that several points are located far and above from the diagonal implying that DD-ILP approach is much better in those instances in comparison with the ILP solver. We also notice that the DD-ILP approach with $(n_{ILP}, p_{ILP}) = (180, 20)$ (represented by blue rhombus) achieves the best performance. None of those 25 instances are located below the diagonal. Conversely, if we observe the DD-ILP performance with either $(n_{ILP}, p_{ILP}) = (180, 15)$ (green triangles) or $(n_{ILP}, p_{ILP}) = (150, 20)$ (yellow circles), we can observe more markers below the diagonal, i.e., for the related instances, ILP solver outperforms DD-ILP framework for such threshold combination.

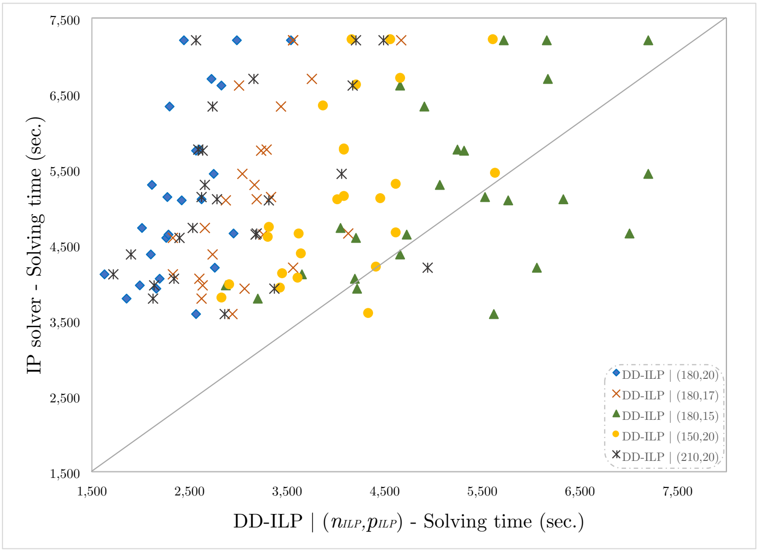


Figure 5.7 DD-ILP solver (using 5 different thresholds) versus the ILP solver in terms of solution time per instance of group ERHK-300-20

The solving time reduction obtained by the DD-ILP approach is explained by a smaller number of DD nodes explored in the DD-based branch and bound. Evidently, by the way in which the ILP-based pruning strategy is defined, there exists a trade-off between the number of ILP subproblems solved and the number of DD-nodes explored. The latter, in comparison with a pure DD solver. It is then worth mentioning the computation time spent solving the ILP subproblems within the hybrid DD-ILP. For the base case combination $(n_{ILP}, p_{ILP}) = (180, 20)$ in this group of instances, the average total solution time is $2,437.91$ seconds while the average total time solving the ILP subproblems is $1,555.29$. That is, the 64% of the algorithm time is consumed solving the ILP subproblems that are on average $1,457$ ILPs for this group.

The box plot presented in Figure 5.8 compares the stand-alone DD solver (red box) against the DD-ILP framework with the five different combinations varying the (n_{ILP}, p_{ILP}) thresholds. The plot shows, in log scale, the five-number summary (minimum, first quartile, median, third quartile, and maximum) of the number of DD nodes explored for the 25 instances of group ERHK-300-20. On the other hand, Figure 5.9 presents a box plot for the number of ILP subproblems solved for the same group (ERHK-300-20) using the different (n_{ILP}, p_{ILP}) combinations in the DD-ILP solver. To ease the analysis, note that we use the same color for each combination (n_{ILP}, p_{ILP}) in Figures 5.7, 5.8, and 5.9.

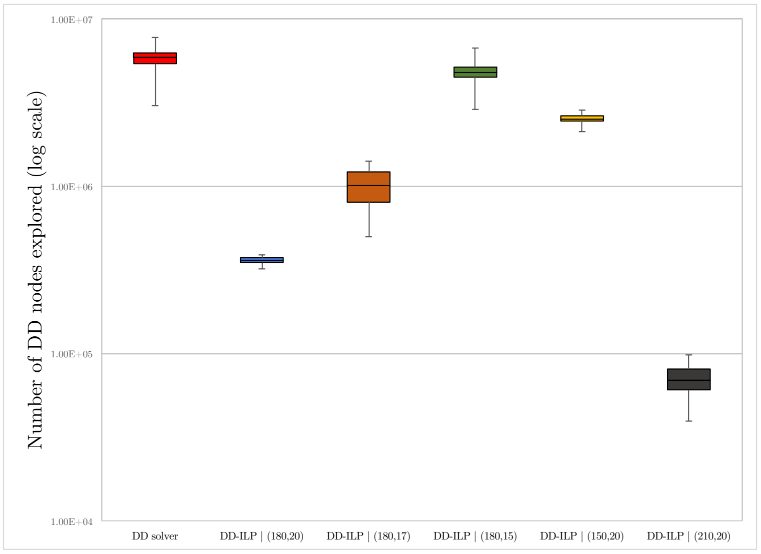


Figure 5.8 DD-ILP solver (using 5 different thresholds) versus the DD solver, in terms of the number of DD nodes explored for instances of group ERHK-300-20

From Figure 5.8, we observe that the number of DD-nodes explored with the pure DD solver is greater than such number for any combination of the DD-ILP approach. For example, on average, we can reduce 98.7% of the DD-nodes explored with the stand-alone DD solver (red box) by using the proposed DD-ILP approach with $(n_{ILP}, p_{ILP}) = (210, 20)$ (black box). If we compare the DD-ILP solver with $(n_{ILP}, p_{ILP}) = (180, 20)$ (blue box) and the DD-ILP solver $(n_{ILP}, p_{ILP}) = (210, 20)$ (black box), we note that the latter remarkably explores less DD-nodes. When we notice the number of ILP subproblems solved per instance by each configuration (Figure 5.9), DD-ILP $(n_{ILP}, p_{ILP}) = (210, 20)$ (black box) is close to the number reported by DD-ILP $(n_{ILP}, p_{ILP}) = (180, 20)$ (blue box). However, for the $(n_{ILP}, p_{ILP}) = (210, 20)$ combination, the ILP calls occur in DD nodes associated at subproblems with a larger number of vertices which are likely located in higher layers in the relaxed DD representations.

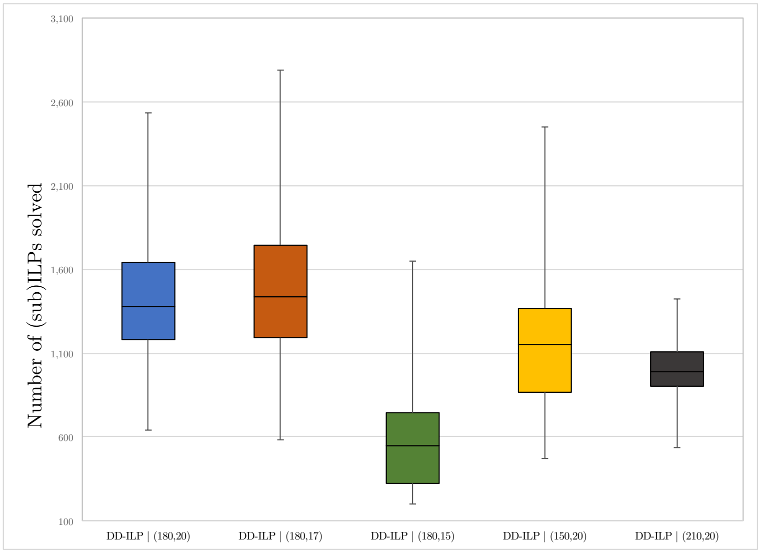


Figure 5.9 Number of subILPs solved in the DD-ILP solver (using 5 different thresholds) for instances of group ERHK-300-20

For this particular case, such earlier ILP calls (i.e., with the $((n_{ILP}, p_{ILP}) = (210, 20))$ combination) lead to prune more DD-nodes which is consistent with the behavior observed in Figure 5.8. However, by observing Figure 5.7, we note that this strategy does not pay off because DD-ILP with $(n_{ILP}, p_{ILP}) = (180, 20)$ outperfoms $(n_{ILP}, p_{ILP}) = (210, 20)$. In particular, on average for these instances, DD-ILP with $(n_{ILP}, p_{ILP}) = (180, 20)$ is faster than the DD-ILP with $(n_{ILP}, p_{ILP}) = (210, 20)$. We can observe such a superior performance in Figure 5.7 because in general the blue markers are located to the left of the black markers. This analysis indicates that a good approach is not just to prune as much as we can with the ILP calls but to use the ILP-based pruning in a more strategical way where it really exploits complementary strengths.

As we anticipated, the mechanism to classify the DD nodes in which the ILP-based pruning is performed is a critical component for the hybrid DD-ILP performance. In particular with this experiment, we can observe the significant impact when varying the threshold values in the classification. Moreover, the most remarkable fact is that the hybrid DD-ILP works effectively when using this simple decision tree.

5.5 Conclusion

In this paper, we propose a generic hybrid DD-ILP approach that is in principle suitable to any discrete optimization problem for which both a (mixed) integer programming formulation and a decision diagram representation are available. Our methodology consists of identifying DD nodes as subproblems through their associated states. We then introduce an original way to profit from an ILP representation when solving such subproblems. In addition, we use a supervised learning approach to derive a trained classifier and a decision-making tool which guide the exploration by verifying whether an ILP would efficiently prune a DD node.

The supervised learning experiments (Section 5.3.1) are essential for designing the proposed hybrid approach. From the trained classifiers and their feature importance scores, we incorporate mechanisms for node selection into the DD-ILP algorithm. The `ILP/NO-ILP` trained classifier is used at the DD root node, as a ML-driven black-box predictor to incorporate an algorithm-portfolio like feature that proves to be effective. Moreover, we profit from the feature importance scores (obtained when training the `ILP/DD` and `E/H` classifiers) to get insights and derive a decision tree that effectively proxies the classification needed for applying the ILP-based pruning strategy. Finally, the `ILP/DD` trained classifier is deployed in a traditional portfolio-based algorithm selection to benchmark the hybrid DD-ILP.

Computational results on the maximum independent set problem show that the DD-ILP approach can be effective if the DD representation reveals a problem structure that the ILP exploits well. In addition, we show that the framework works effectively even when using a simple decision tree to classify DD nodes. For the group of instances where the hybrid DD-ILP is shown to be superior, we observe that the problem structure is exploited by complementary strengths that are only leveraged through the proposed methodology. Then, it could be worth to include families of instances proposed in this paper, in other combinatorial optimization problems over graphs to compare the algorithms' performance.

We highlight that there is still room for research to define a unique and efficient trained classifier to identify DD nodes that can be pruned by ILP technology. In any case, the classification mechanism to be included within the hybrid algorithm is problem-specific and a feature where another machine learning approach (e.g., reinforcement learning) can be adopted. In addition, another research direction from the proposed hybrid DD-ILP is related to the search strategy. We experiment with a best-first search algorithm but novel search strategies could exploit the complementarity offered by the hybrid approach.

This work suggests a research avenue where decision diagrams are used to explore and enumerate subproblems which can be tackled by other technologies, such as mixed-integer programming or constraint programming. In future research, we plan to extend the hybrid algorithm to other cases where a DD representation could be advantageous, such as in sequencing and scheduling problems.

CHAPTER 6 ARTICLE 2: BDD-BASED OPTIMIZATION FOR THE QUADRATIC STABLE SET PROBLEM

Authors: Jaime E. González, Andre A. Cire, Andrea Lodi, Louis-Martin Rousseau
Published: *Discrete Optimization*, 2020.[1]

Abstract: The quadratic stable set problem (QSSP) is a natural extension of the well-known maximum stable set problem. The QSSP is NP-hard and can be formulated as a binary quadratic program, which makes it an interesting case study to be tackled from different optimization paradigms. In this paper, we propose a novel representation for the QSSP through binary decision diagrams (BDDs) and adapt a hybrid optimization approach which integrates BDDs and mixed-integer programming (MIP) for solving the QSSP. The exact framework highlights the modeling flexibility offered through decision diagrams to handle nonlinear problems. In addition, the hybrid approach leverages two different representations by exploring, in a complementary way, the solution space with BDD and MIP technologies. Machine learning then becomes a valuable component within the method to guide the search mechanisms. In the numerical experiments, the hybrid approach shows to be superior, by at least one order of magnitude, than two leading commercial MIP solvers with quadratic programming capabilities and a semidefinite-based branch-and-bound solver.

Keywords: Decision Diagrams, Hybrid Optimization, Quadratic Stable Set Problem, Binary Quadratic Programs, Dynamic Programming.

6.1 Introduction

A stable set is a pairwise non-adjacent subset of vertices in a graph and defines a fundamental structure in discrete optimization. Classical problems associated with stable sets, such as the *maximum stable* (or *independent*) *set problem*, have been extensively studied in the optimization literature and arise in applications such as social networks [83], data mining [65] and computational biology [84], to name a few. We refer to Wu and Hao [85] for a survey of existing methodologies and other applications.

Of growing interest in the optimization community is the *quadratic stable set problem* (QSSP), a difficult variant of the maximum stable set problem where additional profits are associated

[1]Available at [82]

with pairs of vertices. The QSSP is a key component of modern applications (e.g., protein structure prediction [66] and marketing [67]), but its related computational methodologies are still limited in comparison to classical stable set problems. Moreover, given the natural relationship between stable sets and cliques (i.e., induced complete subgraphs) in a graph, the study of the QSSP can contribute to approaches for solving clique related problems. To the best of our knowledge, the QSSP first appeared as a subproblem when estimating the quality of cellular networks [86], later addressed through mixed-integer programming (MIP) reformulations [63]. Karimi and Ronagh [87] also investigate Lagrangian methods and report experiments for instances of up to 30 vertices.

The QSSP can also be formulated as a binary quadratic problem (BQP) and addressed via non-linear solvers. In this context, Furini and Traversi [88] develop a semidefinite programming (SDP) relaxation that is used as a bounding mechanism in a branch-and-bound search, solving instances with up to 100 vertices. In a related approach, another generic methodology for solving the QSSP is BiqCrunch [89], a state-of-the-art semidefinite-based solver which has solved instances having up to 150 vertices. The QSSP was also used as a benchmark in a computational study of different linearization techniques for BQPs [79] using a MIP solver, demonstrating that current generic methodologies are still limited to graphs with 150 vertices. Nevertheless, leading commercial MIP solvers incorporate flexible quadratic programming (QP) capabilities that allow to directly tackle the BQP formulation of the QSSP through nonlinear programming (NLP) based branch and bound. Such an alternative is worth being considered as another benchmark for this problem.

In this paper, we propose a novel QSSP solution approach based on *decision diagrams* (DDs). The theory and practice of DDs for optimization has been gradually established as a fruitful research area in operations research [6]. A variety of stand-alone methodologies, decomposition approaches, and integrated techniques based on decision diagrams contributed to novel state-of-the-art methods in a large array of applications [17, 80, 39, 42, 49, 40]. In particular, decision diagrams have been effective for non-linear integer optimization problems [46, 47] as they provide a *discrete* type of relaxation which can be leveraged as an alternative bounding mechanism.

