Combining Constraint Programming and Machine Learning: From Current Progress to Future Opportunities

QUENTIN CAPPART*, Polytechnique Montreal, Canada and UCLouvain, Belgium
TIAS GUNS, KU Leuven, Belgium
MICHELE LOMBARDI, Università di Bologna, Italy
GILLES PESANT, Polytechnique Montreal, Canada
DIMOS TSOUROS, KU Leuven, Belgium and University of Western Macedonia, Greece

The integration of constraint programming (CP) together with machine learning (ML) has emerged as a promising direction for tackling complex decision-making and combinatorial optimization problems. While CP offers expressive modeling capabilities and formal guarantees, ML provides adaptive methods for learning from data and generalizing across instances. This survey presents a comprehensive overview of recent advances in combining CP and ML. We first show how ML has been used to improve the CP toolbox, both in modeling and in the efficiency of solving. Then, we examine how CP can support ML, particularly in providing structure, guarantees, and symbolic reasoning capabilities. Finally, we identify key open challenges inherent to such hybrid approaches and outline promising directions for future research. This survey provides a first conceptual and structured review of recent advancements in this emerging field, aiming to serve as a resource for practitioners and researchers in both the CP and ML communities. To keep the progress up to date, a curated list of references is hosted on an accompanying repository (https://github.com/corail-research/CPML-paper-list) and is open to community contributions.

JAIR Track: Constraint Programming and Machine Learning

JAIR Associate Editor: Chu-Min LI

JAIR Reference Format:
Quentin Cappart, Tias Guns, Michele Lombardi, Gilles Pesant, and Dimos Tsouros. 2025. Combining Constraint Programming and Machine Learning: From Current Progress to Future Opportunities. *Journal of Artificial Intelligence Research* 84, Article 28 (December 2025), 52 pages. DOI: 10.1613/jair.1.19533

1 Introduction

Constraint programming (CP) (Rossi et al. 2006) is a computing paradigm that focuses on solving combinatorial problems by specifying constraints that must be satisfied, rather than providing explicit procedural steps to reach a solution. It has deep historical roots in logic, mathematics, artificial intelligence, and operations research. At its core, CP is generally understood through the lens of three fundamental components:

$$\text{CP} = \text{Modeling} + \text{Search} + \text{Reasoning}.$$

*Corresponding Author.

Authors' Contact Information: Quentin Cappart, ORCID: 0000-0002-8742-0774, quentin.cappart@polymtl.ca, Polytechnique Montreal, Montréal, Québec, Canada and UCLouvain, Louvain-la-Neuve, , Belgium; Tias Guns, ORCID: 0000-0002-2156-2155, tias.guns@kuleuven.be, KU Leuven, Leuven, , Belgium; Michele Lombardi, ORCID: 0000-0003-4709-8888, michele.lombardi2@unibo.it, Università di Bologna, Bologna, , Italy; Gilles Pesant, ORCID: 0000-0001-9797-0780, gilles.pesant@polymtl.ca, Polytechnique Montreal, Montréal, Québec, Canada; Dimos Tsouros, ORCID: 0000-0002-3040-0959, dtsouros@uowm.gr, KU Leuven, Leuven, , Belgium and University of Western Macedonia, Kozani, , Greece.

DOI: 10.1613/jair.1.19533

Modeling in CP is mainly about representing a problem in a declarative way, specifying what conditions a valid solution must satisfy rather than describing how to compute it. The goal is to make CP technology accessible to non-experts, enabling them to focus on describing the problem rather than solving it: "*the user states the problem, the computer solves it.*" (E. Freuder 1996). This vision, often referred to in CP folklore as the pursuit of the Holy Grail, encapsulates the ideal of fully automated problem-solving (Barták 1999).

Once a problem is modeled, the next step is to perform a *search* for solutions by systematically assigning values to variables, leading to an exploration of a search tree. Standard CP solvers typically use depth-first search with backtracking, allowing the solver to revisit previous decisions when constraints are violated. Additionally, branching strategies determine the order in which variables and values are considered. These strategies are often guided by heuristics to prioritize promising parts of the solution space, such as the *first-fail principle*, which assigns values to the variable with the fewest remaining options first. Heuristic search is unfortunately insufficient to achieve acceptable performance on challenging combinatorial problems. To address this, *constraint propagation*, a form of logical reasoning, was introduced as a core component of CP solving. The key idea is to prune infeasible values from variable domains before branching at each node, reducing the search space and improving efficiency.

Since its inception, modeling, search, and propagation have been the foundation of CP technology. These principles have guided the development of various solvers (e.g. Gecode (Schulte et al. 2006), Choco (Prud'homme and Fages 2022), CPOptimizer (Laborie, Rogerie, et al. 2018)) and modeling tools (e.g. MiniZinc (Nethercote et al. 2007), XCSP3 (Audemard, Boussemart, et al. 2020), CPMpy (Guns 2019)), all built upon these core concepts. However, despite its strengths, CP has faced persistent limitations over the years. Two challenges, in particular, directly hinder its strategic vision of easily modeling combinatorial problems and solving them efficiently. The first is *scalability*: CP can struggle with large-scale problems as the search space grows exponentially, highlighting the need for more efficient search and propagation mechanisms. The second is its *reliance on expert knowledge* for modeling. Defining appropriate constraints often requires domain expertise, making CP less accessible to non-specialists despite the great advancements in modeling tools.

Interestingly, these limitations were already recognized from the early days of CP. Barták (1999) highlighted the challenge of selecting the appropriate propagator for a given problem and the difficulty of handling objective functions effectively. Later, Puget (2004) raised concerns about CP's complexity, noting that it remained too difficult for non-experts and, more critically, that ongoing research in the field tends to exacerbate this issue. More recently, E. C. Freuder (2018) provided a comprehensive analysis of CP's advancements and the challenges that remain. Three key directions were proposed for future research: automating the modeling process, automating the solving process, and automating the explanation of solutions. Each of them aimed at making CP more accessible and practical for a broader range of users.

Automating parts of the CP toolbox has then been recognized as a promising opportunity for the community. However, achieving this level of automation is challenging with the existing tools. To address this, several researchers have advocated for integrating an additional core mechanism, one inherently suited to automation, into the foundation of CP: *machine learning*. This shift would redefine the standard vision of CP, incorporating learning alongside modeling, search, and propagation to enhance efficiency and accessibility:

$$\text{Future CP} = \text{Modeling} + \text{Search} + \text{Reasoning} + \text{Learning}.$$

However, a natural question arises: *"How can machine learning be appropriately used for improving CP?"* This question is still open and has generated a lot of interest in the community. Notable collaborative initiatives in this direction include the establishment of the workshop on *Progress Towards the Holy Grail* at the flagship CP conference since 2017, the *Constraint Programming and Machine Learning Bridge* at AAAI since 2023, the *Dagstuhl Seminars* 22431 (Frejinger et al. 2023) and 24441 (Ajwani et al. 2025), and the special issue on CP and learning in the *Journal of Artificial Intelligence Research*. This is in addition to a growing body of research publications in the

field. Surveying and classifying these works comprehensively is the first contribution of this paper.

Machine learning (ML) is the ability of a computer program to improve its performance on a task, as measured by a performance metric, through experience (Mitchell 1997). This concept forms the foundation of various ML paradigms, including supervised learning (i.e. learning from labeled data), unsupervised learning (i.e. identifying patterns without labels), and reinforcement learning (i.e. learning through interaction and feedback). This technology has revolutionized numerous fields by enabling computers to learn patterns from data and improve their performance over time. However, traditional machine learning approaches, such as decision trees, random forests, and shallow neural networks have limitations. These models often require manual feature engineering, struggle with high-dimensional data, and lack the ability to generalize effectively from large-scale and structured information such as images, text, or graphs. The advent of deep learning in the late 2000s marked a major breakthrough in overcoming these limitations. *Deep learning* (LeCun et al. 2015), a term coined for artificial neural networks with multiple layers, leverages hierarchical representations to automatically extract features from raw data, eliminating the need for extensive manual preprocessing. Since its introduction, deep learning has been a cornerstone of several cutting-edge technologies, driving breakthroughs across various domains. Notable examples include AlphaGo for decision-making in games (Silver et al. 2016), AlphaFold for protein structure prediction (Jumper et al. 2021), and large language models for conversational agents (Brown et al. 2020; Vaswani et al. 2017), among many others.

Despite its success, machine learning, including deep learning, also faces fundamental limitations that hinder its applicability and robustness for many applications. One significant limitation is its inability to reason and understand symbolic information. By design, ML excels at pattern recognition, relying on statistical inference derived from acquired data. However, in its basic form, it lacks an inherent notion of truth or formal logic. As a result, ML struggles with logical reasoning, causal inference, and symbolic manipulation. This limitation makes it less effective for tasks requiring structured reasoning, such as mathematical theorem proving, formal verification, and solving combinatorial problems. As a concrete example, Chollet (2019) introduced the *abstraction and reasoning corpus* (ARC) benchmark in 2019 as an initial attempt to evaluate how well machine learning approaches can tackle tasks requiring logical reasoning and abstraction. The benchmark is not yet fully solved, even after five years, with state-of-the-art large language models struggling to reach 50% accuracy. At the time of writing, the best-performing approach is *openAI o3* reasoning-based large language model, achieving a score of 75%[1]. Shojaee et al. (2025) also pointed out the limitations of large language models, focusing on reasoning models, for exact computation of solutions to reasoning problems.