Our exact methodology exploits the strength of both DD-based relaxations and QP capabilities in MIP technology to solve the QSSP more efficiently. Specifically, we model and solve the QSSP via a hybrid approach which integrates binary decision diagrams (BDDs), MIP, and machine learning, following the generic framework proposed in [80]. The hybrid BDD-MIP algorithm is based on a BDD-based search mechanism that branches on underlying equivalent classes of variable assignments, here represented as states in a dynamic programming reformulation of the problem. We describe a novel BDD representation for the QSSP and highlight the flexibility of the decision diagram modeling framework to deal with quadratic problems. We also remark how the hybrid approach leverages two different representations of a problem in a collaborative framework.

Moreover, supervised learning plays an important role within the hybrid approach to guide the exploration of the solution space. In our algorithm, we use traditional machine learning to train a classifier which dynamically selects whether a branch should be explored either by MIP or BDD technology. We present computational experiments using the proposed BDD-MIP algorithm and compare its performance against general-purpose approaches; namely, two leading commercial MIP solvers with QP capabilities and, a competitive semidefinite-based solver. Numerical results show that the BDD-MIP can significantly outperform such solvers in existing benchmarks.

The remainder of the paper is organized as follows. Section 6.2 defines the quadratic stable set problem and introduces relevant notation. Section 6.3 presents the decision diagram representation for the problem, which relies on a novel dynamic programming reformulation of the problem. Section 6.4 describes the hybrid BDD-MIP optimization approach adapted for the QSSP. Finally, the experimental evaluation is presented in Section 6.5, while Section 6.6 contains concluding remarks.

6.2 The quadratic stable set problem

Let $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ be an undirected graph where $\mathcal{V} := \{1, \ldots, n\}$ is a set of n vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. A stable set of $\mathcal{G}$ is a subset of vertices $\mathcal{S} \subseteq \mathcal{V}$ where no two vertices are connected by an edge in $\mathcal{E}$. With each vertex $i \in \mathcal{V}$ we associate a profit w_i that is collected if vertex i is included in $\mathcal{S}$. Moreover, we consider a profit q_{ij} for each pair of vertices $i, j \in \mathcal{V}$ that is collected if both vertices are included in $\mathcal{S}$. We denote by $Q = \{q_{ij}\}_{i,j=1,\ldots,n} \in \mathbb{R}^{n \times n}$ the profit matrix, here not restricted to be positive semidefinite.

The quadratic stable set problem (QSSP) asks for the stable set in $\mathcal{G}$ with maximum total profit. For a mathematical formulation, let us define x_i as a binary variable that takes value of 1 if vertex $i \in \mathcal{V}$ belongs to the stable set $\mathcal{S}$ and 0 otherwise. The QSSP can be formulated as the following binary quadratic program (BQP):

$$\max_x \quad \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2q_{ij} x_i x_j \tag{6.1}$$

$$\text{subject to} \quad x_i + x_j \leq 1, \qquad \forall \{i,j\} \in \mathcal{E}, \tag{6.2}$$

$$x_i \in \{0,1\}, \qquad \forall i \in \mathcal{V}. \tag{6.3}$$

Note that $x_i^2 = x_i$ for all $i \in \mathcal{V}$ then the linear profits w_i can be easily adjusted and consider the terms q_{ii} in Q, i.e., its main diagonal. An alternative formulation for the QSSP is obtained by leveraging the concept of a *clique cover*, i.e., a partition of the vertices of $\mathcal{G}$ into cliques. Namely, let $\mathcal{K}$ be any collection of cliques that covers $\mathcal{G}$. Since each stable set can contain at most one vertex in a clique $K \in \mathcal{K}$, the *clique formulation* for the QSSP is obtained by replacing inequalities (6.2) with:

$$\sum_{i \in K} x_i \leq 1, \qquad \forall K \in \mathcal{K}, \tag{6.4}$$

which is well-known to provide stronger relaxations when only linear costs are considered [64]. The set $\mathcal{K}$ is typically constructed using maximal cliques that can be computed efficiently, e.g., by a greedy procedure [6]. In particular, a maximal clique K is obtained by initially selecting the vertex with highest degree. Next, we iteratively include adjacent vertices (sorted by highest degree) to each vertex in K until no more inclusions are possible. We then add clique K to set $\mathcal{K}$ and remove from $\mathcal{G}$ all the edges belonging to the included clique. We update the vertices degrees, and repeat the procedure.

As an illustrative example, consider the QSSP instance defined by an undirected graph with 5 vertices and 5 edges presented in Figure 6.1. Red numbers next to each vertex $i \in \mathcal{V}$ correspond to the vertex profit w_i. We also provide the matrix of quadratic profits Q. In the figure, vertices 1, 2 and 5 form a stable set whose total profit is -3 yielded by $w_1 + w_2 + w_5 + 2 \cdot q_{12} + 2 \cdot q_{15} + 2 \cdot q_{25}$. The optimal stable set for the illustrative example consists of vertices 1 and 2 with optimal objective value of 9.

$$Q = \begin{bmatrix} 0 & 3 & 2 & 0 & -4 \\ 3 & 0 & -2 & -5 & -3 \\ 2 & -2 & 0 & 0 & 1 \\ 0 & -5 & 0 & 0 & 2 \\ -4 & -3 & 1 & 2 & 0 \end{bmatrix}$$

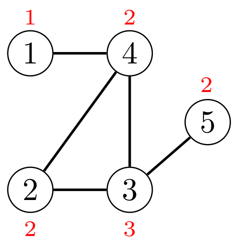


Figure 6.1 Undirected graph and matrix Q for an illustrative QSSP example

For notation purposes, we denote by z^* the optimal objective value of the QSSP. We also note in passing that the problem reduces to the classical maximum weighted stable set problem either when $Q = \mathbf{0}$, or when the quadratic profits are non-positive, i.e., $Q \leq \mathbf{0}$, the latter case requiring a non-trivial transformation of the original graph [86]. In addition, we observe that when only quadratic profits are considered (i.e., $w_i = 0$, $i \in \mathcal{V}$), the resulting particular QSSP could be reformulated as a *maximum edge-weighted clique problem* [90, 91, 92, 93] in the complement graph $\bar{\mathcal{G}}$ where profits q_{ij} become the weights on edges $(i, j) \in \bar{\mathcal{E}}$. Nevertheless, since the maximum stable set problem is NP-hard in the strong sense [94], the same is true for the QSSP, which in turn is generally more computationally challenging than the classical linear version.

6.3 A decision diagram representation for the QSSP

Conceptually, a decision diagram is a compressed representation of the state-transition graph of a dynamic programming (DP) model [6]. In this section, we begin by deriving a novel problem representation for the QSSP through a DP formulation (Section 6.3.1). Next, we describe how to extract the resulting decision diagram representation from such formulation (Section 6.3.2), which will be central to our exact methodology.

6.3.1 A dynamic programming model for the QSSP

Before introducing our DP formulation to QSSP, we first reformulate (6.1)-(6.3) in order to reveal recursive structure. Namely, note that the quadratic model

$$\begin{aligned} \max_{x,s} \quad & \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n-1} x_i s_i && && (6.5) \\ \text{subject to} \quad & x_i + x_j \leq 1, && \forall \{i,j\} \in \mathcal{E}, && (6.6) \\ & s_i = \sum_{j=i+1}^{n} 2q_{ij} x_j, && \forall i \in \mathcal{V}, && (6.7) \\ & x_i \in \{0,1\}, && \forall i \in \mathcal{V}. && (6.8) \end{aligned}$$

is also valid to QSSP, where the difference with respect to the original model is the introduction of variables s_i representing the inner sum within each quadratic term i of the objective function (6.1).

Using model (6.5)-(6.8), we can now evaluate the marginal impact of adding a vertex $i \in \mathcal{V}$ to a given stable set as follows. Suppose that variables $x_1, \ldots, x_{i-1}$ are fixed to values that can be extended to a feasible solution (e.g., by appending variables $x_i, \ldots, x_n$ and setting them to zero), thereby defining a stable set $S := \{k \in \{1, \ldots, i-1\} \colon x_k = 1\}$. This, in turn, leads to the *eligible* set

$$\mathcal{I} := \{j \in \mathcal{V} \colon \{j,k\} \notin \mathcal{E} \text{ for all } k \in S\}$$

of all vertices that can still be added to the stable set S; i.e., vertex i can be added to S (i.e., x_i set to one) only if $i \in \mathcal{I}$. Furthermore, if $i \geq 2$ is added to S, we collect a profit $w_i + s_i$ (and only a profit of w_i if $i = 1$). Thus, to evaluate whether i can be added to a (partial) stable set and the resulting profit, it suffices to have the set $\mathcal{I}$ of eligible vertices and the partial sum s_i as defined in (6.7).

This leads to the following DP reformulation of QSSP. Given a fixed vertex ordering $1, \ldots, n$, we consider a system of $n+1$ stages where, at each stage i, we decide whether to add or not a vertex $i \in \mathcal{V}$, as represented by the value of variable $x_i \in \{0,1\}$. The state of the system at each stage i is a pair $(\mathcal{I}, s)$, where $\mathcal{I} \subseteq \mathcal{V}$ is the set of eligible vertices according to the assignment in stages $1, \ldots, i-1$ and $s = (s_i, s_{i+1}, \ldots, s_n)$ is a vector of $n-i+1$ elements with the sums (6.7) indexed from i to n. We remark that the definition of state s is related to the state information defined for unconstrained binary quadratic programming in [45]. When assigning x_i (i.e., deciding whether to include i or not), we transition to a new state defined by the function

$$g_i(\mathcal{I}, s, x_i) := \begin{cases} (\mathcal{I} \setminus N_i, (s_{i+1} + 2q_{i+1\,i}, \ldots, s_n + 2q_{ni})), & \text{if } x_i = 1, \\ (\mathcal{I}, (s_{i+1}, \ldots, s_n)), & \text{otherwise.} \end{cases}$$

where $N_i = \{i\} \cup \{j \in \mathcal{V} \colon \{i, j\} \in \mathcal{E}\}$ is the neighborhood of i including the vertex itself. That is, we update the eligible set $\mathcal{I}$ according to x_i and the sum state s according to (6.7). The total profit in terms of x_i is

$$h_i(\mathcal{I}, s, x_i) := w_i x_i + \mathbb{I}(i \geq 2) s_i x_i,$$

where $\mathbb{I}(C)$ is an indicator function that evaluates to 1 if condition C is true and 0 otherwise. Finally, a vertex i can only be added if it belongs to the eligibility set $\mathcal{I}$, i.e., the set of feasible assignments at a stage i is

$$\mathcal{F}_i(\mathcal{I}) := \{0\} \cup \{\, \mathbb{I}(i \in \mathcal{I}) \,\}.$$

Equipped with $\mathcal{F}_i(\cdot)$, the transition function $g_i(\cdot)$, and the profit function $h_i(\cdot)$, an optimal solution x^* to QSSP solves the Bellman equations

$$V_i((\mathcal{I}, s)) = \max_{x_i \in \mathcal{F}_i(\mathcal{I})} \left\{ h_i(\mathcal{I}, s, x_i) + V_{i+1}(g_i(\mathcal{I}, s, x_i)) \right\}, \qquad i = 1, \ldots, n, \tag{6.9}$$

$$V_{n+1}((\mathcal{I}, s)) = 0, \tag{6.10}$$

where $V_1((\mathcal{V}, \mathbf{0}))$ yields the optimal solution value of the QSSP. In particular, we denote $(\mathcal{V}, \mathbf{0})$ by *root state* of the system.

6.3.2 Constructing the BDD representation for the QSSP

We now describe how to generate the decision diagram representation based on the dynamic program presented above. For notation purposes, let $\mathcal{S}(\mathcal{G})$ be the family of stable sets of $\mathcal{G}$.

A binary decision diagram for a QSSP instance is a layered directed acyclic graph $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N}$ is the node set and $\mathcal{A}$ is the arc set. The node set $\mathcal{N}$ is partitioned into $n+1$ layers $L_1, \ldots, L_{n+1}$, where the first and last layers L_1 and L_{n+1} are singletons containing a *root node* r and a *terminal node* t, respectively. We denote by $l(u)$ the index of the layer of a node $u \in \mathcal{N}$, i.e., $u \in L_{l(u)}$. A BDD arc $a \in \mathcal{A}$ only connects nodes in adjacent layers and is equipped with a binary label $d_a \in \{0, 1\}$ and a profit h_a. We refer to arc a by *1-arc* if $d(a) = 1$ and by *0-arc* otherwise.

A BDD is a compact graphical representation of the state transition graph of the DP (6.9)-(6.10). Specifically, each node in $\mathcal{N}$ represents a state $(\mathcal{I}, s)$ and the layer L_i contains the nodes associated with the states that are reachable at stage i, $i = 1, \ldots, n$. In particular, the root node r is associated with the root state $(\mathcal{V}, \mathbf{0})$. Arcs encode the transition $g_i(\cdot)$, i.e., there exists an arc $a = (u, u') \in \mathcal{A}$ with label d_a iff, given the state $(\mathcal{I}, s)$ associated with u, we have $d_a \in \mathcal{F}(\mathcal{I})$ and the state associated with u' is $g_{l(u)}(\mathcal{I}, s, d_a)$. The profit of such arc is $h_a := h_{l(u)}(\mathcal{I}, s, d_a)$. The terminal node t, in turn, represents all terminal states of (6.9)-(6.10), i.e., they are perceived as merged into a single node.

In such a representation, we have a one-to-one mapping between stable sets in $\mathcal{G}$ and paths of the BDD. Namely, for every arc-specified path $(a_1, a_2, \ldots, a_n)$ starting from the root r and ending at the terminal t, the arc labels yield a feasible assignment $x := (d_{a_1}, \ldots, d_{a_n})$ by validity of the DP. Conversely, every such feasible assignment must be encoded in some path of the BDD. This implies that the QSSP now reduces to finding a longest-path problem over the BDD, where arc lengths are the arc profits h_a.

Figure 6.2 illustrates the exact BDD for the instance in Figure 6.1. States are included on top of each of their corresponding nodes in each layer. Solid and dash arcs represent $d(a) = 1$ and $d(a) = 0$, respectively, and arc profits h_a are included (in blue) on top of each arc.

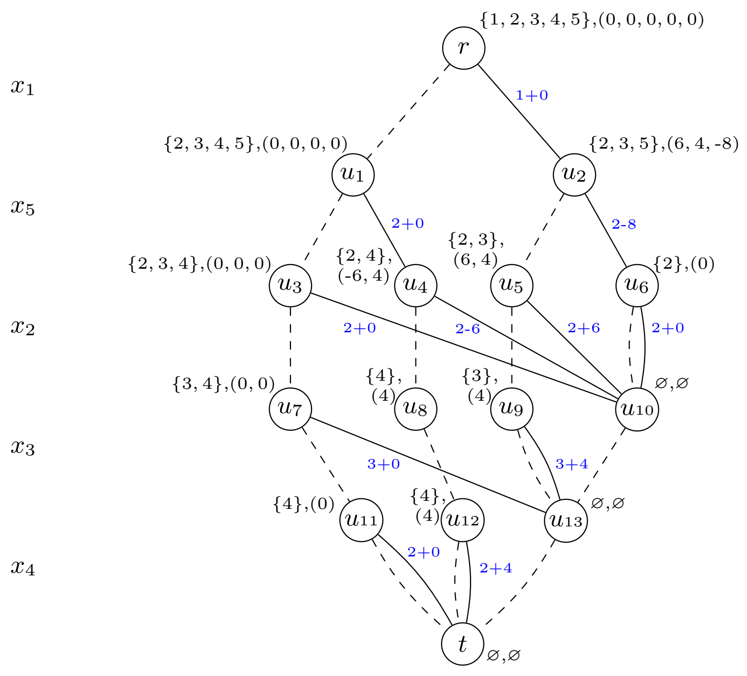


Figure 6.2 Exact BDD for the QSSP instance of Figure 6.1

The variable ordering in the decision diagram of Figure 6.2 is established by x_1, x_5, x_2, x_3, x_4, i.e., these are the variables associated with outgoing arcs from layers L_1,L_2, L_3,L_4,L_5, respectively. To illustrate the BDD compilation, we observe that the root node r has state information $(\{1,2,3,4,5\},(0,0,0,0,0))$. Note that, at the beginning, all vertices are candidate to be added to the stable set and, no matter the vertex associated with outgoing arcs of layer L_1 only the vertex weight w_i is collected. Next, for instance, there is an outgoing 1-arc which represents $x_1 = 1$ and leads to node u_2 with state information $(\mathcal{I}(u_2), s(u_2)) = (\{2,3,5\},(6,4,\text{-}8))$. The latter indicates that by including vertex 1 in the partial stable set, we get to a state where we remove vertex 1 and 4 from the possible vertices, updating states accordingly.

In addition, note that variable x_5 is associated with L_2. Consequently, the weight of the outgoing 1-arc (u_2, u_6) is $2 - 8$, this is, the profit of vertex 5 (i.e., $w_5 = 2$) plus "-8" which is associated with the sum of quadratic contributions for vertex 5 in the state component $s(u_2)$.

In the BDD in Figure 6.2, the longest path is given by the arc-specified path $((r, u_2)$, (u_2, u_5), (u_5, u_{10}), $(u_{10}$, $u_{13})$, $(u_{13}, t))$ indicating that the optimal stable set for the problem is $\{1, 2\}$ with optimal objective value $z^* = 9$.

Compiling BDDs We follow a typical top-down procedure for constructing an exact BDD, which is equivalent to a forward recursion over the DP. The procedure is defined as follows.

Let us denote by $(\mathcal{I}(u), s(u))$ the state associated with the BDD node $u \in \mathcal{N}$. Layers are compiled one at a time in the order $L_1, \ldots, L_{n+1}$. At each iteration $i = 1, \ldots, n$, we calculate all state transitions from the states associated with nodes in L_i to generate nodes in L_{i+1}, adding arcs as necessary. We also ensure that no two nodes have the same state, i.e., either $\mathcal{I}(u) \neq \mathcal{I}(u')$ or $s(u) \neq s(u')$ for any two nodes u, u' at the same layer. Finally, all nodes in the last layer L_{n+1} are merged into a single terminal node t.

6.3.3 Approximate decision diagrams for the QSSP

In general, exact BDDs grow exponentially large on the problem input and are not computationally tractable. Because of this, we instead manipulate the so-called *approximate* versions, i.e., relaxed and restricted decision diagrams. Such diagrams are key as a bounding mechanism since they exploit discrete structure to relax the state space.

A DD is *relaxed* if it over-approximates the solution set of a problem, encoding all feasible solutions but also allowing infeasible ones. Relaxed BDDs are obtained similarly as exact BDDs when using a top-down approach. In particular, if the number of nodes in a layer (i.e., the layer width) exceeds a given pre-specified limit W during its construction, two nonidentical nodes u and u' are heuristically selected and merged into a new node u''. Next, the longest (resp., shortest) path value in a relaxed BDD yields a dual bound for a maximization (resp., minimization) optimization problem. Note that the maximum width W is then a relevant parameter because it allows to trade-off computational effort and bound quality, i.e., the larger the W value, the better the bound that is obtained. More details on the experiments leading to a suitable value of W are reported in Section 6.5.

For the QSSP, we propose a valid merging operator which guarantees that no feasible solution is lost meanwhile keeping the relaxed BDD size under control. The state of the merged node u'' is set as the pair $(\mathcal{I}(u''), s(u''))$ where $\mathcal{I}(u'') := \mathcal{I}(u) \cup \mathcal{I}(u')$ and $s(u'') := \max\{\{s(u)\}_j, \{s(u')\}_j\}_{j \in \mathcal{I}(u) \cup \mathcal{I}(u')}$. The strategy used to define which nodes are merged consists of selecting BDD nodes u and u' with the partial longest path from the root node. Once nodes are merged, all previous incoming arcs to u and u' are directed to the new merged node u''. The proposed merging operator assures that all valid stable sets are preserved and that a longest path computation in the resulting relaxed BDD provides an upper bound on the optimal objective value z^*.

Furthermore, we also manipulate *restricted* decision diagrams to obtain feasible solutions. A restricted BDD under-approximates the solution set of a problem i.e., it only allows feasible solutions but could miss the optimal one. They can be also compiled through a top-down construction. In the restricted-version case, when reaching the maximum width W in a layer, instead of merging nodes, we heuristically select nodes to be removed from such a layer. A longest path computation in this case provides a lower bound on z^*.

6.4 A BDD-based hybrid optimization approach for the QSSP

Given relaxed and restricted BDD representations as well as a BQP formulation of the QSSP (presented in Sections 6.2 and 6.3), we propose to deploy a hybrid BDD-MIP ([80]) algorithm as a solution methodology. The integrated BDD-MIP method leverages two problem representations and BDD-based search mechanisms to exploit complementary strengths coming from the different optimization paradigms. In Section 6.4.1, we initially describe a typical BDD-based exploration of the solution space. Next, we focus on the hybrid algorithm mechanisms considered for the QSSP (Section 6.4.2).

6.4.1 BDD-based search scheme

A key component of the combined framework is the BDD-based exploration in which a relaxed binary decision diagram plays the role of a search tree in a branch-and-bound scheme. In such a search mechanism, the solution space is divided and explored by recursively branching on suitable BDD nodes (i.e., set of partial solutions) instead of branching on variable-value pairs as in traditional linear programming-based branch and bound.

We now describe the general idea of a BDD-based search mechanism in the context of a stand-alone decision diagram approach [17]. Consider a relaxed binary decision diagram $\bar{\mathcal{B}}$ of the QSSP as the example provided in Figure 6.3, for the illustrative instance in Figure 6.1, where the maximum width is set as $W = 2$. For every pair of nodes $u, u' \in \bar{\mathcal{B}}$ such that $l(u) < l(u')$, let $\bar{\mathcal{B}}_{uu'}$ be the binary decision diagram induced by all the nodes and arcs that lie on directed paths from u to u', e.g., $\bar{\mathcal{B}}_{rt} = \bar{\mathcal{B}}$. We say that a node u in $\bar{\mathcal{B}}$ is *exact* if all $r - u$ paths lead to the same state $s(u)$, and is relaxed otherwise. In addition, a *cutset* of $\bar{\mathcal{B}}$ is a subset of nodes C such that any $r - t$ path in $\bar{\mathcal{B}}$ contains at least one node in C. In specific, C defines an exact cutset if all nodes in C are exact. Several strategies have been proposed for obtaining exact cutsets, such as *the frontier cutset (FC)* and *the last exact layer (LEL)* (we refer the reader to [6]).

Figure 6.3 illustrates an exact cutset C defined by the LEL and formed, in this case, by BDD nodes $\bar{u}_1$ and $\bar{u}_2$ (in orange). Note that nodes in blue, $\bar{u}_3$ and $\bar{u}_4$, were forcefully merged to meet W using the merging operator proposed in Section 6.3.3.

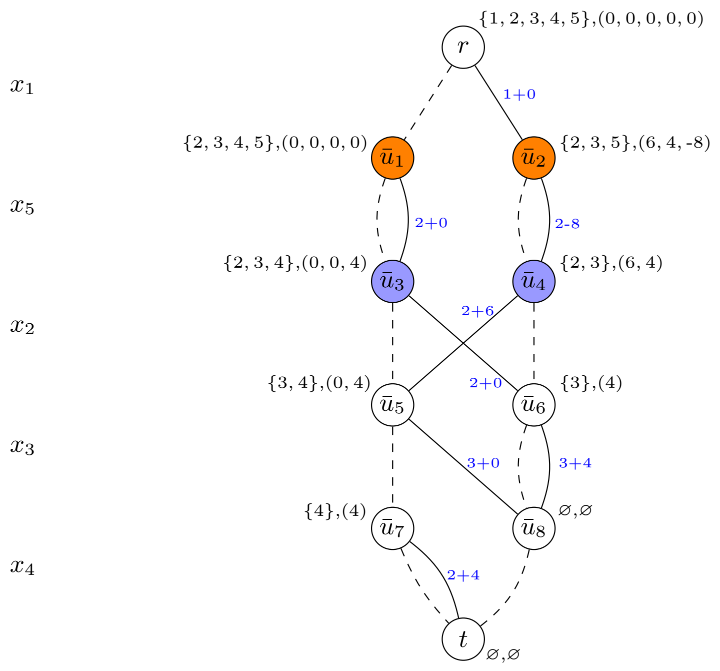


Figure 6.3 Relaxed BDD for the QSSP instance of Figure 6.1

Given an exact cutset C, a BDD-based branch and bound explores each BDD node in C to find and prove the optimal solution. Let $v^*(u)$ be the longest-path value from r to u for each $u \in C$. Let z^*_u be the optimal value of the subproblem for which its solutions are exactly encoded in $\mathcal{B}_{ut}$, therefore $v^*(u) + z^*_u$ is the value of the best solution across all $r - t$ paths that contain BDD node u. Since all $r - t$ paths of a decision diagram must contain some node in C, we search the optimal value z^* of the problem by solving the subproblems associated with $\mathcal{B}_{ut}$. Such subproblems (every $u \in C$) are solved separately and each subproblem then leads to a smaller and hence more tractable binary decision diagram. This procedure can be applied recursively for each u if the relaxed BDD rooted in u is either not exact (i.e., if the maximum width is reached while its construction) or cannot be implicitly pruned.

As an illustration of the BDD-based exploration scheme in Figure 6.3, we have to explore BDD nodes $\bar{u}_1$ and $\bar{u}_2$ further to continue searching for the optimal solution. In a stand-alone BDD-based search, such an exploration implies the compilation of two different relaxed decision diagrams, each rooted in $\bar{u}_1$ and $\bar{u}_2$, respectively.

6.4.2 Hybrid BDD-MIP mechanisms

In the integrated approach, MIP technology is incorporated into the BDD-based exploration scheme. Namely, a MIP solver can directly prune BDD nodes by solving them up to optimality while also providing lower bounds. Meanwhile, the incumbent solution found so far in the BDD-based branch and bound can be used to define a lower cutoff in the objective function of the subproblems explored by the MIP solver.

MIP-based pruning strategy

We specifically propose to adapt the *IP-based pruning* strategy in [80]. Consider a BDD-based branch and bound and a relaxed BDD which has to be explored further. Once a cutset C is defined and selected, a node $u \in C$ can be directly explored and pruned in advance by finding its optimal solution z^*_u. Along these lines, the strategy consists of solving the subproblem associated with u using a MIP solver as opposed to recursively relax and explore $\mathcal{B}_{ut}$ through BDD technology. In addition, once the subproblem associated with node u is solved, $v^*(u) + z^*_u$ also establishes a lower bound on z^*.

For the QSSP, a subproblem encoded in a BDD node u corresponds to a vertex-induced subgraph defined by the vertices considered in the state component $\mathcal{I}(u)$. Note that such a subproblem also leads to a BQP model if it is tackled with MIP technology. Thus, the mechanisms within the hybrid BDD-MIP algorithm leads to an algorithm-selection decision during the exploration. There are three important aspects to be considered, (i) the subproblems (i.e., BDD nodes to be explored) are dynamically generated during search, (ii) solving a BQP subproblem up to optimality may be computationally too expensive, and (iii) such an algorithm-selection decision is made based on the subproblem features encoded by the state $(\mathcal{I}(u), s(u))$. Determining whether to apply the MIP-based pruning at each BDD node plays a central role in the hybrid approach, and we cast this algorithmic question as a classification task.

In [80], when solving the maximum stable set problem, machine learning (ML) is used to derive trained classifiers and a decision tree to determine when to use the MIP-based pruning strategy. At the root node, a classifier automatically detects whether the related subproblem (i.e., the instance itself) is simply solved by MIP technology. Then, at the remaining BDD nodes, a computationally cheap decision tree determines if each BDD node is either pruned using a MIP solver or the BDD continues the exploration further. In this paper, we take a further step and propose using ML to guide the exploration within the hybrid BDD-MIP algorithm but at every BDD node subproblem.

ML-driven exploration

We rely on the recent connection between ML and discrete optimization (see, e.g., [50] for a survey). Specifically, we cast the algorithm-selection decision within the hybrid BDD-MIP algorithm as a classification task which is addressed by ML. We employ traditional supervised ML techniques and learn a classifier to decide which technology should be used (i.e., a BDD relaxation or a MIP representation) to explore, on-the-fly, a BDD subproblem.

In a supervised classification problem, the objective is to learn a function which assigns a discrete class label to an unseen instance, given a set of already classified examples (i.e., the training data). Each example in the training set is described by a set of features (attributes) and an associated label. The learned function (classifier) then maps the features to the available classes revealing a possible hidden structure of the training dataset. In our case, an example corresponds to a QSSP instance and we define two class labels that represent the BDD and MIP technology.

We perform the learning experiments offline to train the classifier which is later incorporated in the hybrid BDD-MIP algorithm. Thus, we proceed to describe the supervised learning methodology and the experiments to learn such a classifier. The methodology comprises five main steps, (i) the generation of instances, (ii) the feature design, (iii) the label definition for the learning task, (iv) the dataset composition, and finally, (v) the learning experiments.

Instances generation As previously mentioned, a QSSP instance mainly corresponds to both an undirected graph and a symmetric matrix of quadratic profits. We randomly produce examples for the training dataset by generating graphs following the Erdös-Rényi (ER) [69] model where we vary the number of nodes (n) and graph density (p). Moreover, for each instance, we generate a symmetric matrix Q where we define its percentage of positive coefficients (v).