Another significant limitation is the lack of guarantees on the output (i.e. the *predictions*) of an ML model. Since predictions are derived through statistical inference, errors are inevitable. Moreover, ML models are highly susceptible to adversarial attacks, i.e. small, carefully crafted input perturbations that can lead to incorrect predictions with high confidence. This vulnerability makes ML unsuitable for high-stakes applications, where a single error can have severe consequences. These challenges highlight the persistent limitations of current ML techniques in tasks requiring rigorous reasoning or high guarantees.

A promising approach to addressing the intrinsic limitations of learning is to incorporate reasoning and symbolic models into it. The term *neurosymbolic* (Garcez et al. 2008) is commonly used to describe methods that integrate symbolic reasoning with deep learning. However, we emphasize that even machine learning models not based on neural networks (e.g. decision trees) can also benefit from reasoning modules. Given its versatility and strong foundation in search and logical reasoning, CP has emerged as a promising approach to overcoming the limitations of learning-based methods. Surveying how CP can mitigate the shortcomings of learning constitutes

[1]https://arcprize.org/leaderboard

the second contribution of this paper.

In the long term, we believe that integrating ML and CP has the potential to play a key role in future AI architectures. However, it is important to clarify that this survey does not advocate for CP+ML hybridization as the overall best approach for tackling combinatorial optimization problems in the short term. Significant limitations and fundamental challenges remain, requiring further research before these methods can reach their full potential. The third contribution of this survey is to present a comprehensive overview of these challenges and to highlight promising open research directions that we believe can drive future advancements in the field.

1.1 Summary of Contributions and Scope of the Survey

The aim of this survey is to provide an overview of recent advances in combining constraint programming and machine learning, with the intent of bridging and informing both research communities. To this end, we begin by introducing the technical background on the core mechanisms underlying CP and ML. Our main contributions are as follows:

(1) We review approaches that integrate learning into CP technology, either to assist in the modeling process or to accelerate the solving phase.
(2) We survey how CP can address certain limitations of ML, particularly regarding guarantees, explainability, and reasoning capabilities. One notable example is the management of uncertain information in an optimization problem.
(3) We highlight that this survey should not be interpreted as a recommendation that combining learning with constraint programming is currently the most effective or appropriate approach for all tasks. We outline the key obstacles in building such hybrid systems and provide a list of open research directions to inspire future work.

The main references discussed in this survey are also summarized in an accompanying repository[2]. The goal is to keep the list up to date and to allow the community to contribute new references. We also emphasize that this survey is exclusively focused on methods that involve both a CP and an ML component within their overall pipeline. For example, this includes approaches where a trained ML model is used as a heuristic within a CP solver, learning mechanisms that improve branching decisions in an online setting, or CP solvers used to enforce constraints on the outputs of generative ML models. Conversely, we explicitly exclude related work in which either the CP or ML component is absent. Concrete examples of excluded topics include pure algorithmic propagators in CP, learning techniques for mixed-integer programming solvers, decision-focused learning approaches that do not involve CP, or learning-based heuristics that are not integrated into a CP framework.

1.2 Related Work Covered by Other Surveys

To our knowledge, no other survey is fully and extensively dedicated to the integration of CP and ML. Existing surveys either focus on specific aspects of how ML can enhance CP (e.g. learning constraints) or cover broader fields (e.g. ML for combinatorial optimization), where CP is addressed only superficially, or not at all. This gap highlights the need for a comprehensive examination of the interplay between CP and ML, which this survey aims to provide.

In the late 1990s, Smith (1999) reviewed the use of neural networks for combinatorial optimization. More recently, Bengio et al. (2021) identified three key ways ML can be leveraged for combinatorial optimization: end-to-end learning, learning to configure algorithms, and integrating ML with optimization algorithms. Specific aspects of these approaches have been later surveyed in greater detail, such as end-to-end learning for constrained optimization by Kotary et al. (2021) and automated algorithm configuration by Schede et al. (2022). Additionally,

[2] https://github.com/corail-research/CPML-paper-list

several surveys have explored how ML can enhance optimization algorithms. For example, Lodi and Zarpellon (2017) focused on learning to branch but only within the context of the branch-and-bound algorithm for mixed-integer programming. Expanding beyond branching, Scavuzzo et al. (2024) reviewed ML-driven enhancements for primal heuristics, cutting planes, node selection, and solver configuration, yet still exclusively within the scope of mixed-integer programming. On another front, Mandi, Kotary, et al. (2024) explored decision-focused learning, which aims to improve decision quality by training ML models in an end-to-end optimization system, and Sadana et al. (2025) surveyed contextual optimization methods for decision-making under uncertainty. Despite these advancements, few of the works covered there focus on CP specifically.

Lombardi and Milano (2018) reviewed various approaches designed to enhance the modeling process for combinatorial optimization, including CP among other technologies. Later, Fajemisin et al. (2024) introduced a framework for optimization with constraint learning, further extending the discussion on integrating learning into constraint-based modeling. Learning constraints from examples has been surveyed by De Raedt, Passerini, et al. (2018), highlighting methods that infer constraints from data rather than requiring explicit specification. On the solving side, Popescu et al. (2022) provided an overview of ML-driven methods for constraint solving, covering constraint satisfaction, SAT solving, and answer set programming. However, their survey is heavily focused on applications, whereas our work primarily emphasizes methodological advancements. Graph neural networks, an architecture designed for learning over graph-structured data, have gained attention in combinatorial optimization. Their role in this field has been reviewed by Cappart, Chételat, et al. (2023).

Finally, algorithm configuration has been extensively surveyed in the literature, notably by Kerschke et al. (2019), Kotthoff (2016), and more recently by Schede et al. (2022). A standardized benchmark library for that was introduced by Bischl et al. (2016) and is, at the time of writing, still actively maintained.

2 Preliminaries

This section provides the formal background on constraint programming and machine learning that is used throughout the survey. For a more comprehensive introduction to these fields, we refer the reader to the *handbook of constraint programming* (Rossi et al. 2006) and the *probabilistic machine learning* books (Murphy 2012, 2023, 2022). The numerous references for the concepts introduced in this section can be found in these textbooks.

2.1 Primer on Constraint Programming

In most general terms, constraint programming is the study of computational systems based on constraints, with incursions into human-machine interfaces, declarative programming languages, and relational databases. However, the majority of current research and practical applications in CP focus on solving combinatorial problems, and in this introduction, we restrict our attention to that core aspect. Like several other approaches to combinatorial optimization, CP expresses the problem to solve through a declarative mathematical model and *guarantees that any solution it generates satisfies every requirement in that model.* Unlike most of them, its modeling primitives are rather high-level and each typically represents a common combinatorial structure that has been exploited for inference and search. It may perform some learning (e.g. nogoods and lazy clause generation; see Section 3.1.5) but its main distinction lies in deductive reasoning in the form of domain filtering algorithms.

A problem to be solved by CP is framed as a *constraint satisfaction problem* (CSP) defined as a tuple $\langle X, D, C \rangle$ where D is a finite set of values, $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of discrete variables each taking their value from a given subset of D ($x_i \in D_i \subseteq D,\ 1 \leq i \leq n$), and $C = \{c_1, c_2, \ldots, c_m\}$ is a finite set of constraints (i.e. relations) each expressed on a subset of the variables (called its scope). To solve a CSP, one must find a combination of values from the domain of each variable that simultaneously satisfies every constraint (i.e. belongs to every relation). For optimization problems, this framework is extended to a *constraint optimization problem* (COP) defined as a tuple $\langle X, D, C, f \rangle$ where the first three are as above and $f : D^n \rightarrow \mathbb{R}$ is an objective function to

optimize over all satisfying combinations of values for X. The concept of *soft constraint* refers to a constraint that does not necessarily need to be satisfied, as opposed to how we previously defined them (as *hard*). The *MaxCSP* asks to maximize the number of satisfied soft constraints.

Conceptually, one can view a CSP as a bipartite graph on variable nodes and constraint nodes with edges indicating that a variable is in a constraint's scope. Such a *constraint network* can be seen as a deterministic special case of graphical models (Koller and Friedman 2009), and in particular factor graphs where constraints play the role of factors. Continuing with this parallel, *constraint propagation*, a powerful inference mechanism that can significantly reduce the search space by propagating through the network the result of local inferences from individual constraints to their variables and back, is a form of message passing as introduced for graphical models. Initially popularized by Waltz's labeling algorithm in computer vision (Waltz 1975), demonstrating that propagation can sometimes eliminate the need for backtracking entirely, the development of propagators has since been a major focus of the CP community. A notable example is the AC-3 algorithm for enforcing arc consistency (Mackworth 1977).

The process of solving a CSP usually takes the form of backtracking tree search interleaving constraint propagation and branching at each tree node. Given the constraints and the current domains of the variables in the *model*, constraint-level *reasoning* is applied to exclude some unsatisfiable assignments. Following this, we either: *stop* if what remains identifies a particular solution; *search* through the remaining combinatorial space by partitioning it and then reapplying constraint-level reasoning on a selected part; or undo previous search decisions (i.e. *backtrack*) if no possible assignment remains. In the following sections we develop the key aspects of reasoning, modeling, and search.

2.1.1 Reasoning. The distinctive force of CP lies in its encapsulation of powerful dedicated deductive reasoning algorithms within constraints. These algorithms fuel each other through shared variables. This yields global inferences that are much stronger than the sum of their parts, and in an automated way. This is enabled by CP's other major strength: a rich set of high-level modeling primitives.