Feature design Since we label an instance between two different problem representations (i.e., BDD and MIP), the prediction should be a function of only problem-specific features. For the QSSP, we rely on the graph properties as well as attributes associated with the symmetric matrix Q. As the trained classifier is a component of the BDD-MIP algorithm and it is potentially invoked several times during search, we target features that can be efficiently computed for new instances that will be dynamically generated when exploring the solution space. For the graph properties, we select 10 features, namely, number of nodes (n), number of edges ($|\mathcal{E}|$), density (p), and seven features derived from node degrees, specifically, their mean, median, standard deviation (SD), maximum, minimum, the interquartile range (IR), and the variability score (SD/mean).

Conversely, for the features associated with Q, we select the percentage of positive coefficients in Q (v) and also characteristics associated with its main diagonal which is related to the linear profits w_i. For features coming from w_i, we define the mean, median, standard deviation (SD), minimum and maximum. Finally, we use a total of 16 features for the learning experiments.

Label definition Since we cast the algorithmic question as a binary classification problem, we now define a procedure to binarize the label, which is a function on the performance of the two optimization technologies. Each QSSP instance is solved with both the MIP and a stand-alone BDD solver based on the representation proposed in this paper. For MIP, each QSSP example is solved with CPLEX version `12.8` (5 times with a different random seed to deal with performance variability issues [68]) using its nonlinear programming-based branch and bound to get the MIP solving time MIP_{time} as the average of the five runs. Next, each instance is also solved with the stand-alone BDD solver (by setting $W = 128$ for all instances) to obtain the BDD_{time}. Finally, for each example, we assign the label `MIP` if the MIP solver is α times faster than the BDD solver (i.e., if $MIP_{time} \cdot \alpha \leq BDD_{time}$), otherwise we assign the label `BDD`.

Dataset composition We assess the distribution of solving times and binarization parameter α to generate a dataset that is meaningful for the learning task. The dataset is composed of 12,500 QSSP instances where $n \in \{20, 30, 40, 50, 60, 70, 80, 90, 100, 120\}$, $p \in \{10, 15, 20, 25, 30, 40, 45, 50, 60, 65, 70, 75\}$, $v \in \{25, 50, 75\}$, and label binarization parameter $\alpha = 5$. Finally, the number (resp., percentage) of instances labeled as `MIP` and `BDD` in the dataset is 9110 (72.88%) and 3390 (27.12%), respectively.

Supervised learning experiments We construct the classifier using a Support Vector Machine (SVM) with RBF kernel [25], a classical supervised learning algorithm. We randomly split the dataset into training (75%) and test set (25%). To obtain features in the same range, we apply feature scaling and mean normalization so, each feature is normalized to have a mean of 0 and a standard deviation of 1. Each experiment consists of a training phase with 5-fold cross validation and grid search method for hyperparameter tuning, as well as a test phase on the neutral test set. We compare the SVM versus a dummy classifier (DUM), which follows a stratified strategy, i.e., it makes predictions based on the class distribution of the training set. The benchmark with DUM is a good practice as a measure of baseline performance for the trained classifier.

The learning methodology is implemented using Python with Scikit-learn [76]. Table 6.1 presents the standard performance measures of binary classification, namely, accuracy, precision, recall, and f1-score.

We highlight that the SVM classifier achieves high performance metrics. It compares favorably versus a dummy classifier corroborating that there is a statistical pattern to be learned when discriminating either a MIP or BDD solver for tackling a QSSP instance. We remark that, for this particular application, false positives are computationally expensive, even more once a classifier is taken to production within the hybrid BDD-MIP. In this case, metrics different than accuracy could provide a better insight on the classifier performance. Precision, which is defined as the true positives divided by all positive predictions can be a good metric to analyze. A high precision indicates a low number of false positives, and we observe that the classifier presents a very good precision.

Table 6.1 Performance measures for the classifiers when predicting `MIP`/`BDD`

	DUM	SVM
Accuracy	0.607	0.976
Precision	0.257	0.953
Recall	0.289	0.954
F1-score	0.272	0.953

The classifier is able to capture enough about the discrimination from the selected features on training set to make meaningful predictions on test set. We conclude that the learning experiments support the selection and inclusion of the trained classifier within the BDD-MIP optimization algorithm.

BDD-MIP cutoff

Moreover, we incorporate another strategy used in [80]. We let LB denote a global incumbent solution obtained from the hybrid BDD-MIP exploration. Next, when each subproblem (associated with BDD node u) is explored through a MIP solver, i.e., the trained classifier predicts `MIP`, the corresponding BQP subproblem is modified by including the following constraint:

$$\sum_{i\in\mathcal{I}(u)} w_i x_i + \sum_{i\in\mathcal{I}(u)} \sum_{j\in\mathcal{I}(u)|i\neq j} 2q_{ij}x_ix_j \geq LB - v^*(u). \tag{6.11}$$

Constraint (6.11) establishes a lower bound on the objective function for the subproblem, considering the global incumbent solution and the longest-path value from r to u. The lower-cutoff procedure already proved effective in [80] to speedup a hybrid BDD-MIP algorithm. In such a case, subproblems terminate significantly earlier the solving procedure, sometimes with the proof that no feasible solution meets the modified BQP conditions, also leading to prune the BDD node.

Algorithm 2 describes the hybrid BDD-MIP algorithm for the QSSP. At the beginning, the list of nodes to be explored consists of only the root node r. In line 2, while either L is not empty or the best bound is not less than or equal to the best incumbent solution we have to search for the optimal solution value. We take a BDD node u from L and evaluate in line 4, if the MIP-based pruning strategy should be applied to by calling, on-the-fly, the trained `MIP/BDD` SVM classifier described in Section 6.4.2. If the classifier predicts `MIP`, we use the MIP representation of the BDD subproblem and a MIP solver to prune the node in advance, updating the best incumbent if necessary. Otherwise, in line 9, we create a relaxed decision diagram rooted in u.

Next, if the BDD is exact we have found a feasible solution, update the incumbent solution if necessary and no further exploration is needed from the generated BDD. On the contrary (line 15), if the BDD is relaxed because the maximum width was reached when constructing any layer, we check if the node can be pruned by bound. If the best bound yielded by the relaxed BDD is greater than the incumbent solution, we must explore further by identifying an exact cutset C, and including the BDD nodes in C to L. The algorithm keeps exploring the solution space until the stopping criterion is met and the optimal solution is provided.

Algorithm 2 Hybrid BDD-MIP solver for the QSSP

```
    Input: QSSP instance
    Output: Optimal value
 1: Initialize list of nodes to be explored ($L$) with BDD root node
 2: while stopping criteria not met do
 3:     take node $u$ from $L$
 4:     if MIP-based pruning strategy applied to $u$ then
 5:         if best incumbent found then
 6:             update incumbent
 7:         Prune node $u$
 8:     create relaxed BDD rooted in $u$
 9:     if BDD is exact then
10:         if best incumbent found then
11:             update incumbent
12:         Prune node $u$
13:     if BDD is not exact then
14:         if Best bound greater than incumbent then
15:             Identify an exact cutset $C$
16:             for all nodes in $C$ do:
17:                 Add node to $L$
18:         create restricted BDD rooted in $u$
19:         if best incumbent found then
20:             update incumbent
21: return optimal value
```

Note that depending on the subproblem properties and hence the `MIP/BDD` classifier prediction, it may occur that no complementarity is identified by the algorithm, i.e., the MIP solver is never called to prune a BDD node (step 4 of Algorithm 2). In such a case, the behavior of the hybrid algorithm actually reduces to a stand-alone decision diagram solving procedure.

6.5 Computational experiments on the QSSP

In this section, we benchmark the hybrid BDD-MIP algorithm against other general-purpose solvers coming from different optimization paradigms. In particular, we compare the hybrid approach versus two leading commercial MIP solvers with QP capabilities and a competitive SDP-based solver. We use `IBM-CPLEX 12.8` and `Gurobi 9.0.0` as the MIP solvers and we refer to them as CPLEX and Gurobi, respectively. We also implement the hybrid BDD-MIP approach in C++ and solve the MIP subproblems with CPLEX. All experiments are run on a Linux machine, Intel(R) Xeon(R) Gold 6142 CPU @ 2.60GHz and 512 GB of RAM.

As it is described in [58], CPLEX can solve a BQP, such as the QSSP, in different ways. Let us denote the QSSP relaxation as the continuous problem where the integrality constraints (6.3), in Section 6.2, are relaxed. The semi-definiteness of matrix Q determines whether the QSSP relaxation is convex and therefore, the way CPLEX can tackle the problem.

In case the QSSP relaxation is convex, i.e., Q is positive semi-definite ($Q \succeq 0$), the problem can be solved by NLP-based branch and bound where a BQP relaxation is solved at each node of the search tree. Also in the convex case, CPLEX can linearize the BQP by transforming it into a MIP model by means of the McCormick inequalities and tackle the resulting formulation with standard MIP techniques. On the other hand, if the problem is not convex ($Q \not\succeq 0$), CPLEX has two alternatives. First, the problem can be convexified through the augmentation of the main diagonal of Q and then be solved by NLP-based branch and bound. Finally, CPLEX can also linearize the BQP and tackle the resulting larger MILP model via a traditional branch and bound. In the numerical experiments, we solve the QSSP using both alternatives i.e., linearizing and not linearizing the QSSP which can be simply selected through the CPLEX parameter `QToLin`[2]. The linearization mode allows us to evaluate the performance of a generic linearization technique (aiming at considering this approach as proposed in [79] where different linearization techniques are used to solve the QSSP). In addition, we tackle the instances using the CPLEX's NLP-based branch and bound algorithm whose performance has not been evaluated for the QSSP in the literature. For both modes, all internal CPLEX's capabilities (i.e., presolving, heuristics and cuts) and remaining default parameter settings are enabled.

We also solve the QSSP instances using Gurobi as MIP solver. The Gurobi version used in this experiments, like CPLEX, allows to solve to global optimality both convex and non-convex binary quadratic programmming models. As reported in its technical documentation, Gurobi implements a MIP solver where simplex and barrier algorithms tackle continuous BQPs. In addition, Gurobi's presolve may either convexify a problem by using bilinear constraints or linearize the problem so it can be solved by standard MIP techniques. Gurobi is also used with the default parameter settings and all internal capabilities (i.e., presolve, heuristics and cuts) enabled to make the comparison even more sound.

[2]Recently, CPLEX version `12.10` incorporates a ML algorithm to make this decision.

In addition to MIP solvers, BiqCrunch [89] becomes an interesting and natural alternative to be considered. This SDP-based branch and bound has also been used for the QSSP [95]. BiqCrunch is executed in the generic problem setup enabling internal heuristics 1, 2, and 3.

Regarding the Hybrid BDD-MIP solver, we use the MIP-based pruning strategy. The trained SVM classifier is invoked to label each BDD node subproblem as either `MIP` or `BDD` to define if the MIP-based pruning is performed. In addition, we implement the lower cutoff procedure and when the MIP solver is called to prune a BDD node, we solve the equivalent BQP subproblem using the clique formulation and CPLEX as MIP solver without any time limit. The variable ordering and exact cut selection strategies used are *the minimum number of states (MIN)* and *the last exact layer (LEL)*, respectively. We refer the reader to [6] for details on different variable ordering heuristics and exact cutset strategies.

As mentioned before, the decision diagram width (W) is a critical parameter when manipulating limited-size DDs. Moreover, the mechanisms within the hybrid BDD-MIP approach generate an interesting dynamic between the DD-based branching scheme, the width, the bound quality, the size of the exact cutsets, the MIP-based pruning strategy, and the classifier predictions guiding the exploration. For instance, a very small width can dramatically deteriorate the bound quality, lead to a greater number of DD nodes to explore, and increase the computing time of the whole optimization process, up to one order of magnitude slower in the denser instances. Conversely, a very large width can improve the bounds and lead to a smaller number of DD nodes to explore. However, it can also trigger a greater computational effort when compiling each relaxed DD in the procedure which might not payoff when observing the whole computing time. Nevertheless, the effect of selecting a wrong value of W could be mitigated and indeed exploited in the hybrid BDD-MIP algorithm by calling more or less frequently the MIP-based pruning strategy. After exploring and exploiting this trade-off, the maximum width W for all instances is set to 64.

The testbed presented in these computational experiments corresponds to the dense instances used in the computational experiments in [79] plus, a set of sparser (and hence harder) instances which are also considered in [95]. The set of instances has number of nodes $n = \{100, 150\}$, density $p = \{25, 50, 75\}\%$, and percentage of positive coefficients in Q, $v = \{25, 50, 75\}\%$. The testbed considers 3 instances per combination (n, p, v) for a total of 54 instances. Every instance is processed five times by solving it with: BiqCrunch, CPLEX for the clique formulation (linearizing and not linearizing the BQP), Gurobi for the clique formulation, and finally, the proposed hybrid BDD-MIP solver. Each solver run uses only one thread with a time limit of $7,200$ seconds.

Table 6.2 compares the performance of the different solvers for the testbed instances. We present the average performance by combination (n, p, v) for a total of 18 different group of instances. Column 1 corresponds to the group id for the three instances of each combination (n, p, v). Columns 2, 3, and 4 present, for each group, the number of nodes, density, and percentage of positive coefficients in Q. Each solver is represented by a column where the base number corresponds to the average computational time employed for solving the 3 instances of the corresponding group. An exponent, in case it appears, indicates how many of those instances could not be solved to optimality within the time limit. In this way, column 5 is associated with BiqCrunch. For such an SDP-based solver, we generated the model from both MIP formulations, the edge and clique models. The edge formulation presents slightly better results and it is the one reported in these experiments. Columns 6 and 7 correspond to the performance of CPLEX solving the clique formulation of the problem both linearizing and not linearizing the QSSP, respectively. Next, column 8 corresponds to Gurobi's performance. For the MIP solvers, we use the clique formulation which reported 4% better results than the edge formulation. Column 9 relates to the performance of the hybrid BDD-MIP solver. Finally, column 10 shows the average speedup reached by the hybrid BDD-MIP solver with respect to the most competitive benchmark for each instance of the group. For example, the value in the seventh column "$6555.89^{(2)}$" of group #10 indicates that, within the time limit, CPLEX (when non linearizing the BQP) solved only 1 of the instances of the group $n = 150$, $p = 25$, $v = 25$, and the average solving time of the 3 instances is 6555.89 seconds. We present the detailed computational experiments for each instance in Appendix A.

We can observe from Table 6.2 that the hybrid solver outperforms all the benchmarks. The hardest instances are the sparsest ones ($p = 25$) where the speedups obtained with the hybrid solver are the smallest ones but greater than 1x. In addition, for group 12 ($n = 150 - p = 25 - v = 75$), the hybrid solver is the only one able to solve the 3 instances to optimality. On the other hand, when the instance is denser ($p = \{50, 75\}$) the BDD-based approach achieves remarkable speedups of at least one order of magnitude faster than the MIP solvers and the SDP-based solver.