In order to characterize the result of these deductive reasoning algorithms, some levels of *local consistency* have been defined. We describe the two main ones. They are all based on the concept of support. With respect to a given constraint c, a *support* for value $d \in D_i$ being assigned to variable x_i consists of a combination of values for each other variable in c's scope that satisfies c. *Domain consistency* (also called *generalized arc consistency* for historical reasons) is achieved if the domains only contain values for which a support exists using values restricted to belong to the domain of each other variable. This is the best we can hope to achieve locally with respect to c while removing individual values from domains: any remaining domain value appears in at least one satisfying combination for c. A weaker level of consistency, but also a cheaper one computationally, is often applied to numerical constraints. It relies on a relaxation of domains when seeking a support. Let $D_i^{\mathbb{R}} = \{d \in \mathbb{R} \ : \ \min(D_i) \leq d \leq \max(D_i)\}$ be the smallest real interval containing every value in domain D_i. *Bounds consistency* is achieved if the smallest and largest values of each domain (but not necessarily values in between) have a support using values belonging to the relaxed domains $D_i^{\mathbb{R}}$ of the respective variables.

Each constraint implements one or several *domain filtering* algorithms (also called *propagators* in the literature) that, given the current state of the domains of the variables in its scope and its own semantics, remove unsupported values in domains. It may remove all such values (and achieve domain consistency), shrink the span of domains (and achieve bounds consistency), or only remove some values, thus reducing the combinatorics of the model without achieving any particular consistency level. This behavior depends heavily on the nature of the constraints and the computational cost associated with the chosen algorithm; further details are provided in Section 2.1.2. For COPs, feasibility reasoning can be complemented by optimality reasoning, known as *cost-based filtering* (Focacci et al. 1999), which removes domain values that cannot be part of any optimal solution. All these filtering algorithms often borrow ideas from other fields such as graph and network flow theory, dynamic programming, linear

programming, machine scheduling, and even computational geometry. Because they are called repeatedly at each search tree node with little change in the state between calls, care is usually taken to make these algorithms incremental.

The domain values removed by the reasoning algorithm of some constraints are communicated to the relevant variables and in turn communicated to the other constraints which have some of these variables in their scope, potentially triggering more value removals through their own domain filtering algorithms. This whole process, termed *constraint propagation*, will eventually terminate and reach a *fixpoint* since domains are finite and one cannot remove values indefinitely. As mentioned previously, this process can be viewed as an instance of message passing through the constraint network. Note that in practice, wake-up conditions may be defined for each constraint, dictating for which type of event (e.g. any domain value removal, a variable being fixed) it should be sent a message.

Nowadays, additional reasoning mechanisms such as *lazy clause generation* are also embedded into a number of CP solvers. Such solvers are built around SAT solvers that perform conflict-driven clause learning (Marques-Silva et al. 2021). Unlike machine learning, clause learning is a deductive reasoning technique whereby a solver conflict is analyzed and iteratively resolved with other clauses, such that a new clause is dynamically learned, which will prevent the solver from reaching the same conflict again.

2.1.2 Modeling. The first step in any CP model is to define a set of variables and a set of possible values for them such that any total assignment of values to these variables describes a solution to the CSP. For a COP, we add an objective variable and a constraint making it equal to the function we optimize. The next step is to define the constraints that restrict the possible assignments. While in principle a single tabular constraint over all decision variables at once would be sufficient, its size would make this intractable in general and, in a way, the CSP would already be solved since its solutions would be listed as tuples in that constraint. Instead the model is factored into several constraints in such a way that each has a well-posed semantics for which efficient specialized domain filtering algorithms can be designed to help solve the overall problem. There are often many ways to write a CP model for a given combinatorial problem and it is even sometimes beneficial to combine two or more such ways in order to improve reasoning. Constraint programming offers a rich modeling language with families of constraints, coined *global constraints*, defined for many of the usual combinatorial structures. Unfortunately, this flexibility introduces the challenge of designing an appropriate model. J.-C. Régin (2011) highlighted several common modeling pitfalls, underscoring the importance and motivation for automating parts of the modeling process. A comprehensive catalog of constraints is being maintained (Beldiceanu, Carlsson, et al. 2007) but we restrict our presentation to those that are either unavoidable or that will occur later on in this survey. We refer the reader to this catalog for details and references about these constraints or any other.

Value occurrence constraints. We start with perhaps the most important constraints, both from a historical and a practical perspective, which restrict the number of occurrences of each value being assigned to a set of variables. The ubiquitous AllDifferent restricts every value to occur at most once, i.e. variables must take distinct values. The most general member of this family is the Cardinality constraint, either representing the number of occurrences of each value as an additional variable or restricting it by specifying lower and upper bounds. Given their importance and the trade-off between the strength of the consistency level and the computational effort to achieve it, several domain filtering algorithms have been proposed, including for domain and bounds consistency.

Extensional constraints. Constraints may be given in extension as a relation of given arity. The Table constraint lists the set of allowed or forbidden tuples, which may also be given in more compact form through the use of starred entries (as in regular expressions). The MDD constraint uses a multi-valued decision diagram to represent tuples as paths from the root node to a terminal node, achieving a compact representation by sharing

subpaths between tuples. Several efficient algorithms achieving domain consistency have been proposed for such extensional constraints.

Sequencing constraints. Sometimes variables are ordered (e.g. representing a sequence of decisions over time) and one wishes to restrict the possible combinations of consecutive values taken by these variables. The SEQUENCE constraint restricts the number of occurrences of each value, much like CARDINALITY, but only inside a sliding window of a given width over the sequence of variables. The REGULAR constraint states that the respective values taken by the sequence of variables, seen as a word, should belong to the regular language described by a given automaton. Many intricate sequencing rules can thus be enforced by specifying an automaton, often in a concise way. Domain consistency is achieved on that constraint.

Numerical constraints. The CP toolbox also offers several common constraints for variables with integer domains (and sometimes domains over real numbers): linear constraints but also nonlinear ones, including those featuring trigonometric functions, absolute values, and exponentials. Typically the domain filtering algorithms applied to them achieve bounds consistency but in some cases there exist tractable algorithms achieving the stronger domain consistency.

Scheduling constraints. Resource scheduling has been a remarkably successful application area of CP. The DISJUNCTIVE and CUMULATIVE constraints model their namesake scheduling problem and implement efficient domain filtering algorithms (e.g. edge-finding, timetable, energetic reasoning), though they fall short of achieving the consistency levels previously defined.

2.1.3 Search. Since polynomial-time reasoning will rarely solve a combinatorial problem on its own, tree search is usually added. In most cases it proceeds in depth-first fashion, even for COPs, backtracking whenever the domain of a variable becomes empty and therefore prevents a complete assignment. Branching at a search tree node may consist of adding constraints on each branch in such a way that it partitions the search space, but it is almost systematically implemented as binary branching with the first branch fixing a variable to some value in its domain and the second branch removing that value from its domain. Some CP solvers select that variable and value automatically, relieving the user from the burden of writing a search procedure, but search is also programmable, allowing the user to choose among several variable-selection and value-selection heuristics, or even to program one dedicated to the problem at hand.

In the case of a COP, whenever a solution is found, thereby fixing the objective variable, a unary constraint is automatically added to require a strictly better value for that variable, triggering backtracking and continuing the search for improving solutions until none can be found, proving the optimality of the latest solution. Because domain filtering algorithms are a multiway process, domain changes for the decision variables can trigger changes in the objective variable's domain and changes in the latter, e.g. a stronger bound on the objective can impact the decision variables.

Variable-selection branching heuristics. Extensive research has been devoted to designing effective generic variable-selection heuristics, and we highlight only a few representative examples here. Some heuristics are static, such as simply selecting variables in lexicographical or chronological order, but most are dynamic by exploiting the current state of the search. An early and still widespread example of this is *smallest-domain-first* which favors variables with the fewest number of remaining assignment options. The state-of-the-art branching heuristic DOMWDEG builds on the previous idea, to which it adds the weighted degree of the variable, that is the sum of all constraints in which it is involved, each being weighted by the number of times it triggered a backtrack. That simple updated weight mechanism makes it a learned heuristic. Variable-selection heuristics are generally concerned with feasibility and not so much with optimization, with the notable exception of the

regret heuristic which favors a variable with the largest difference between the best and second-best values in its domain according to the objective function.

Value-selection branching heuristics. These have also generated interest in the research community but to a lesser degree, perhaps underestimating their importance. For generic heuristics, proceeding in simple lexicographic order or randomly is common. Some heuristics, such as *impact-based search* and *activity-based search*, can typically select both a variable and an appropriate value simultaneously. Dedicated heuristics may also be designed that exploit a user's knowledge of the problem.

Beyond depth-first search. Given a trustworthy value-selection heuristic, *limited discrepancy search* is a good alternative to depth-first search: leaves of the search tree are visited in order of increasing number of discrepancies, where a discrepancy in a path to a leaf corresponds to not following the recommendation of the heuristic. This approach prioritizes the search to follow the heuristic as much as possible. Given a branching heuristic that learns during search or one that involves some randomness, restarting the search periodically has been shown to increase robustness by avoiding remaining stuck deep in the search tree. *Large neighborhood search* (LNS) is conceptually a local search method that explores a neighborhood through CP tree search. Operationally at each iteration some fraction of the variables in the CP model are fixed to their value in the current solution, defining the neighborhood to search over.

2.2 Primer on Machine Learning

The primary objective of machine learning is to improve the performance of a given task by learning from historical data and optimizing a specific performance metric. In this section, we provide an overview of the core methodologies and concepts in ML that are essential for understanding the subsequent sections of this paper.

2.2.1 *Learning Paradigms.*

We first introduce the main paradigms of ML that are needed for this survey.

Supervised learning. Given a finite training set of input-output pairs, supervised learning aims to learn a mapping from inputs to outputs. Formally, given N data and a labeled dataset $E = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i is a feature vector from the input space $\mathcal{X}$ and y_i is a label from the output space $\mathcal{Y}$, the goal of supervised learning is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that maps inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. The function f is typically parameterized by a set of parameters θ, and the learning process involves finding the optimal parameters θ^* that minimize a loss function $\mathcal{L}(y, f_\theta(x))$ over the dataset. In *classification* tasks, the output space $\mathcal{Y}$ consists of discrete labels. The objective is to assign each input $x \in \mathcal{X}$ to one of the predefined classes in $\mathcal{Y}$. For example, in a binary classification problem, $\mathcal{Y} = \{0, 1\}$, the task is to determine which of the two classes an input belongs to. A special case of (binary) classification is *concept learning*, a symbolic learning approach where the goal is to infer a hypothesis $h : \mathcal{X} \to \{0, 1\}$ from a hypothesis space $\mathcal{H}$, that separates positive and negative examples of a concept. In *regression* tasks, the output space $\mathcal{Y}$ is continuous. The goal is to predict a continuous value for each input $x \in \mathcal{X}$. Such supervised learning paradigms differ in the nature of the output space and the type of loss function used. Supervised learning methods can be further categorized into *passive* and *active*. In *passive* (standard) learning methods, the finite labeled dataset E is given, and the learner has no influence over the collected samples. In contrast, in *active* learning, the learner interacts with an oracle $O : \mathcal{X} \to \mathcal{Y}$, actively selecting unlabeled samples $x' \in \mathcal{X} \setminus E$ to query for labels. This results in an iteratively growing training set $E \leftarrow E \cup \{(x', O(x'))\}$.

Unsupervised learning. The objective is to discover patterns in data without explicit output labels. The goal is to learn a function $f : \mathcal{X} \to \mathcal{Z}$ that maps inputs $x \in \mathcal{X}$ to a latent space $\mathcal{Z}$, based on an unlabeled dataset $E = \{x_i\}_{i=1}^{N}$. The function f is typically parameterized by a set of parameters θ, and minimizes an unsupervised loss function $\mathcal{L}(f_\theta(x))$, that captures the structure of the data, such as clustering or dimensionality reduction.

Reinforcement learning (RL). Like unsupervised learning, RL does not use labeled data. It involves an agent interacting with an environment by executing actions, where the goal is to learn a policy that maximizes the cumulative reward. Formally, at a given time step t, the agent observes a state $s_t \in \mathcal{S}$, with $\mathcal{S}$ denoting the state space, and takes actions $a_t \in \mathcal{A}$, with $\mathcal{A}$ denoting the action space. An RL problem can be modeled as a *Markov decision process*, defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R)$, with T being the transition function and R the reward function. The transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ specifies the probability $T(s, a, s')$ of transitioning from state s to state s' when action a is taken, thus satisfying $\sum_{s' \in \mathcal{S}} T(s, a, s') = 1$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ assigns a numerical reward for each action taken in a given state. The environment of the RL agent can be either *deterministic* or *stochastic*. In a deterministic environment, the transition function satisfies $T(s, a, s') \in \{0, 1\}$, i.e. a state-action pair (s, a) uniquely determines the next state s'. In a stochastic environment, transitions are probabilistic, with $T(s, a, s') \in [0, 1]$, resulting in multiple possible successor states given a state-action pair. The agent acts according to a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which indicates the probability of selecting action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. The goal is to learn a policy that maximizes the expected cumulative reward $\mathcal{R} = \sum_{t=0}^{T} R_t(s_t, a_t)$, where R_t is the reward at time t. The cumulative reward can be discounted by a factor $\gamma \in [0, 1]$, i.e. $\mathcal{R} = \sum_{t=0}^{T} \gamma^t R(s_t, a_t)$.

Imitation learning. Imitation learning aims to learn a policy directly from demonstrations provided by an expert, rather than relying on explicit reward functions like RL. Specifically, imitation learning leverages a dataset $E = \{\tau_i\}_{i=1}^{N}$ of expert trajectories $\tau_i = \{(s_t^{(i)}, a_t^{(i)})\}_{t=0}^{T_i}$ to guide learning, where T_i is the length of the i-th trajectory. Formally, the objective is typically to minimize the discrepancy between the agent's learned policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ and the expert's policy π_E. Like supervised learning, imitation learning methods can be passive or active.

2.2.2 Learning Algorithms. We now provide a formal overview of several prominent algorithms that are applied within these learning paradigms.

Decision trees. Decision trees are hierarchical models represented as directed acyclic graphs, commonly employed for supervised classification and regression problems. Formally, given training data $E = \{(x_i, y_i)\}_{i=1}^{N}$ with inputs $x_i \in \mathcal{X}$ and associated outputs $y_i \in \mathcal{Y}$, a decision tree recursively partitions the input space $\mathcal{X}$ into mutually exclusive regions through feature-based decision rules. Each internal node represents a decision on an attribute of the input $x \in \mathcal{X}$, while each leaf node specifies a predicted output $y \in \mathcal{Y}$. During the learning process, each internal node j selects the splitting rule that *locally* maximizes a suitable splitting criterion. Common splitting criteria include information gain (based on entropy reduction), gain ratio, Gini impurity, or variance reduction (for regression). In contrast to these methods, few approaches have been proposed to build optimal decision trees according to various criteria, e.g. maximizing the accuracy given a fixed depth. An advantage of decision trees is that they are naturally interpretable.

Decision rules. Decision rules are interpretable supervised learning models that explicitly represent the learned function $f_\theta : \mathcal{X} \to \mathcal{Y}$ as a set of logical IF-THEN statements. Unlike decision trees, which organize rules hierarchically, decision rule models maintain a flat structure, resolving potential conflicts among rules through metrics such as accuracy, coverage, or rule length. Decision rules can be learned directly from data using rule induction algorithms (Grzymala-Busse 2023) or indirectly extracted from other models (Gilpin et al. 2018), such as decision trees or ensemble methods.

Version spaces. Version spaces represent a symbolic supervised learning approach for the task of concept learning. Given a hypothesis space $\mathcal{H}$ consisting of hypotheses $h : \mathcal{X} \to \{0, 1\}$ and a labeled dataset $E = \{(x_i, y_i)\}_{i=1}^{N}$, the version space $VS_{\mathcal{H},E}$ is the subset of hypotheses consistent with all examples:

$$VS_{\mathcal{H},E} = \{h \in \mathcal{H} \mid h(x_i) = y_i, \forall (x_i, y_i) \in E\}.$$

The version space is typically represented compactly by two boundary sets: the set of most specific hypotheses S_H and the set of most general hypotheses G_H. These boundary sets are defined with respect to a partial order of generality, where for two hypotheses $h_1, h_2 \in \mathcal{H}$, we say h_1 is at least as general as h_2 (denoted $h_1 \geq h_2$) if and only if $\forall x \in \mathcal{X} : h_2(x) = 1 \implies h_1(x) = 1$. Given the dataset E, learning incrementally updates the boundary sets S_H and G_H by removing hypotheses inconsistent with observed examples, progressively narrowing the version space.

Ensemble methods. Ensemble learning combines several base learners, i.e. a set $\{f_1(x), f_2(x), \ldots, f_m(x)\}$, into a stronger aggregated learner $f_{\text{ensemble}}(x)$. Formally, given m trained base learners, an ensemble produces predictions by applying an aggregation operator Φ:

$$f_{\text{ensemble}}(x) = \Phi\big(f_1(x), f_2(x), \ldots, f_m(x)\big).$$

Common operators Φ include weighted voting, averaging predictions or stacking.

Neural networks (NNs). Neural networks are computational models consisting of interconnected layers of nodes referred to as *neurons*. Neural networks are capable of approximating complex nonlinear functions. Formally, a feedforward neural network parameterized by θ consists of L layers, each computing a transformation:

$$h^{(l)} = \sigma^{(l)}\left(w^{(l)}h^{(l-1)} + b^{(l)}\right), \quad l = \{1, \ldots, L\},$$

where $h^{(0)} = x$ is the input to the model, $w^{(l)}$ and $b^{(l)}$ are learnable parameters representing weights and biases at layer l, respectively, and $\sigma^{(l)}(\cdot)$ is a nonlinear activation function (e.g. ReLU, sigmoid, or tanh). The output is given by $f_\theta(x) = h^{(L)}$. Learning involves optimizing the parameters $\theta = \{w^{(l)}, b^{(l)}\}_{l=1}^{L}$ to minimize a specified loss function via gradient-based optimization algorithms such as stochastic gradient descent, Adam, or RMSProp.

2.2.3 *Common Neural Architectures.*

We now focus more on neural architectures commonly used in the literature.

Multilayer perceptrons (MLPs). Multilayer perceptrons are *fully connected* feedforward neural networks consisting of multiple layers of neurons. They are a special case of the general neural network definition provided earlier, where each layer is fully connected.