Note that the percentage of positive coefficients v is related to the definiteness of matrix Q and evidently seems to be a very important feature of the instance. Let us observe instances with $n = 150$ and $p = 50$ (i.e., groups 13, 14, and 15 in Table 6.2) to analyze this behavior. In such instances, if we observe the performance of the hybrid algorithm, no matter the value of v, the solving time is almost equivalent for the three groups. However, this is not the case for CPLEX and Gurobi where the v value greatly impacts how the model is solved and hence the resulting performance. Noteworthy, the proposed BDD representation for the QSSP embeds the problem nonlinearity making no difference in the performance with respect to the semi-definiteness of matrix Q.

Table 6.2 Comparison between the hybrid BDD-MIP algorithm and other optimization paradigms for the QSSP

Instances				**BiqCrunch**	**CPLEX**		**Gurobi**	**Hybrid BDD-MIP**	**Speedup**
					Linearize	Non Linearize			
Group	n	p	v	CPU time (s)	CPU time (s)	CPU time (s)	CPU time (s)	CPU time (s)	vs. best benchmark
1	100	25	25	1139.31	272.65	113.13	84.75	24.50	3.50
2	100	25	50	7200.00$^{(3)}$	728.37	138.56	287.55	36.16	4.03
3	100	25	75	7200.00$^{(3)}$	1009.52	1388.85	184.00	167.02	1.10
4	100	50	25	829.49	18.28	9.67	11.09	0.72	13.53
5	100	50	50	4832.11	54.55	13.50	11.63	0.83	13.94
6	100	50	75	6677.62	89.59	30.51	14.37	0.74	19.50
7	100	75	25	461.94	3.89	4.18	1.03	0.06	18.34
8	100	75	50	1356.93	17.33	4.26	1.57	0.06	27.86
9	100	75	75	1162.46	30.49	6.12	1.71	0.05	32.28
10	150	25	25	7200.00$^{(3)}$	4776.18	6555.89$^{(2)}$	1607.86	560.80	2.87
11	150	25	50	7200.00$^{(3)}$	7200.00$^{(3)}$	7200.00$^{(3)}$	6918.76$^{(2)}$	888.37	7.92
12	150	25	75	7200.00$^{(3)}$	7200.00$^{(3)}$	7200.00$^{(3)}$	7200.00$^{(3)}$	2958.74	2.47
13	150	50	25	6901.33$^{(2)}$	130.06	135.64	46.01	7.45	6.19
14	150	50	50	7200.00$^{(3)}$	1184.23	182.24	96.15	7.39	13.03
15	150	50	75	7200.00$^{(3)}$	1752.94	582.37	91.46	6.83	13.38
16	150	75	25	2895.55	40.35	23.09	16.18	0.21	54.63
17	150	75	50	7160.97$^{(2)}$	79.11	26.49	43.05	0.19	139.37
18	150	75	75	7200.00$^{(3)}$	87.21	38.58	48.94	0.21	179.39
	Geom. Mean			3763.68	210.96	100.49	55.55	3.95	12.74

Figure 6.4 presents a performance profile to benchmark each solver in terms of the percentage of instances solved (out of the total 54 instances) within the execution time which ends at the time limit of $7,200$ seconds. Each of the five lines corresponds to one of the solvers considered in the comparison.

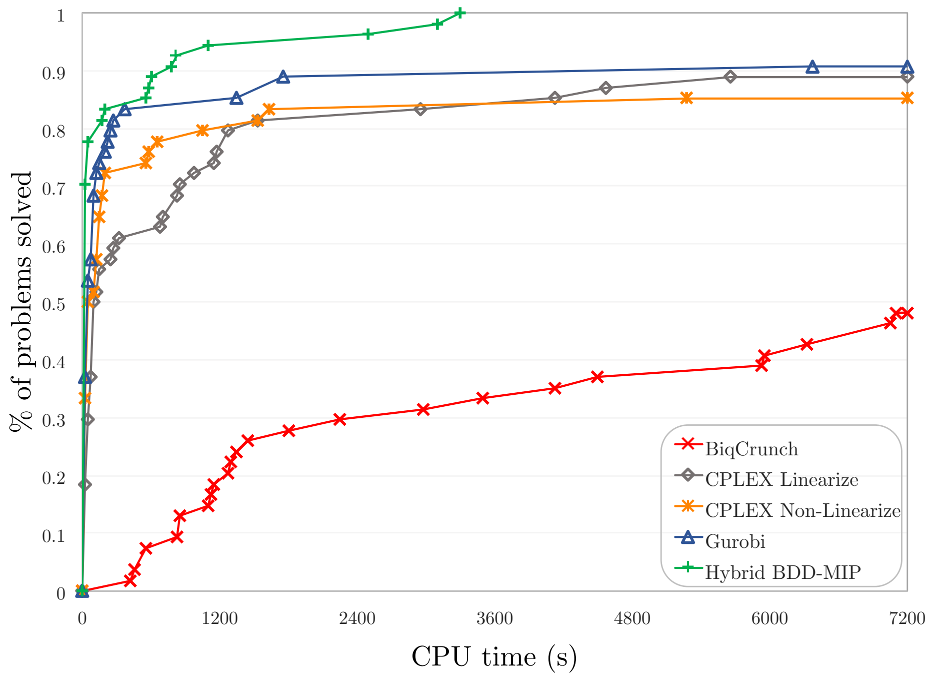


Figure 6.4 Performance profile for different solvers when tackling the QSSP

The performance profile shows the considerable dominance of the hybrid BDD-MIP solver with respect to the other solvers. The hybrid approach is the only method able to solve all instances long before the time limit is reached, solving each of the 54 instances within $3,300$ seconds.

Figure 6.5 compares the solution time of the hybrid BDD-MIP solver versus the solution time of the best benchmark across the different solvers (BiqCrunch, CPLEX linearizing, CPLEX non-linearizing, and Gurobi). We color instances by density value so that, green, red, and gray points correspond to $p = 25$, $p = 50$, and $p = 75$ instances, respectively. In a similar way, shapes are associated with the problem size n. Triangle and circle markers are related to $n = 100$ and $n = 150$ instances, respectively. A point located above the diagonal means that the hybrid BDD-MIP approach outperforms the best benchmark in such an instance for the corresponding $n - p$ group. As we use log scale to be able to represent all instances in one chart, some group of instances are represented by superimposed points, e.g., instances with $n = 150 - p = 50$ (red circles) appear represented by one red point to the most left side.

Such a point indicates that the hybrid performs much better than the best benchmark.

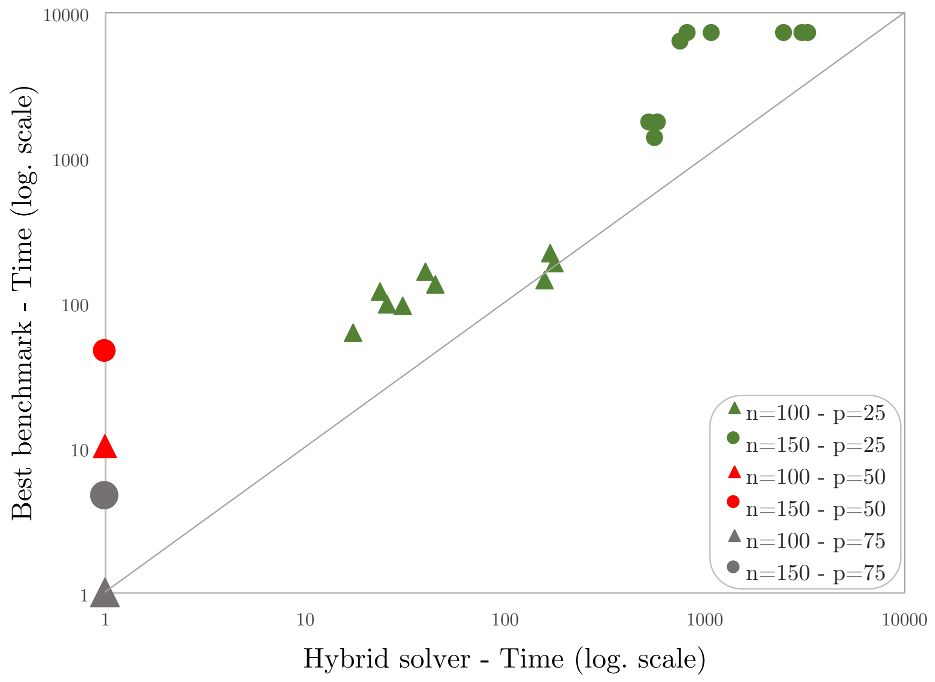
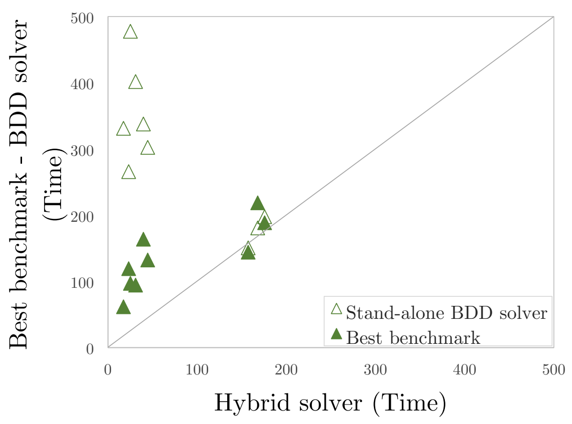


Figure 6.5 Hybrid BDD-MIP solver versus the best benchmark in terms of solution time per instance in log scale

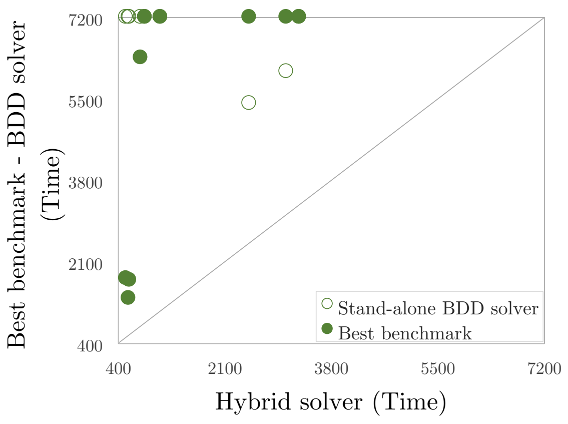
Nevertheless, Figure 6.5 allows us to focus on the hardest instances, i.e., the sparsest ones with $p = 25$ (green markers). All but one instance are above the diagonal indicating that even for the hardest cases, the hybrid algorithm achieves a better performance than the benchmarks. For a fix density value, circle markers appear above triangle markers indicating in that case that larger instances are naturally harder.

Although, the BDD representation is one of the contributions in this paper, we evaluate the global effectiveness of the hybrid framework in comparison with a stand-alone BDD solver. We continue focused on the hardest (i.e., sparsest) subset of 18 instances with $p = 25$. Figure 6.6(a) and Figure 6.6(b) compare the solution time of the hybrid method versus a pure BDD solver but considering as well the best benchmark. As we analyze instances of the same density, solving times have the same magnitude and we generate the scatter plot not using the log scale. Figure 6.6(a) and Figure 6.6(b) are related to $n = 100$ and $n = 150$ instances, respectively. In both cases, instances related to the best benchmark are associated with filled markers, while those related to the stand-alone BDD solver correspond to markers with no fill.

We can observe that the hybrid method consistently outperforms the stand-alone BDD solver.



(a) Instances $n = 100 - p = 25$



(b) Instances $n = 150 - p = 25$

Figure 6.6 Hybrid BDD-MIP solver versus both stand-alone BDD solver and the best benchmark for the sparsest instances

This is even more evident for large instances ($n = 150$) where the hybrid method really pays off. For some instances, we go from hitting the time limit with the pure BDD solver, to getting the best performance using the hybrid algorithm.

In a last computational experiment, we evaluate the performance of the BDD-based optimization approach on larger QSSP instances. We compare Gurobi, which is the best benchmark in the set of instances from the literature (Table 6.2), versus the hybrid BDD-MIP. The new set of instances has number of nodes $n = \{175, 200\}$, density $p = \{25, 50, 75\}\%$, and percentage of positive coefficients in Q, $v = \{25, 50, 75\}\%$. Similarly to the previous dataset, we consider 3 instances per combination (group) (n, p, v) for a total of 54 instances and 18 groups. Table 6.3 shows the computational experiments. The first four columns correspond to the instance information for each group. Column 5 and 6 report the performance of Gurobi and the hybrid BDD-MIP, respectively. Once again, if an exponent appears, it indicates how many of those instances were not solved to optimality within the time limit. Finally, column 7 presents the speedup by group. We present the detailed computational experiments for each instance in Appendix A.

For experiments reported in Table 6.3, the hybrid BDD-MIP uses the same trained classifier and maximum width W than in the previous experiment. Gurobi is also employed with the default parameter settings enabling all its internal capabilities to make the comparison more stringent. For $n = 175$, we note that Gurobi can solve only 1 out of 9 of the sparsest ($p = 25$) instances within time limit, whereas the Hybrid approach struggles with 2 (i.e., it solves 7 out of 9). When we increase the number of vertices to 200, we observe that neither Gurobi nor the Hybrid BDD-MIP can solve the challenging sparse instances. However, the lower and

Table 6.3 Comparison between the hybrid BDD-MIP and Gurobi for QSSP instances with $n \in \{175, 200\}$

	Instances			**Gurobi**	**Hybrid BDD-MIP**	**Speedup**
Group	n	p	v	CPU time (s)	CPU time (s)	
19	175	25	25	$6821.25^{(2)}$	4132.60	1.70
20	175	25	50	$7200.00^{(3)}$	4686.14	1.55
21	175	25	75	$7200.00^{(3)}$	$6733.51^{(2)}$	1.08
22	175	50	25	117.35	14.97	7.89
23	175	50	50	223.18	24.01	9.31
24	175	50	75	260.92	19.90	13.14
25	175	75	25	58.07	0.35	166.56
26	175	75	50	75.74	0.35	216.22
27	175	75	75	83.60	0.37	222.73
28	200	25	25	$7200.00^{(3)}$	$7200.00^{(3)}$	1.00
29	200	25	50	$7200.00^{(3)}$	$7200.00^{(3)}$	1.00
30	200	25	75	$7200.00^{(3)}$	$7200.00^{(3)}$	1.00
31	200	50	25	263.15	37.91	6.92
32	200	50	50	576.81	54.58	10.39
33	200	50	75	504.95	44.11	11.65
34	200	75	25	107.89	0.53	203.68
35	200	75	50	128.78	0.53	245.02
36	200	75	75	161.21	0.56	290.24
	Geom. Mean			581.01	42.74	13.64

upper bounds obtained within time limit are, in general, better for the Hybrid BDD-MIP. Gurobi obtained slightly better lower bounds for only 3 out of the 81 instances, and this corresponds to the group $(200, 25, 25)$. Then, as the percentage of positive coefficients in Q increases, the bounds are more favorable for the Hybrid approach. For denser instances, no matter the size, the speedups show that BDD-based technology is significantly superior. It reaches, in some cases, more than 2 orders of magnitude faster computing times than a state-of-the-art commercial MIP solver. Instances of size 200 seem to be the current limit the Hybrid algorithm can reach within a 2-hour time limit on sparser instances. Instead, the method scales very well for instances of size 250 (and bigger) on denser graphs (see Table A.6 in the appendix).