Recurrent neural networks (RNNs). Recurrent neural networks extend the neural network definition to sequential data by introducing recurrent connections. Formally, given an input sequence $x = (x_1, x_2, \ldots, x_T)$, an RNN computes hidden states h_t recursively as:

$$h_t = \sigma\left(w_{xh}x_t + w_{hh}h_{t-1} + b_h\right), \quad t = \{1, \ldots, T\},$$

where w_{xh}, w_{hh}, b_h are learnable parameters, and $\sigma(\cdot)$ is a nonlinear activation function.

Graph neural networks (GNNs). Graph neural networks generalize neural network architectures to graph-structured data. Formally, given a labeled graph $G = (V, E)$, where each node $v \in V$ is associated with an initial feature vector $x_v \in \mathcal{X}$, a GNN iteratively updates node representations by applying a neighborhood aggregation operator:

$$h_v^{(l)} = f_{merge}^{(l)}\left(w_1^{(l)} \cdot h_v^{(l-1)}, f_{aggr}\left(\{w_2^{(l)} \cdot h_u^{(l-1)} : u \in \mathcal{N}(v)\}\right)\right), \quad l = \{1, \ldots, L\},$$

with initial node embeddings set as $h_v^{(0)} = x_v$. The biases have been omitted for the clarity of the representation. Here, $\mathcal{N}(v)$ denotes the neighborhood of node v, $\{w_1^{(l)}, w_2^{(l)}\}$ are learnable parameters at layer l, $f_{aggr}(\cdot)$ is a permutation-invariant aggregation function (e.g. mean, sum, or max pooling), and $f_{merge}(\cdot)$ is a merging operator. This architecture is widely used for node classification, link prediction, and graph classification tasks.

Transformers. Transformers are neural architectures based on self-attention, designed to efficiently capture long-range dependencies. They have achieved state-of-the-art performance in various tasks, including machine translation, text generation, and other natural language processing applications. A transformer typically consists of stacked encoder and/or decoder layers. Given an input sequence $X \in \mathbb{R}^{T \times d}$, where T is the sequence length (e.g. the number of tokens) and d is the dimensionality of the token embeddings, each encoder layer computes multi-head self-attention:

$$MultiHead(X) = \textsc{Concat}(head_1, \ldots, head_H) \cdot W_o,$$

where W_o is a matrix of learnable parameters and $\textsc{Concat}(.)$ denotes concatenation of the H attention heads along the feature dimension. Each attention head is defined as:

$$head_h = \textsc{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}}\right) \cdot V_h,$$

with $Q_h \in \mathbb{R}^{d_k}$, $K_h \in \mathbb{R}^{d_k}$, and $V_h \in \mathbb{R}^{d_v}$ are queries, keys, and values respectively, obtained by linear projections of the input for head h. Here, d_k represents the dimensionality of queries and keys, while d_v is the dimensionality of values (often chosen so that $d_v = d_k$). The decoder layers similarly apply multi-head self-attention to previously generated outputs, followed by cross-attention over the encoder's output representations. Each encoder and decoder layer also includes a position-wise feedforward neural network applied independently at each position. Transformer-based models come in several variants: encoder-only, optimized for understanding tasks such as classification, decoder-only, optimized for generative tasks such as text completion or dialogue, and encoder-decoder, combining understanding and generation, often used for sequence-to-sequence tasks like translation or summarization.

Generative models. Generative models aim to learn the underlying probability distribution of observed data in order to enable the generation of new samples that resemble the original data. Formally, given an unlabeled dataset $E = \{x_i\}_{i=1}^N$ with samples $x_i \in \mathcal{X}$ drawn independently from an unknown data distribution $p_{\text{data}}(x)$, a generative model learns a parameterized distribution $p_\theta(x)$ that approximates $p_{\text{data}}(x)$. The learning objective typically involves minimizing a divergence measure $\mathcal{D}_v$ between the learned distribution $p_\theta(x)$ and the true data distribution $p_{\text{data}}(x)$:

$$\theta^* = \arg\min_\theta \mathcal{D}_v\big(p_{\text{data}}(x), p_\theta(x)\big).$$

After training, the generative model can produce new synthetic samples $x' \sim p_{\theta^*}(x)$ that closely resemble samples from the original data distribution.

Large language models (LLMs). Based on the transformer architecture, LLMs are deep learning systems with billions of parameters, capable of learning and generating complex language patterns. For the purposes of this paper, we consider them as black-box functions that are configured by an input text p, called *prompt*, and a randomness parameter τ, namely the *temperature*. Formally:

$$LLM(p, \tau) = (x_1, x_2, \ldots, x_w), \quad \text{where } x_{t+1} \sim P_\tau(x_{t+1} \mid p, x_{\leq t})$$

Here, $x_{\leq t}$ denotes the sequence of tokens generated up to time step t, with $x_{\leq 0}$ as the empty sequence and x_1 as the first generated token. At each step, the next token x_{t+1} is sampled from the temperature-adjusted probability distribution $P_\tau(x_{t+1} \mid p, x_{\leq t})$, which is typically computed by applying the softmax function to the model's logits, i.e. raw unnormalized scores over the vocabulary. The temperature τ scales the logits before softmax, with higher values increasing randomness and diversity, while lower values favor the most likely tokens. The generation process continues until a predefined stop token is encountered or the maximum context length is reached. We assume tokenization and detokenization into text are handled internally by the LLM.

3 Current Progress in Combining CP and ML

As previously discussed, CP and ML can interact in two main ways: either ML is used to enhance components of the CP pipeline, or CP is used to enforce structure to ML approaches. The purpose of this section is to survey these two modes of interaction. First, Section 3.1 covers approaches that integrate machine learning techniques into constraint programming frameworks, either during the modeling or solving phases. Second, Section 3.2 discusses how constraint programming's structured reasoning can help address limitations inherent to machine learning methods, such as the lack of guarantees, reasoning capabilities, or interpretability.

3.1 Improving CP with ML

Section 2.1 introduced the fundamental building blocks of the CP pipeline (i.e. *modeling*, *search*, and *reasoning*). Notably, each of these components has benefited from integration with machine learning techniques. This section aims to comprehensively summarize these improvements. Figure 1 provides a high-level illustration of this pipeline, along with pointers to relevant subsections. Additionally, we included an *automatic configuration* step in the pipeline. While this step is not exclusive to CP, several studies have used it within a CP context, making it relevant to include in this survey. Finally, we close with a subsection on application-specific approaches as well as works investigating generic ML representations for CP.

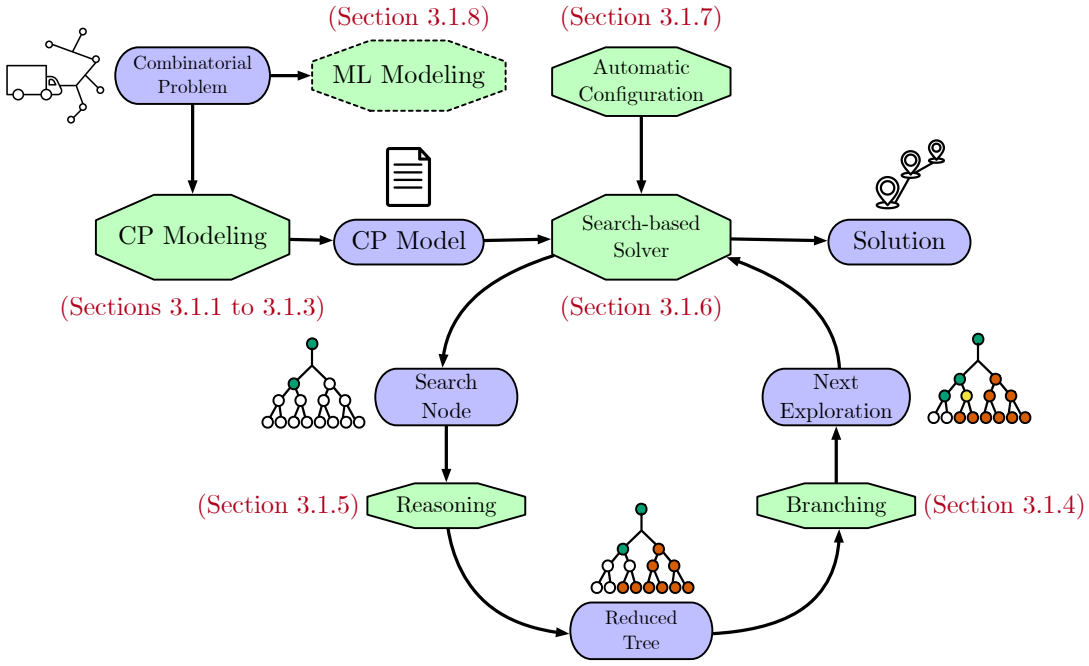

Fig. 1. Overview of a complete CP pipeline, highlighting the specific points where machine learning techniques can be integrated. The green boxes represent individual tasks within the pipeline, while the blue boxes indicate the elements used as input or produced as output. The dashed box corresponds to tasks whose results can be reused in downstream components. Each subsection below details a particular integration point. The CP modeling step is detailed across three subsections and figures, focusing respectively on learning: constraints (Figure 2), objective functions (Figure 3), and entire models (Figure 4).

3.1.1 Learning to Model - Learning Constraints (Figure 2). Modeling the constraints of a given problem is not straightforward and is considered an important drawback in the wider use of CP. As a result, learning the constraints of the problem in order to assist the user in the modeling process has received a lot of attention in recent years. In the literature, this research area is called *constraint acquisition* (CA) (Bessiere, Koriche, et al. 2017), with the goal of learning the constraints of a given problem from data. The data used is most commonly examples of solutions and non-solutions to the problem, i.e. assignments to the variables of the problem together with a positive or negative label. Such data can either be pre-existing, using past solutions or non-solutions of the problem, or generated through an interaction with the user. This separates the two main families of methods in CA, *passive* and *active* (typically called *interactive*) acquisition. Recent realistic applications of CA show its significant potential (Barral et al. 2024; Menguy, Bardin, Gotlieb, et al. 2025). Besides acquiring the complete set of constraints, CA can be used to complete ambiguous or incomplete specifications of non-expert users, as highlighted by De Raedt, Passerini, et al. (2018) and Bessiere, Carbonnel, Dries, et al. (2023).

Note that constraint acquisition differs from the usage of the term *constraint learning* (or *clause learning*) in the solving process, where it typically refers to the process of dynamically adding or adapting constraints to the model to enhance solving efficiency. This is later covered in Section 3.1.5.

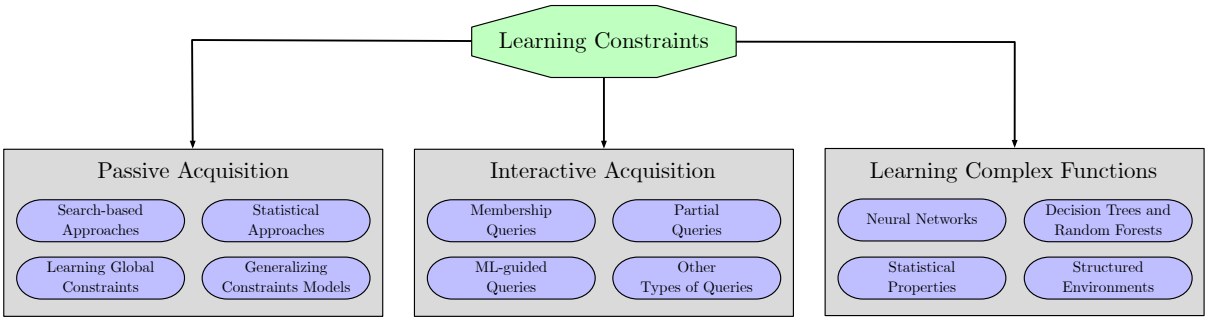

Fig. 2. Overview of Section 3.1.1 - *learning constraints.*

The main assumption in CA is that the user has formulated the variables and domains of the problem and thus are given as input to the system. Then, the task of acquiring the constraints is typically formulated as a *concept learning* task, using approaches originating from *version space learning*. The learning system is given a *language* of constraint relations that may appear in the problem. This language is used to generate the set of candidate constraints for the problem, which is often called the *constraint bias*. During the acquisition process, the system stores two sets of constraints to represent the most specific and most general hypotheses in this version space approach: the set of remaining candidates in the bias B and the set of learned constraints C_L. During CA, constraints are added to C_L and removed from B, incrementally updating the version space. We now focus on methods for the two families of CA, i.e. passive and interactive constraint acquisition.

Passive constraint acquisition. In this setting, a dataset of existing solutions and/or non-solutions to the problem is provided by the user, as in standard passive ML. Several approaches have been proposed, differing in the types of constraints they aim to learn, their robustness to noise or inconsistencies in the data, and their ability to generalize beyond the observed examples. They are as follows:

(1) *Search-based approaches:* one of the earliest approaches to passive CA is Conacq (Bessiere, Coletta, Koriche, et al. 2005; Bessiere, Koriche, et al. 2017), a SAT-based version space algorithm that incrementally narrows down the set of consistent candidate constraints based on the given examples in the dataset. A similar

idea, using a CP-based model of the version space, is presented by Coulombe and Quimper (2022). Later, Bessiere, Carbonnel, and Himeur (2023) proposed a system that aims to tackle a more difficult problem: computing a suitable constraint language (in an extensional form), as well as a consistent set of constraints using this language. To achieve that, they model the acquisition problem as a partial *MaxSAT* problem.

(2) *Statistical approaches:* a key limitation of the previous approaches is their inability to handle noise in the training data, in contrast to modern ML techniques. Recent work consequently explored tighter connections with such statistical approaches. Among them, ClassAcq (S. D. Prestwich et al. 2021) is a paradigm that trains a classifier to distinguish between solutions and non-solutions, and then derives a constraint model from the trained classifier. This approach was successfully demonstrated using a naive Bayes classifier in the BayesAcq system (S. D. Prestwich et al. 2021). S. Prestwich (2020) and S. Prestwich and Wilson (2024) later proposed another statistical approach based on sequential analysis. Specifically, they apply the sequential probability ratio test to evaluate candidate constraints during the acquisition process. Wiegand et al. (2024) formalized the constraint acquisition problem using the *minimum description length principle*, selecting the model with the best lossless compression of the data. Acknowledging that collecting labeled data can be both labor-intensive and error-prone, an approach based on unsupervised learning was proposed by S. Prestwich (2021), with the MineAcq system, which extracts constraints using statistical measures from association rule mining.

(3) *Learning global constraints:* all previous methods focus on learning fixed-arity constraints and are not equipped to handle global constraints, primarily due to the exponential growth in the number of candidate constraints. One of the first CA methods to target global constraints was the ModelSeeker system (Beldiceanu and Simonis 2011, 2012, 2016), along with a similar system proposed by Balafas, D. C. Tsouros, et al. (2024). These systems rearrange the variables as a matrix structure in various ways and apply predefined patterns that capture where global constraints from a given language might appear in the target constraint set. In a different direction, Picard-Cantin et al. (2016, 2017) and Jung and J.-C. Régin (2021) aim to learn the parameters of global constraints, assuming that we already know their scope. Picard-Cantin et al. (2016, 2017) propose a branch and bound method to learn such parameters, while Jung and J.-C. Régin (2021) focus on global constraints encoded as MDD.

(4) *Generalizing constraint models:* while most constraint acquisition methods focus on learning instance-specific constraint models, a few works aim to learn interpretable, parameterized forms of constraints *during* the acquisition process. To achieve that, these approaches are provided with examples from multiple instances of a problem, each annotated with corresponding instance parameters. Lallouet et al. (2010) proposed an *inductive logic programming* approach, to learn rules of the form $condition \Rightarrow constraint$, enabling generalization across problem instances. On the other hand, Count-CP (Kumar et al. 2022) first learns instance-specific constraints in the form of linear expressions. It then applies a pattern-matching procedure to group these constraints and derive *first-order constraints* that generalize across unseen instances. Another approach to generalizing constraint models is to decouple the acquisition phase from the generalization step, by first acquiring instance-specific constraints (using any method) and then generalizing them to new instances based on their parameters. S. Prestwich (2022) uses genetic programming to achieve this generalization; however, it often learns contrived and non-interpretable generalization rules. In contrast, GenCon (D. Tsouros, Berden, S. Prestwich, et al. 2025) aims to learn interpretable constraint specifications, acquiring them from decision rules extracted from ML classifiers.

An intrinsic limitation of passive CA is its reliance on a sufficiently large and diverse set of examples, often including both solutions and non-solutions, in order to converge. This is an assumption that may not hold in many practical scenarios. Balafas, D. Tsouros, et al. (2024) showed how the diversity of the provided examples

affects the quality of the acquired model. They also proposed using ML regression models to estimate the quality of the learned constraint set based on characteristics of the training data.

Interactive constraint acquisition. In contrast to its passive counterpart, the system here interacts with the user during the acquisition process to learn the constraints of the problem. This corresponds to the *active learning* setting, where an oracle, often a human user, is queried by the system, usually to provide labels for unlabeled examples. Specifically, interactive CA is a special case of query-directed learning known as exact learning (Bshouty 1995; Bshouty et al. 1994). Different families of queries can be posed during the acquisition process, and different methods can be used to generate them, as follows:

(1) *Membership queries:* introduced by Angluin (1988), these queries ask the user to classify a given example, i.e. an assignment to the variables, as either a solution or a non-solution. Conacq.2 (Bessiere, Coletta, O'Sullivan, et al. 2007; Bessiere, Koriche, et al. 2017) was the first active constraint acquisition method based solely on membership queries. It extends ConAcq to operate in an interactive setting. However, it was later shown by Bessiere, Koriche, et al. (2017) that constraint networks are not learnable with a polynomial number of membership queries, highlighting a fundamental limitation of relying solely on this query type.
(2) *Partial queries:* to overcome the limitations of membership queries, Bessiere, Coletta, Hebrard, et al. (2013) introduced partial queries, which ask the user to classify *incomplete variable assignments*. This approach led to the development of the QuAcq algorithm, which improves learning efficiency, guaranteeing learnability of sets of constraints in a polynomial number of (partial) queries. This advancement directly enabled a new direction in interactive constraint acquisition research, giving rise to a family of algorithms and methods aimed at further improving the efficiency of the process. Several works (Arcangioli et al. 2016; Bessiere, Carbonnel, Dries, et al. 2023; Bessiere, Daoudi, et al. 2016; D. C. Tsouros, Stergiou, and Bessiere 2019; D. C. Tsouros, Stergiou, and Sarigiannidis 2018) proposed variations of the QuAcq algorithm, primarily focused on reducing the number of queries and minimizing interaction time. In parallel, Lazaar (2021) introduced an approach that allows multiple users to participate concurrently in the interaction process, opening new possibilities for collaborative constraint acquisition. Later, D. C. Tsouros, Berden, et al. (2023) proposed a meta-algorithm called GrowAcq, which adopts a bottom-up procedure within the interactive CA framework. This approach significantly improves efficiency by allowing the use of larger sets of candidate constraints while accelerating the overall acquisition process. Finally, several works have focused on the efficient generation of informative queries for interactive CA, aiming to minimize query generation times while maintaining convergence guarantees (Addi, Bessiere, et al. 2018; Addi and Ezzahir 2019; Bessiere, Carbonnel, Dries, et al. 2023; D. C. Tsouros, Berden, et al. 2023; D. C. Tsouros and Stergiou 2020).
(3) *ML-guided queries:* building on improvements in query generation efficiency, later research shifted focus toward generating more informative queries. D. C. Tsouros, Berden, et al. (2023) introduced a way to guide query generation using probabilistic estimates on how likely each candidate constraint is to belong to the target model. These estimates are derived using a frequentist counting approach, allowing the system to prioritize queries that are expected to yield the most useful information. Extending this work and tightly connecting interactive CA with statistical ML for the first time, D. Tsouros, Berden, and Guns (2024) showed how ML probabilistic classifiers can be used to provide the probabilities for each constraint. Specifically, an ML classifier is used at the individual constraint level using a dataset E_C, which consists of constraints as training examples, each labeled to indicate whether the constraint c belongs to the target set $c \in C_T$ or not. This dataset is incrementally updated during the acquisition process. D. Tsouros, Berden, and Guns (2024) proposed a feature representation for constraints, which was further extended by D. Tsouros, Berden, S. Prestwich, et al. (2025). This representation leverages relation-based and scope-based information characteristics that can be extracted directly from the constraints.

(4) *Other types of queries:* in addition to (partial) membership queries, several alternative forms of queries have been introduced in the interactive constraint acquisition literature to provide greater flexibility in user interaction. D. C. Tsouros, Stergiou, and Bessiere (2020) introduced the *limited membership query*, allowing omissions in the user's answers. More expressive query types have also been introduced, with the goal of reducing the number of queries needed to accurately learn the target constraint set. Daoudi, Mechqrane, et al. (2016) introduced the *recommendation query*, which asks the user directly whether a specific constraint belongs to the target constraint model. Techniques from link prediction (Daoudi, Mechqrane, et al. 2016) or ML classification (Islah and Mechqrane 2024) have been used for this purpose. Bessiere, Coletta, Daoudi, et al. (2014) introduced the *generalization query*, which asks the user whether a pattern of constraints, rather than a single constraint, belongs to the target constraint model. Daoudi, Lazaar, et al. (2015) showed how to extract variable types during the acquisition process to help generalization, using information derived from the learned constraints. More recently, LLMAcq was introduced as a novel approach that combines interactive CA with LLMs (Mechqrane et al. 2024). This approach allows the user to give natural language feedback to the system on why they rejected an example, after which an LLM is used to extract potential constraints from this feedback.

We also note that the framework of interactive constraint acquisition has been extended in several directions to support the acquisition of different types of constraints and problem settings. This includes the acquisition of disjunctive constraints (Menguy, Bardin, Lazaar, et al. 2023), as well as the learning of qualitative constraints through the use of qualitative queries (Belaid et al. 2022, 2024; Mouhoub et al. 2023). Finally, D. C. Tsouros and Stergiou (2021) extended the framework to handle soft constraints for *MaxCSPs* by introducing partial preference queries.

Learning and constraining complex functions. A different direction to learning constraints than CA is *empirical model learning* (Lombardi, Milano, and Bartolini 2017), which aims to learn complex, nonlinear, functions through ML models, and then integrate them into CP models. This is often needed to approximate elements that are hard to model explicitly, especially in cases where the underlying physics is not fully known. While CA learns symbolic constraints, empirical model learning aims to embed functions learned through data-driven ML components, such as neural networks or decision trees, into CP models. In this paradigm, ML models are trained offline to approximate a desired function, and then compiled into CP-compatible forms via global constraints or specialized propagators, allowing the solver to natively reason with the learned behavior. Hence, the main challenge is how to embed such learned functions into CP models. This has been done mainly for the following classes of classifiers:

(1) *Neural networks:* Bartolini et al. (2011) introduced a new global constraint designed to capture individual neurons of neural networks. Then, the integration of a trained neural network into a CP model happens by introducing a *neuron constraint* for each node. However, this decomposition approach sometimes leads to weak bounds during propagation. To overcome this, Lombardi and Gualandi (2016) proposed a network-level propagator based on a non-linear Lagrangian relaxation.
(2) *Decision trees and random forests:* Bonfietti et al. (2015) proposed three CP-friendly encodings to handle tree-based classifiers as follows: (1) Boolean meta-constraints that mirror tree-branching logic, interpreting each path from root to leaf as a logical implication; (2) discretization combined with a global TABLE constraint representing leaf assignments; and (3) conversion of trees into compact multi-valued decision diagrams, with custom MDD propagators for efficient inference.

Beyond classifiers, dedicated global constraints have also been proposed to enforce statistical properties of candidate solutions. Notable examples include the SPREAD constraint (Pesant and J.-C. Régin 2005; Schaus and J. Régin 2014), which ensures that a collection of values satisfies specified statistics (e.g. arithmetic mean, standard

deviation, or median), the DEVIATION constraint (Schaus, Deville, and Dupont 2007; Schaus, Deville, Dupont, and J.-C. Régin 2007), which also enforces balance among a set of variables but relies on the Manhattan norm rather than the Euclidean one, and the DISPERSION constraint (Pesant 2015) which generalizes the previous two by considering a general L_p norm. In addition, global constraints can be used to encapsulate environments amenable to learning, such as the MARKOV constraint (Pachet and Roy 2011), which allows the control of Markov sequences, or the (COST)-REGULAR constraint (Demassey et al. 2006; Pesant 2004), which constrains sequences of variables to match automata, potentially with costs on transitions. Such modeling approaches are particularly relevant in reinforcement learning, where structured environments are ubiquitous.

3.1.2 Learning to Model - Learning the Objective Function (Figure 3). While the previous section focuses on learning constraints from data, we now turn to work learning the objective function from data. As in constraint acquisition, we can separate two cases: *passive learning*, where we assume the data consists of historically preferred solutions that must be reconstructed; and *(inter)active learning*, where the user can be asked about their preference regarding candidate solutions. In both cases, the data represents solutions and the goal is to learn an objective that represents user preferences.

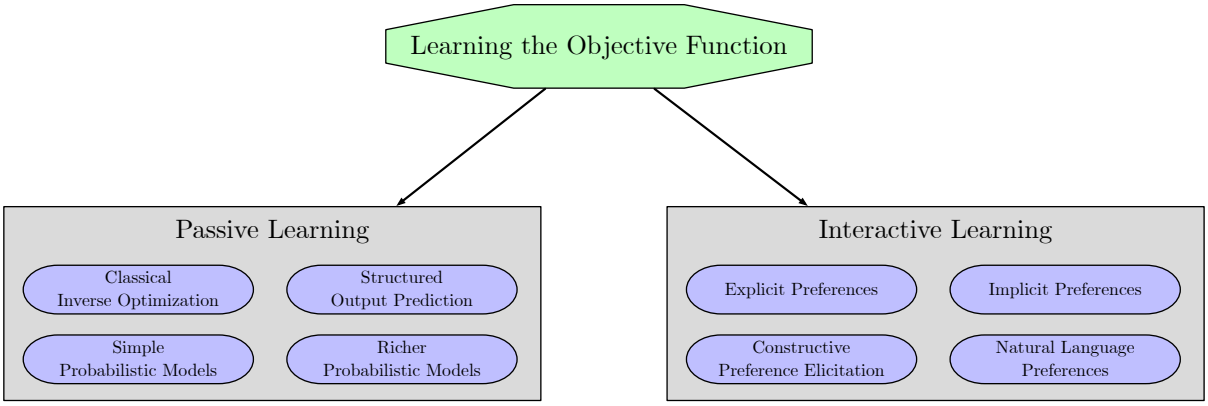

Fig. 3. Overview of Section 3.1.2 - *learning the objective function.*

There is another related family of work on learning (objective) functions and integrating them into constraint models, namely where the data represents uncertain external processes like electricity price prediction or demand estimation. The main goal here is not to capture user preferences, but to handle the uncertainty of the external processes. Such decision-focused methods either rely on a primal-dual solver, or are agnostic to the solver technology used and hence not CP specific; we refer to Mandi, Kotary, et al. (2024) for a recent survey when the values to learn are in the objective.

Passive learning. In this setting, we assume access to a dataset of historical solutions $\{(C', x')\}$, where C' is a parametrization of the known constraint satisfaction problem $\langle X, D, C\rangle$ and x' is a feasible assignment to the decision variables X. The goal is to learn an unknown objective function f that would yield high quality solutions like the given x'. Different mechanisms have been designed for this purpose.

(1) *Classical inverse optimization:* this method often assumes a linear objective function $\theta \cdot X$, that a prior weight estimate $\hat{\theta}$ is given and that one has to learn the weights θ to make a single given solution x^* optimal (Chan et al. 2025). This leads to a bi-level optimisation formulation that can be solved under different assumptions on the constraint set C.