Different elements of the hybrid BDD-MIP solver, such as the BDD-based branching scheme and, mainly, the black-box classifier guiding the exploration, make getting insights from the method's performance and evolution a complex task. However, we observe that the mechanisms of the hybrid BDD-MIP are mainly exploited for the sparsest (hardest) instances. On average, for instances with $p = 25$, the MIP solver prunes 91.56% of the total number of BDD nodes explored. In such a case, the MIP solver is frequently invoked indicating that the classifier possibly identifies a special pattern in sparse instances that leads to the complementarity leveraged by the hybrid method. As the graph density increases, the MIP calls drastically decrease to the extent that the MIP-based pruning strategy is not employed in very dense instances. We can infer that calling the MIP solver for such instances could be a computationally expensive strategy. Nevertheless, the ML-based exploration seems to detect such a pattern.

6.6 Conclusions

We solved the quadratic stable set problem (QSSP) via BDD-based optimization contributing with both a BDD representation and an adapted hybrid BDD-MIP solver for the problem. We performed extensive computational experiments to compare the proposed hybrid method with other general optimization paradigms such as a semidefinite-based solver and two leading commercial MIP solvers with QP capabilities. We have shown that the hybrid BDD-MIP provides state-of-the-art results for solving the QSSP.

In addition, the BDD-based optimization technology shows high flexibility to represent quadratic problems. One important distinction of the proposed BDD representation is that it does not assume any special structure for the quadratic cost matrix which is an usual assumption in quadratic programming. Indeed, the proposed modeling to handle the quadratic profits in the state variables could be extended to any binary quadratic programming model where the Markov property holds in the conceptual DP model used to compile the decision diagram. We also contribute with a machine learning application to cast an algorithmic question within the hybrid BDD-MIP into a classification task. In an offline fashion, we train a classifier that is invoked, on-the-fly, by the hybrid algorithm to guide the exploration.

Therefore, when tackling an optimization problem, if both a representation from a different paradigm such as MIP or SDP and a mechanism to identify complementarity are available, a BDD-based hybrid approach could stand up as an effective solving method. This work opens up more research avenues where DD-based approaches, integrated methods, and machine learning are considered to tackle other quadratic combinatorial optimization problems.

CHAPTER 7 GENERAL DISCUSSION

Hybrid optimization that combines complementary strengths from different paradigms into unified frameworks can be shown superior to independent solving technologies. Nevertheless, to achieve the highest performance, the designed mechanisms must be advantageous enough that the computational cost of handling different problem representations actually pays off.

Figure 7.1 presents an illustrative summary of the main elements of this dissertation, the main chapter associated with each contribution, and how all contributions are related between them. Contribution I, II, and III correspond to Chapters 4, 5, and 6, respectively. The different contributions are closely interconnected and allow a unified view of the subject of this dissertation: machine learning driven hybrid optimization based on decision diagrams.

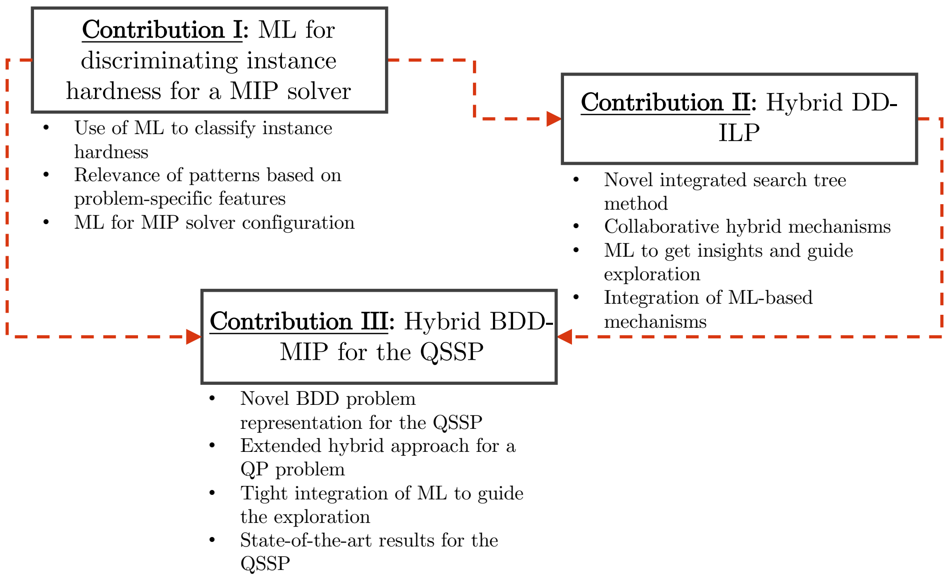

Contribution I: ML for discriminating instance hardness for a MIP solver
• Use of ML to classify instance hardness
• Relevance of patterns based on problem-specific features
• ML for MIP solver configuration
Contribution II: Hybrid DD-ILP
• Novel integrated search tree method
• Collaborative hybrid mechanisms
• ML to get insights and guide exploration
• Integration of ML-based mechanisms
Contribution III: Hybrid BDD-MIP for the QSSP
• Novel BDD problem representation for the QSSP
• Extended hybrid approach for a QP problem
• Tight integration of ML to guide the exploration
• State-of-the-art results for the QSSP

Figure 7.1 Illustrative scheme of contributions in this dissertation and their relation.

The tight integration of ML within optimization solvers is a key element of the hybrid approach presented in Chapters 5 and 6. Along these lines, Chapter 4 is fundamental to explore how machine learning can address optimization algorithmic questions as learning tasks. We present a supervised learning framework which is flexible enough to be adapted to different purposes such as algorithm configuration in a MIP solver and algorithm-design decisions in hybrid optimization. In addition, the importance and usage of problem-specific features within ML approaches for OR is presented in all given contributions.

The main hybrid optimization method and mechanisms, which integrate approximate DDs and MIP, are presented in Chapter 5 and validated through a linear combinatorial problem. Then, such approach is adapted in Chapter 6 where its generic nature is validated in a more difficult and nonlinear problem. The only requirements of the hybrid approach are both a DD and MIP representations along a systematic mechanism to identify and exploit complementary strengths which in this case corresponds to ML.

The hybrid DD-MIP framework emphasizes the role of limited-size DDs as approximations of the branch-and-bound search tree of a problem. In particular, the approach perceives the nodes of a relaxed DD as subproblems that may involve some problem structure that is more suitable to another technology (in our case, MIP), which is invoked to prune the node in advance. The approach turns into an integrated search tree that combine information from DD-like search trees and a more traditional lp-based branch and bound. This integration triggers additional pruning and bounding strategies for DD-based technology. In such a way, even the LP relaxation can provide information to prune unpromising subtrees in the DD representation. Conversely, information from the partial exploration can improve the MIP search such as the lower cutoff.

A relevant difference of the hybrid mechanisms is that the node subproblems are dynamically generated as opposed to an a-priori decomposition which is typical in the literature of integrated methods. We then reveal an algorithm-selection decision that must be made, on-the-fly, to explore the solution space. Therefore, the role of ML stands as a significant component in the approach. We show that the interpretability provided by some ML-models, and studied in Chapter 4, leads to the selection of relevant features which are the basis of exploration mechanisms for the hybrid approach in Chapter 5 and 6.

As an important contribution in Chapter 6, we prove the generic nature of the hybrid approach (proposed in Chapter 5) by adapting it for a harder nonlinear problem. For this purpose, a novel DD representation for the nonlinear problem is presented to exploit the modeling benefits of DDs. In addition, we consider the quadratic programming capabilities of MIP solvers to completely adapt the hybrid approach for the problem. Moreover, we also profit from the mature technology and remarkable speedups of MIP solvers, by using them to solve integer QP models as subproblems within the hybrid method, a new perspective brought by this thesis.

Furthermore, the ML framework developed in Chapter 4 is extended in the context presented in Chapter 6 and shows being even more significant to achieve a tighter integration alongside the hybrid approach. In this case, a trained classifier is invoked, on-the-fly, at each BDD node to guide the exploration.

Finally, computational experiments are performed for the MISP and QSSP to validate the efficiency of the proposed integrated method. For the MISP, the numerical results indicate that, in presence of suitable structure, the hybrid DD-MIP approach is competitive versus both a stand-alone BDD and MIP solvers. On the other hand, for the QSSP, the hybrid approach presents state-of-the-art results when it is compared with leading commercial MIP solvers and a SDP-based branch-and-bound solver.

CHAPTER 8 CONCLUSION AND RECOMMENDATIONS

Integrated optimization approaches are an effective alternative to exploit problem structure. We proposed hybrid optimization mechanisms to solve challenging combinatorial optimization problems by leveraging the decision diagrams-based optimization technology, a more classical paradigm such as mixed-integer programming, and machine learning. In Section 8.1, we summarize the contributions derived from this dissertation. We then conclude by discussing, in Section 8.2, the main limitations for each of the proposed contributions and suggest potential research avenues to be further explored.

8.1 Summary of works

Motivated by the research opportunities of the recent and fruitful ML-for-OR approach, Chapter 4 presents a machine learning framework to discriminate problem-specific instance hardness when using a MIP solver. The learning framework allows to learn a function based on both problem-specific features and the MIP solver performance. The interpretability properties of tree-based models are used to understand which problem features are the most relevant when discriminating instance hardness. In addition, the computational experiments show that the trained classifiers can determine the configuration of a MIP solver and boost its performance when solving the quadratic stable set problem. Finally, the framework is flexible enough to be extended to the context of algorithm selection which allows a tight integration of ML alongside hybrid optimization methods.

In Chapter 5, we propose an integrated optimization algorithm based on DDs and MIP for solving combinatorial optimization problems. The designed hybrid mechanisms profit from the two problem paradigms (i.e., DD and integer linear programming representations) to explore the solution space through an integrated search tree. The hybrid framework brings up novel pruning strategies for stand-alone DD solvers as well as tightly collaborative strategies to exploit information coming from the DD for the MIP representation, and vice versa. We use the maximum independent set problem as case study. The numerical experiments show that the hybrid DD-ILP approach is shown to be superior to a stand-alone DD approach and a leading MIP solver when suitable problem structure is identified.

Finally, in Chapter 6, we address the quadratic stable set problem and provide a novel problem representation for it through approximate DDs. Moreover, we adapt the hybrid DD-MIP by considering the QP capabilities of a MIP solver and test its efficiency in a more difficult nonlinear setup. Remarkably, we achieve a tight integration of ML alongside the hybrid approach. A ML model is used to train offline a SVM classifier that is invoked, on-the-fly, at each BDD subproblem within a branch-and-bound scheme. The computational experiments indicate that the hybrid BDD-MIP provides state-of-the-art results for solving the QSSP when compared against both leading commercial MIP solvers and a SDP-based branch and bound.

8.2 Limitations and future research

A natural extension of this dissertation is to adapt the proposed hybrid mechanisms for more challenging optimization problems and cases where a DD representation could be advantageous, such as in sequencing and scheduling problems. In principle, the only requirements other than a DD model are an additional problem representation from a different paradigm and a mechanism (e.g, machine learning based) to identify and exploit possible complementary strengths.

Nevertheless, the proposed hybrid mechanisms require a recursive formulation of the entire solution space to build the DD representation. Future work should consider recent threads on DDO where only subsets of constraints are modeled through DDs, extending its applicability to a more general class of discrete problems. Next, the hybrid mechanisms could be adapted to strengthen the DD representation of individual constraints.

The proposed hybrid algorithm relies heavily on MIP representations of the DD nodes during search. Possible future work concerns the integration of another paradigm such as constraint programming to replace the MIP paradigm. Nevertheless, given the flexible and generic nature of the proposed hybrid approach, it could be adapted to handle even more than two problem representations. The latter is worth to be considered as long as complementary strengths are identified and exploited.

The results of this dissertation suggests that the role of DDs as a means to explore the solution space and enumerate subproblems in a divide-and-conquer approach should be further explored. One possible direction is to profit from the DD representation in a multilevel optimization setting where different DDs can represent the solution space.

Related to the machine learning component, we could further improve the integration of ML within the hybrid method's performance by considering a multi-class classification framework. In this way, the learning task could consider different configurations of the DD representation based mainly on the performance when using different maximum width values. Similarly, for quadratic optimization problems and the MIP solver, the learning task could also consider the performance of the linearization option. The labels would then correspond to more than the two DD or MIP classes but, take into account different configurations of the solvers that could exploit further the problem structure.

REFERENCES

[1] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, “Learning combinatorial optimization algorithms over graphs,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS’17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6351–6361.

[2] A. Lodi, “Mixed integer programming computation,” in *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 619–645.

[3] J. N. Hooker, *Integrated Methods for Optimization.* Boston, MA: Springer US, 2012.

[4] J. N. Hooker and W.-J. van Hoeve, “Constraint programming and operations research,” *Constraints*, vol. 23, no. 2, pp. 172–195, Apr 2018.

[5] M. Milano and M. Wallace, “Integrating operations research in constraint programming,” *Annals of Operations Research*, vol. 175, no. 1, pp. 37–76, 2009.

[6] D. Bergman, A. A. Cire, W.-J. v. Hoeve, and J. Hooker, *Decision Diagrams for Optimization*, 1st ed. Springer International Publishing, 2016.

[7] Y. Pochet and L. A. Wolsey, Eds., *Production Planning by Mixed Integer Programming*, 1st ed. Springer-Verlag New York, 2006.

[8] P. Toth and D. Vigo, Eds., *Vehicle Routing: Problems, Methods, and Applications*, 2nd ed., ser. MOS-SIAM Series on Optimization. SIAM, 2014, no. 18.

[9] S. Akers, “Binary decision diagrams,” *Computers, IEEE Transactions on*, vol. C-27, no. 6, pp. 509–516, June 1978.

[10] C. Lee, “Representation of switching circuits by binary-decision programs,” *Bell System Technical Journal, The*, vol. 38, no. 4, pp. 985–999, July 1959.

[11] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, Aug 1986.

[12] H. Andersen, “An introduction to binary decision diagrams,” Lecture notes for Efficient Algorithms and Programs, 1999.

[13] I. Wegener, *Branching Programs and Binary Decision Diagrams.* Society for Industrial and Applied Mathematics, 2000. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9780898719789

[14] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann, "A constraint store based on multivalued decision diagrams," in *Principles and Practice of Constraint Programming – CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings*, C. Bessière, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 118–132.

[15] J. N. Hooker, "Decision diagrams and dynamic programming," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, C. Gomes and M. Sellmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 94–110.

[16] D. Bergman, A. A. Cire, W.-J. van Hoeve, and T. Yunes, "Bdd-based heuristics for binary optimization," *Journal of Heuristics*, vol. 20, no. 2, pp. 211–234, Apr 2014. [Online]. Available: https://doi.org/10.1007/s10732-014-9238-1

[17] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. Hooker, "Discrete optimization with decision diagrams," *INFORMS Journal on Computing*, vol. 28, no. 1, pp. 47–66, 2016.

[18] T. M. Mitchell, *Machine Learning.* New York: McGraw-Hill, 1997.

[19] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. [Online]. Available: https://science.sciencemag.org/content/349/6245/255

[20] C. Bishop, *Pattern Recognition and Machine Learning.* Springer-Verlag New York, 2006.

[21] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data mining, Inference and Prediction*, 2nd ed. Springer-Verlag New York, 2009.

[22] J. B. M.D., "Application of the logistic function to bio-assay," *Journal of the American Statistical Association*, vol. 39, no. 227, pp. 357–365, 1944.

[23] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[24] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[25] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[26] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] T. Achterberg, "Constraint integer programming," Ph.D. dissertation, Technische Universität Berlin, 2007.

[28] J. N. Hooker and G. Ottosson, "Logic-based Benders decomposition," *Mathematical Programming*, vol. 96, no. 1, pp. 33–60, 2003.

[29] J. N. Hooker, "Planning and scheduling by logic-based Benders decomposition," *Operations Research*, vol. 55, no. 3, pp. 588–602, 2007.

[30] C. E. Cortés, M. Gendreau, L. M. Rousseau, S. Souyris, and A. Weintraub, "Branch-and-price and constraint programming for solving a real-life technician dispatching problem," *European Journal of Operational Research*, vol. 238, no. 1, pp. 300–312, 2014.

[31] K. Easton, G. Nemhauser, and M. Trick, "Cp based branch-and-price," in *Constraint and Integer Programming: Toward a Unified Methodology*, M. Milano, Ed. Boston, MA: Springer US, 2004, pp. 207–231.

[32] B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer, "Bdds in a branch and cut framework," in *Experimental and Efficient Algorithms*, S. E. Nikoletseas, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 452–463.

[33] M. Behle, "Binary decision diagrams and integer programming," Ph.D. Thesis, Saarland University, Saarbrücken, Germany, 2007.

[34] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. Hooker, "Optimization bounds from binary decision diagrams," *INFORMS Journal on Computing*, vol. 26, no. 2, pp. 253–268, 2014.

[35] T. Hadzic, J. N. Hooker, B. O'Sullivan, and P. Tiedemann, "Approximate compilation of constraints into multivalued decision diagrams," in *Principles and Practice of Constraint Programming*, P. J. Stuckey, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 448–462.

[36] T. Hadzic, J. N. Hooker, and P. Tiedemann, "Propagating separable equalities in an mdd store," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, L. Perron and M. A. Trick, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 318–322.

[37] S. Hoda, W.-J. van Hoeve, and J. N. Hooker, "A systematic approach to mdd-based constraint programming," in *Principles and Practice of Constraint Programming – CP 2010*, D. Cohen, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 266–280.

[38] J. Kinable, A. A. Cire, and W.-J. van Hoeve, "Hybrid optimization methods for time-dependent sequencing problems," *European Journal of Operational Research*, vol. 259, no. 3, pp. 887 – 897, 2017.

[39] A. A. Cire and W.-J. van Hoeve, "Multivalued decision diagrams for sequencing problems," *Operations Research*, vol. 61, no. 6, pp. 1411–1428, 2013.

[40] M. P. Castro, A. A. Cire, and J. C. Beck, "An MDD-based lagrangian approach to the multicommodity pickup-and-delivery TSP," *INFORMS Journal on Computing*, 2019.

[41] D. Bergman, A. A. Cire, and W.-J. van Hoeve, "Lagrangian bounds from decision diagrams," *Constraints*, vol. 20, no. 3, pp. 346–361, 2015. [Online]. Available: https://doi.org/10.1007/s10601-015-9193-y

[42] C. Tjandraatmadja and W.-J. van Hoeve, "Target cuts from relaxed decision diagrams," *INFORMS Journal on Computing*, vol. 31, no. 2, pp. 285–301, 2019.

[43] D. Davarnia and W.-J. van Hoeve, "Outer approximation for integer nonlinear programs via decision diagrams," *Mathematical Programming*, 2020.

[44] D. R. Morrison, E. C. Sewell, and S. H. Jacobson, "Solving the pricing problem in a branch-and-price algorithm for graph coloring using zero-suppressed binary decision diagrams," *INFORMS Journal on Computing*, vol. 28, no. 1, pp. 67–82, 2016.

[45] D. Bergman and A. A. Cire, "Decomposition based on decision diagrams," in *Integration of AI and OR Techniques in Constraint Programming*, C.-G. Quimper, Ed. Cham: Springer International Publishing, 2016, pp. 45–54.

[46] D. Bergman and A. A. Cire, "Discrete nonlinear optimization by state-space decompositions," *Management Science*, vol. 64, no. 10, pp. 4700–4720, 2018.

[47] D. Bergman and L. Lozano, "Decision diagram decomposition for quadratically constrained binary optimization," *Preprint optimization-online*, 2018. [Online]. Available: http://www.optimization-online.org/DB_FILE/2018/10/6837.pdf

[48] L. Lozano and J. C. Smith, "A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs," *Mathematical Programming*, pp. 1–24, 2018.

[49] T. Serra, A. U. Raghunathan, D. Bergman, J. Hooker, and S. Kobori, "Last-mile scheduling under uncertainty," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, L.-M. Rousseau and K. Stergiou, Eds. Cham: Springer International Publishing, 2019, pp. 519–528.

[50] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *Preprint arXiv:1811.06128*, 2018.

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

[52] M. Gasse, D. Chetelat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 15 580–15 592. [Online]. Available: http://papers.nips.cc/paper/9690-exact-combinatorial-optimization-with-graph-convolutional-neural-networks.pdf

[53] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao, "Learning to run heuristics in tree search," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 659–666.

[54] Q. Cappart, E. Goutierre, D. Bergman, and L.-M. Rousseau, "Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence.* AAAI Press, 2019, p. 1443–1451.

[55] M. Kruber, M. E. Lübbecke, and A. Parmentier, "Learning when to use a decomposition," in *Integration of AI and OR Techniques in Constraint Programming*, D. Salvagnin and M. Lombardi, Eds. Cham: Springer International Publishing, 2017, pp. 202–210.

[56] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," in *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, C. Bessiere, L. De Raedt, L. Kotthoff, S. Nijssen, B. O'Sullivan, and D. Pedreschi, Eds. Cham: Springer International Publishing, 2016, pp. 149–190.

[57] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *J. Artif. Int. Res.*, vol. 32, no. 1, pp. 565–606, 2008.

[58] P. Bonami, A. Lodi, and G. Zarpellon, “Learning a classification of mixed-integer quadratic programming problems,” in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, W.-J. van Hoeve, Ed. Cham: Springer International Publishing, 2018, pp. 595–604.

[59] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, “Algorithm runtime prediction: Methods evaluation,” *Artificial Intelligence*, vol. 206, pp. 79 – 111, 2014.

[60] M. Fischetti, A. Lodi, and G. Zarpellon, “Learning milp resolution outcomes before reaching time-limit,” in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, L.-M. Rousseau and K. Stergiou, Eds. Cham: Springer International Publishing, 2019, pp. 275–291.

[61] C. McCreesh, P. Prosser, K. Simpson, and J. Trimble, “On maximum weight clique algorithms, and how they are evaluated,” in *Principles and Practice of Constraint Programming*, J. C. Beck, Ed. Cham: Springer International Publishing, 2017, pp. 206–225.

[62] A. Mehrotra and M. A. Trick, “A column generation approach for graph coloring,” *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 344–354, 1996.

[63] T. Hemazro, B. Jaumard, and O. Marcotte, “A column generation and branch-and-cut algorithm for the channel assignment problem,” *Computers & Operations Research*, vol. 35, no. 4, pp. 1204 – 1226, 2008.

[64] M. Grötschel, L. Lovász, and A. Schrijver, “Stable sets in graphs,” in *Geometric Algorithms and Combinatorial Optimization.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 272–303.

[65] S. Butenko, “Maximum independent set and related problems, with applications,” Ph.D. dissertation, University of Florida, USA, 2003.

[66] L. Xiaoli, W. Min, K. Chee-Keong, and N. See-Kiong, “Computational approaches for detecting protein complexes from protein interaction networks: a survey,” *BMC Genomics*, vol. 11, no. 1, 2010.

[67] L. Cavique, “A scalable algorithm for the market basket analysis,” *Journal of Retailing and Consumer Services*, vol. 14, no. 6, pp. 400 – 407, 2007.

[68] A. Lodi and A. Tramontani, “Performance variability in mixed-integer programming,” in *Theory Driven by Influential Applications.* INFORMS, 2014, pp. 1–12.

[69] P. Erdös and A. Rényi, "On the evolution of random graphs," in *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, vol. 5, 1960, pp. 17–61.

[70] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, 1998.

[71] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[72] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Physical Review E*, vol. 65, no. 026107, 2002.

[73] M. E. J. Newman, "Mixing patterns in networks," *Phys. Rev. E*, vol. 67, p. 026126, Feb 2003.

[74] A. Esfahanian, "Connectivity algorithms," in *Topics in Structural Graph Theory*, L. W. Beineke and R. J. Wilson, Eds. Cambridge University Press, 2012, pp. 268–281.

[75] T. Schank and D. Wagner, "Approximating clustering coefficient and transitivity," *Journal of Graph Algorithms and Applications*, vol. 9, no. 2, pp. 265–275, 2005.

[76] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078195

[77] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[78] M. Gendreau, P. Soriano, and L. Salvail, "Solving the maximum clique problem using a tabu search approach," *Annals of Operations Research*, vol. 41, no. 4, pp. 385–403, 1993.

[79] F. Furini and E. Traversi, "Theoretical and computational study of several linearisation techniques for binary quadratic problems," *Annals of Operations Research*, vol. 279, no. 1, pp. 387–411, Aug 2019.

[80] J. E. González, A. A. Cire, A. Lodi, and L.-M. Rousseau, "Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem," *Constraints*, vol. 25, no. 1, pp. 23–46, 2020.

[81] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker, "Variable ordering for the application of BDDs to the maximum independent set problem," in *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, N. Beldiceanu, N. Jussien, and É. Pinson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 34–49.

[82] J. E. González, A. A. Cire, A. Lodi, and L.-M. Rousseau, "Bdd-based optimization for the quadratic stable set problem," *Discrete Optimization*, p. 100610, 2020.

[83] B. Balasundaram, S. Butenko, and I. V. Hicks, "Clique relaxations in social network analysis: The maximum k-plex problem," *Operations Research*, vol. 59, no. 1, p. 133–142, Jan. 2011.

[84] J. D. Eblen, C. A. Phillips, G. L. Rogers, and M. A. Langston, "The maximum clique enumeration problem: Algorithms, applications and implementations," in *Bioinformatics Research and Applications*, J. Chen, J. Wang, and A. Zelikovsky, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 306–319.

[85] Q. Wu and J.-K. Hao, "A review on algorithms for maximum clique problems," *European Journal of Operational Research*, vol. 242, no. 3, pp. 693 – 709, 2015.

[86] B. Jaumard, O. Marcotte, and C. Meyer, "Estimation of the quality of cellular networks using column generation techniques," *GERAD Technical Report*, vol. G-98-02, 1998.

[87] S. Karimi and P. Ronagh, "A subgradient approach for constrained binary optimization via quantum adiabatic evolution," *Quantum Information Processing*, vol. 16, no. 8, p. 185, 2017.

[88] F. Furini and E. Traversi, "Hybrid SDP bounding procedure," in *Experimental Algorithms*, V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 248–259.

[89] N. Krislock, J. Malick, and F. Roupin, "Biqcrunch: A semidefinite branch-and-bound method for solving binary quadratic problems," *ACM Trans. Math. Softw.*, vol. 43, no. 4, pp. 32:1–32:23, Jan. 2017.

[90] S. Hosseinian, D. B. M. M. Fontes, S. Butenko, M. B. Nardelli, M. Fornari, and S. Curtarolo, "The maximum edge weight clique problem: Formulations and solution approaches," in *Optimization Methods and Applications : In Honor of Ivan V. Sergienko's 80th Birthday*, S. Butenko, P. M. Pardalos, and V. Shylo, Eds. Cham: Springer International Publishing, 2017, pp. 217–237.

[91] P. S. Segundo, S. Coniglio, F. Furini, and I. Ljubić, “A new branch-and-bound algorithm for the maximum edge-weighted clique problem,” *European Journal of Operational Research*, vol. 278, no. 1, pp. 76 – 90, 2019.

[92] S. Hosseinian, D. B. M. M. Fontes, and S. Butenko, “A lagrangian bound on the clique number and an exact algorithm for the maximum edge weight clique problem,” *INFORMS Journal on Computing*, 2019.

[93] S. Shimizu, K. Yamaguchi, and S. Masuda, “A branch-and-bound based exact algorithm for the maximum edge-weight clique problem,” in *Computational Science/Intelligence & Applied Informatics*, R. Lee, Ed. Cham: Springer International Publishing, 2019, pp. 27–47.

[94] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[95] N. Krislock, J. Malick, and F. Roupin. (2019) BiqCrunch numerical results. [Online]. Available: https://www-lipn.univ-paris13.fr/BiqCrunch/results

APPENDIX A EXTENDED COMPUTATIONAL EXPERIMENTS - BDD-BASED OPTIMIZATION FOR THE QUADRATIC STABLE SET PROBLEM

Tables A.1, A.2, and A.3 present the extended computational experiments associated with the summarized Table 6.2 which is discussed in Section 6.5. Columns 1, 2, and 3 are associated to the instance properties, size n, density p, and percentage of positive coefficients in Q v, respectively. Columns 4 and 5 correspond to the instance id for a given $n - p - v$ and the optimal objective value. Columns 6, 7, 8, 9, and 10, present the solving time when using BiqCrunch, CPLEX linearizing, CPLEX non-linearizing, Gurobi, and the Hybrid BDD-MIP, respectively. The time limit is set to $7,200$ seconds. Finally, column 11 presents the speedup obtained by the BDD-MIP algorithm with respect to the best solving time between all the benchmarks.

Similarly, Tables A.4, A.5, and A.6 present the extended computational experiments on larger instances with size $n \in \{175, 200, 250\}$ associated with the summarized Table 6.3 which is discussed in Section 6.5. Columns 1, 2, and 3 are associated to the instance properties, size n, density p, and percentage of positive coefficients in Q (v), respectively. Column 4 corresponds to the instance id for a given $n - p - v$. Columns 5, 6, 7, and 8 are associated with Gurobi and present the best lower bound (LB), the best upper bound (UB), the optimality gap (calculated as (UB-LB)/UB), and the computing time. Columns 9, 10, 11, 12, 13, and 14 are related to the hybrid BDD-MIP. They present, in order from 9 to 14, the number of BDD nodes explored, the number of BDD nodes pruned by the MIP solver, the best lower bound (LB), the best upper bound (UB), the optimality gap, and the computing time, respectively. The time limit is set to $7,200$ seconds. Finally, column 15 presents the speedup obtained by the BDD-MIP algorithm with respect to Gurobi.

Table A.1 Comparison between the hybrid BDD-MIP algorithm and other optimization paradigms for the QSSP

Instance					**BiqCrunch**	**CPLEX**		**Gurobi**	**Hybrid BDD-MIP**	**Speedups**
						Linearize	Not Linearize			
n	p	v	#	Opt. Value	CPU (s)	CPU (s)	CPU (s)	CPU (s)	CPU (s)	vs. best benchmark
100	25	25	1	882	525.68	234.79	83.75	62.26	17.22	3.62
100	25	25	2	813	1106.77	318.86	130.86	97.60	25.49	3.83
100	25	25	3	792	1785.48	264.29	124.78	94.40	30.80	3.06
100	25	50	1	2576	7200.00	845.72	119.33	259.56	23.68	5.04
100	25	50	2	2468	7200.00	662.94	163.72	369.53	40.01	4.09
100	25	50	3	2509	7200.00	676.46	132.64	233.55	44.78	2.96
100	25	75	1	4758	7200.00	1257.45	1516.95	189.03	175.52	1.08
100	25	75	2	4877	7200.00	951.13	1606.76	218.82	168.34	1.30
100	25	75	3	5014	7200.00	819.99	1042.85	144.14	157.21	0.92
100	50	25	1	607	831.41	17.61	10.35	10.89	0.68	15.22
100	50	25	2	459	814.58	18.94	9.90	11.64	0.70	14.14
100	50	25	3	542	842.47	18.30	8.76	10.74	0.78	11.23
100	50	50	1	1163	4100.00	48.09	13.00	11.46	0.80	14.33
100	50	50	2	1138	4493.42	55.01	11.72	10.36	0.80	12.95
100	50	50	3	1002	5902.90	60.56	15.78	13.08	0.90	14.53
100	50	75	1	1829	7038.16	90.58	29.85	12.72	0.73	17.42
100	50	75	2	2047	5948.31	84.30	27.43	14.38	0.70	20.54
100	50	75	3	1748	7046.40	93.89	34.25	16.02	0.78	20.54

Table A.2 (Continued) Comparison between the hybrid BDD-MIP algorithm and other optimization paradigms for the QSSP

Instance					**BiqCrunch**	**CPLEX**		**Gurobi**	**Hybrid BDD-MIP**	**Speedups**
						Linearize	Not Linearize			
n	p	v	#	Opt. Value	CPU (s)	CPU (s)	CPU (s)	CPU (s)	CPU (s)	vs. best benchmark
100	75	25	1	267	527.32	5.04	4.68	1.01	0.05	20.20
100	75	25	2	378	440.87	3.19	3.86	0.97	0.06	16.17
100	75	25	3	273	417.62	3.45	4.00	1.12	0.06	18.67
100	75	50	1	483	1435.61	15.82	4.69	1.57	0.05	31.40
100	75	50	2	507	1335.20	18.28	3.96	1.52	0.06	25.33
100	75	50	3	525	1299.99	17.90	4.13	1.61	0.06	26.83
100	75	75	1	607	1255.21	29.69	5.95	1.66	0.05	33.20
100	75	75	2	757	1148.03	36.99	6.14	1.79	0.05	35.80
100	75	75	3	843	1084.15	24.79	6.28	1.67	0.06	27.83
150	25	25	1	1248	7200.00	5638.62	7200.00	1749.09	531.48	3.29
150	25	25	2	1229	7200.00	4117.49	5267.68	1726.29	585.53	2.95
150	25	25	3	1116	7200.00	4572.44	7200.00	1348.20	565.40	2.38
150	25	50	1	3322	7200.00	7200.00	7200.00	7200.00	824.74	8.73
150	25	50	2	3016	7200.00	7200.00	7200.00	7200.00	1078.56	6.68
150	25	50	3	3225	7200.00	7200.00	7200.00	6356.28	761.80	8.34
150	25	75	1	6962	7200.00	7200.00	7200.00	7200.00	2490.08	2.89
150	25	75	2	6438	7200.00	7200.00	7200.00	7200.00	3295.71	2.18
150	25	75	3	6332	7200.00	7200.00	7200.00	7200.00	3090.42	2.33

Table A.3 (Continued) Comparison between the hybrid BDD-MIP algorithm and other optimization paradigms for the QSSP

Instance					**BiqCrunch**	**CPLEX**		**Gurobi**	**Hybrid BDD-MIP**	**Speedups**
						Linearize	Not Linearize			
n	p	v	#	Opt. Value	CPU (s)	CPU (s)	CPU (s)	CPU (s)	CPU (s)	vs. best benchmark
150	50	25	1	665	6303.99	133.97	123.42	46.13	7.29	6.33
150	50	25	2	627	7200.00	134.50	135.07	44.77	7.85	5.70
150	50	25	3	642	7200.00	121.71	148.43	47.13	7.22	6.53
150	50	50	1	1500	7200.00	1166.65	168.72	82.57	7.03	11.75
150	50	50	2	1317	7200.00	1134.84	188.60	108.58	7.20	15.08
150	50	50	3	1322	7200.00	1251.19	189.40	97.30	7.94	12.25
150	50	75	1	2377	7200.00	804.81	540.03	89.54	6.52	13.73
150	50	75	2	2053	7200.00	2937.07	633.60	102.46	7.26	14.11
150	50	75	3	2293	7200.00	1516.94	573.47	82.39	6.70	12.30
150	75	25	1	451	2235.84	40.68	20.91	4.67	0.21	22.24
150	75	25	2	384	2971.97	34.10	23.21	4.61	0.21	21.95
150	75	25	3	347	3478.83	46.26	25.14	39.25	0.21	119.71
150	75	50	1	787	7200.00	79.54	25.75	44.67	0.18	143.06
150	75	50	2	873	7082.92	69.25	24.54	39.15	0.19	129.16
150	75	50	3	663	7200.00	88.54	29.18	45.32	0.20	145.90
150	75	75	1	895	7200.00	94.96	38.96	36.14	0.20	180.70
150	75	75	2	877	7200.00	99.28	37.58	65.96	0.22	170.82
150	75	75	3	888	7200.00	67.40	39.20	44.73	0.21	186.67

Table A.4 Comparison between the hybrid BDD-MIP and Gurobi for QSSP instances with $n \in \{175, 200, 250\}$

Instances				Gurobi				Hybrid BDD-MIP						Speedups
Group	n	p	v	Best LB	Best UB	Opt. Gap	CPU time (s)	BDD-nodes explored	subMIPs solved	Best LB	Best UB	Opt. Gap	CPU time (s)	
175	25	25	1	1298	1298	0	6063.74	100104	96669	1298	1298	0	2983.63	2.03
175	25	25	2	1118	2188	0.96	7200.00	136983	131702	1118	1118	0	4639.45	1.55
175	25	25	3	1043	2649	1.54	7200.00	138306	133879	1043	1043	0	4774.72	1.51
175	25	50	1	3207	9869	2.08	7200.00	117025	113546	3774	3774	0	4410.49	1.63
175	25	50	2	2803	9832	2.51	7200.00	109835	106377	2995	2995	0	5192.52	1.39
175	25	50	3	3671	9238	1.52	7200.00	114012	110094	3671	3671	0	4455.41	1.62
175	25	75	1	7680	15409	1.01	7200.00	83874	81458	7680	7680	0	5798.34	1.24
175	25	75	2	6349	16314	1.57	7200.00	45098	42883	6587	13787	1.09	7200.00	1.00
175	25	75	3	6482	16216	1.50	7200.00	53546	51560	6686	13022	0.95	7200.00	1.00
175	50	25	1	717	717	0	87.59	7771	0	717	717	0	15.29	5.73
175	50	25	2	895	895	0	119.58	5631	0	895	895	0	12.85	9.31
175	50	25	3	728	728	0	144.89	8112	0	728	728	0	16.77	8.64
175	50	50	1	1263	1263	0	227.98	15932	1	1263	1263	0	23.38	9.75
175	50	50	2	1351	1351	0	231.20	16065	6	1351	1351	0	23.79	9.72
175	50	50	3	1249	1249	0	210.36	17719	1	1249	1249	0	24.87	8.46
175	50	75	1	2298	2298	0	262.08	12384	9	2298	2298	0	19.13	13.70
175	50	75	2	2145	2145	0	251.34	14259	4	2145	2145	0	18.96	13.26
175	50	75	3	2335	2335	0	269.36	14161	3	2335	2335	0	21.62	12.46
175	75	25	1	371	371	0	53.68	255	0	371	371	0	0.34	157.88
175	75	25	2	402	402	0	61.77	246	0	402	402	0	0.33	187.18
175	75	25	3	409	409	0	58.75	260	0	409	409	0	0.38	154.61
175	75	50	1	642	642	0	88.12	373	0	642	642	0	0.36	244.78
175	75	50	2	621	621	0	62.12	308	0	621	621	0	0.35	177.49
175	75	50	3	713	713	0	76.97	285	0	713	713	0	0.34	226.38
175	75	75	1	900	900	0	102.81	528	0	900	900	0	0.40	257.03
175	75	75	2	917	917	0	72.70	387	0	917	917	0	0.35	207.71
175	75	75	3	1048	1048	0	75.28	376	0	1048	1048	0	0.37	203.46

Table A.5 (Continued) Comparison between the hybrid BDD-MIP and Gurobi for QSSP instances with $n \in \{175, 200, 250\}$

Instances				Gurobi				Hybrid BDD-MIP						Speedups
Group	n	p	v	Best LB	Best UB	Opt. Gap	CPU time (s)	BDD-nodes explored	subMIPs solved	Best LB	Best UB	Opt. Gap	CPU time (s)	
200	25	25	1	1233	5102	3.14	7200.00	98742	91409	1228	5495	3.47	7200.00	1.00
200	25	25	2	1229	5587	3.55	7200.00	91540	83594	1226	5651	3.61	7200.00	1.00
200	25	25	3	1255	5436	3.33	7200.00	89152	81142	1239	5612	3.53	7200.00	1.00
200	25	50	1	3171	16197	4.11	7200.00	72480	64032	3849	12251	2.18	7200.00	1.00
200	25	50	2	3136	16659	4.31	7200.00	63830	55759	3373	13135	2.89	7200.00	1.00
200	25	50	3	3001	15599	4.20	7200.00	73882	66552	3558	11944	2.36	7200.00	1.00
200	25	75	1	5850	25718	3.40	7200.00	20924	15712	7170	20751	1.89	7200.00	1.00
200	25	75	2	7556	24668	2.26	7200.00	33217	26997	7822	19175	1.45	7200.00	1.00
200	25	75	3	7448	25614	2.44	7200.00	35343	29168	7896	19121	1.42	7200.00	1.00
200	50	25	1	733	733	0	227.33	15466	0	733	733	0	36.39	6.25
200	50	25	2	755	755	0	252.20	14753	0	755	755	0	35.42	7.12
200	50	25	3	712	712	0	309.93	16279	0	712	712	0	41.92	7.39
200	50	50	1	1443	1443	0	813.54	30418	1	1443	1443	0	61.04	13.33
200	50	50	2	1433	1433	0	490.51	26524	0	1433	1433	0	52.33	9.37
200	50	50	3	1570	1570	0	426.39	25492	1	1570	1570	0	50.36	8.47
200	50	75	1	2550	2550	0	478.93	22468	9	2550	2550	0	46.65	10.27
200	50	75	2	2445	2445	0	522.66	19816	1	2445	2445	0	36.93	14.15
200	50	75	3	2410	2410	0	513.28	25190	8	2410	2410	0	48.76	10.53
200	75	25	1	484	484	0	106.55	351	0	484	484	0	0.52	204.90
200	75	25	2	390	390	0	100.98	394	0	390	390	0	0.54	187.00
200	75	25	3	546	546	0	116.14	251	0	546	546	0	0.53	219.13
200	75	50	1	905	905	0	129.93	374	0	905	905	0	0.54	240.61
200	75	50	2	956	956	0	112.08	367	0	956	956	0	0.53	211.47
200	75	50	3	714	714	0	144.32	399	0	714	714	0	0.51	282.98
200	75	75	1	1106	1106	0	158.00	558	0	1106	1106	0	0.61	259.02
200	75	75	2	1050	1050	0	153.22	539	0	1050	1050	0	0.56	273.61
200	75	75	3	1219	1219	0	172.42	339	0	1219	1219	0	0.51	338.08

Table A.6 (Continued) Comparison between the hybrid BDD-MIP and Gurobi for QSSP instances with $n \in \{175, 200, 250\}$

Instances				**Gurobi**				**Hybrid BDD-MIP**						**Speedups**
Group	n	p	v	Best LB	Best UB	Opt. Gap	CPU time (s)	BDD-nodes explored	subMIPs solved	Best LB	Best UB	Opt. Gap	CPU time (s)	
250	50	25	1	898	898	0	1149.65	34998	3	898	898	0	157.53	7.30
250	50	25	2	826	826	0	733.51	44231	1	826	826	0	192.68	3.81
250	50	25	3	874	874	0	1184.44	34912	1	874	874	0	164.12	7.22
250	50	50	1	1557	1557	0	2208.99	84926	3	1557	1557	0	428.82	5.15
250	50	50	2	1532	1532	0	2159.86	84933	4	1532	1532	0	463.70	4.66
250	50	50	3	1645	1645	0	2372.39	81951	8	1645	1645	0	432.17	5.49
250	50	75	1	2744	2744	0	2889.24	86741	77	2744	2744	0	476.84	6.06
250	50	75	2	2678	2678	0	2950.96	97142	77	2678	2678	0	427.36	6.91
250	50	75	3	2751	2751	0	2546.23	77539	30	2751	2751	0	416.00	6.12
250	75	25	1	560	560	0	447.82	429	0	560	560	0	1.27	352.61
250	75	25	2	446	446	0	356.40	723	0	446	446	0	1.42	250.99
250	75	25	3	426	426	0	369.74	775	0	426	426	0	1.50	246.49
250	75	50	1	771	771	0	422.91	1343	0	771	771	0	1.54	274.62
250	75	50	2	828	828	0	418.07	1135	0	828	828	0	1.41	296.50
250	75	50	3	709	709	0	670.93	2096	0	709	709	0	1.72	390.08
250	75	75	1	1103	1103	0	533.13	1543	0	1103	1103	0	1.52	350.74
250	75	75	2	1067	1067	0	501.15	2273	0	1067	1067	0	1.69	296.54
250	75	75	3	1176	1176	0	580.04	1367	0	1176	1176	0	1.40	414.31