(2) *Structured output prediction:* without placing assumptions on the constraint set, as is typical in CP, the problem can be re-interpreted as a structured output prediction problem (Bakir et al. 2007). Following the structured perceptron approach of Collins (2002), this corresponds to an online learning method: starting from an initial (random) weight estimate θ, one can iteratively compute the solution x maximizing $\theta \cdot X$; if $\theta \cdot x > \theta \cdot x^*$ a weight update can be performed on θ based on the difference $(x - x^*)$. This also works for multiple, possibly noisy, historic solutions $\{(C', x^*)\}$ as well as in a multi-objective optimisation setting where one learns a linear weighting over subobjectives $\phi(X)$ rather than over the decision variables X directly. In case of contextual features or neural network models, one can use a learning-to-rank loss (Mandi, Bucarey, et al. 2022) or select other methods from the decision-focused learning literature (Mandi, Kotary, et al. 2024). However, these methods require calling the CP solver in every iteration, which can be time consuming due to the computational complexity of the CP problem. Véjar et al. (2024) investigated ways to scale up learning with the preference perceptron through three different techniques: heuristic solving, only searching for an improving rather than optimal solution in each iteration, or caching and reusing feasible solutions.

(3) *Simple probabilistic models:* the above methods rely on a strong assumption: that the historic solutions $\{(C', x')\}$ are optimal or near-optimal, i.e. that the process generating these solutions x' has considered all possible solutions and chosen the best from it. When dealing with historic hand-crafted, or semi-hand-crafted, solutions this will not be the case. Instead of framing the problem as trying to learn a function that maximizes the given solutions by discriminating it from other solutions, one can treat it as a *generative* learning problem. In this settings, there is an unknown probability distribution $P(x'|C')$ from which x' was sampled, and we wish to learn that distribution. This raises two questions: how to *learn* $P(x'|C')$ and how to *optimize*:

$$\max_X P(X|C') \text{ s.t. } X \in D,\ C'(X) \text{ is true.}$$

In the context of vehicle routing, Canoy and Guns (2019) take a first-order Markov approximation $P(X) \approx P(X_0) \times P(X_1|X_0) \times \cdots \times P(X_n|X_{n-1})$ with X_0 the depot and $\langle X_1, \ldots, X'_n \rangle$ the sequence of stops visited. The advantage of this factorization is that we know that $P(X_0) = 1$, e.g. that we start in the depot. Let us now represent $\theta_{ij} = \log(P(X_j|X_i))$. We then have the simplification:

$$\max_X P(X_0) \times P(X_1|X_0) \times \cdots \times P(X_n|X_{n-1}) = \max_X \textstyle\sum_{ij} \theta_{ij} X_{ij}$$

where X_{ij} are Boolean decision variables indicating that the route goes from stop i to stop j. Hence the factorization combined with the log-probability trick leads to a linear objective function that is natural to optimize for any constraint solver. As to estimating the probabilities $P(X_1|X_0)$, a number of methods have been investigated including counting the observed sequence in combination with Laplace smoothing (Canoy and Guns 2019), maximum likelihood estimation of a neural network (Mandi, Canoy, et al. 2021) and learning to mix the a learned objective with other given sub-objectives such as distance, using structured output prediction (Canoy, Bucarey, et al. 2023).

(4) *Richer probabilistic models:* a generalization of the above approach is to consider learning other, richer types of probabilistic models that represent $P(x'|C')$. Brouard et al. (2020) proposed learning a pairwise graphical model $P(X|C') \approx \prod_{ij}^{X} \exp(F(X_i, X_j))$ where $F(X_i, X_j)$ is a cost function between any two decision variables, representable as a table of costs between any two assignments of these variables. This pairwise factorization captures well known graphical models such as *Markov random fields*. Furthermore, using the same log-probability trick, this pairwise graphical model can be reformulated into a pairwise cost-function network, which corresponds to a *weighted constraint satisfaction problem* (also called *valued constraint satisfaction problem*) (Schiex et al. 1995). This formulation enables the use of highly efficient constraint solvers that can also enforce hard constraints C' (Allouche et al. 2015). For estimating the graphical model,

Brouard et al. (2020) learned the weights in the cost functions using an *alternating direction method of multipliers*. Following on that, Defresne et al. (2023) investigated learning a neural network that predicts the pairwise cost function tables (i.e, the weights), and proposed a pseudo-log likelihood based loss function that does not require calling the solver during learning. They showed it can be used both to learn (pairwise) constraints from data, e.g. for the Sudoku problem, as well as unary preferences (e.g. digit classifiers) for visual Sudoku and pairwise preferences over pairs of proteins in protein design.

Interactive learning. In an interactive learning setting, one is not given a dataset of preferences $\{(C', x')\}$, but rather the system can interact with the user by asking queries, and learn a preference function from the answers. Early works on preference elicitation for combinatorial problems assume that the user can communicate explicit preference structures such as CP-Nets with limited follow-up interaction (Boutilier, Brafman, et al. 1997); or that we can directly elicit the cost of an assignment in a soft constraint (Gelain et al. 2010). Other approaches assume a set of *feasible weights* for the (weighted) objective function that must be estimated. The seminal work of Boutilier, Patrascu, et al. (2006) uses *bound queries* which ask the user whether one of the utility parameters lies above a certain value. This was later extended to a polyhedral method by Benabbou and Lust (2019) that asks pairwise comparison queries which iteratively cut the polyhedron of feasible weights. However, these approaches cannot handle noisy answers from the decision maker. A Bayesian approach was investigated by Bourdache et al. (2020) for combinatorial problems, which utilizes a density function over the weight space, and where the answer to a pairwise query results in a Bayesian update to this density function.

Closer to standard machine learning, namely maximum margin learning such as in structured support vector machines, Teso et al. (2016) proposed the *setwise max-margin algorithm*, which can generate K alternatives solutions and iteratively learn from the users' choice among those (re-interpreted as $K - 1$ ranking constraints). They introduced the name *constructive* preference elicitation to highlight that the method constructs new solutions given a set of (linear) constraints. The method is restricted to Boolean variables/features, and Dragone et al. (2018) generalized the approach to linear utilities over arbitrary subobjectives and arbitrary constraints. Dubbed the choice perceptron, it generalizes the *preference perceptron* algorithm of Shivaswamy and Joachims (2015) to an interactive (constructive) preference elicitation setting. The recurring challenge in interactively eliciting and learning the objective (preference) function is how to efficiently generate solutions to query to the user, such that their answer will result in a strong learning signal.

A completely different approach to preference elicitation is taken by Lawless et al. (2024), in the context of scheduling meetings: they use a chat-based interface and let the user formulate preference elicitation about meeting requirements. These are translated by a large language model into a constraint (Python function) with a corresponding priority/weight; a CP-like procedure then searches for the most preferred meeting slot, and allows the user to iteratively specify new preferences or update previous ones.

3.1.3 Learning to Model - Automatic Model Generation (Figure 4). Modeling a real problem in a CP framework requires first translating it into a formal problem definition, and then formulating this as a CP model. While the first stage can already be difficult for a domain expert, the second stage requires significant expertise in declarative modeling: selecting decision variables and suitable (global) constraints is not a straightforward task. One question then naturally arises: "*Could this process itself be automated, starting from a natural language specification of a combinatorial optimization problem?*". Kiziltan et al. (2016) investigated whether natural language techniques could be used to *detect* which parts of an informal problem description corresponds to constraints. They collected, manually annotated 110 CP problems, and used a sequence classifier to detect constraint sentences. In response to Eugene Freuder's 2019 Holy Grail challenge, Claes et al. (2019) reported on an approach based on natural language processing to convert logic grid puzzle clues and other natural language sentences describing logical relations, into a formal constraint model for the IDP constraint modeling system. When focusing on logic grid puzzles specifically, Jabrayilzade and Tekir (2020) observed there are only 5 types of constraints expressed, and

they showed that fine-tuning a BERT-based classifier to identify the type followed by an extraction procedure leads to 100% accuracy on unseen puzzle descriptions form the same source.

Returning back to the challenge of translating generic constraint specifications, Kadıoğlu et al. (2024) proposed the challenge of *named entity recognition for combinatorial optimization specifications* (Ner4Opt), where the entities to identify are variables names, parameter names, constraint relations/directions, limits/bounds and the objective variable and direction. They use the LP-focused NL4OPT dataset from a 2022 NeurIPS competition (Ramamonjison et al. 2023), and evaluate how well both classical techniques from natural language processing and fine-tuning of a BERT-based language model perform. At that time, this raised unique challenges, with the road to moving from named-entity-recognition to an automatic model generation still open.

However, this changed rapidly with the advent of increasingly capable LLMs, especially GPT-3 and the subsequent development of coding LLMs (M. Chen et al. 2021). Within the context of CP, Michailidis et al. (2024) presented a first systematic study of pipelines and evaluation procedures for automated modeling with LLMs. They evaluated in-context learning techniques with multiple stages, on a logic grid puzzle dataset (satisfaction), NL4Opt (optimization) and on exercises taught in a CP course. While showing good results on the classical dataset, results on increasingly more difficult CP exercises show that more complex modeling remains challenging. In recent months, three CP-oriented datasets have been introduced: Text2Zinc (Singirikonda et al. 2025), CP-Bench (Michailidis et al. 2025), and CPEVAL (Y. Song and Cohen 2025), along with leaderboards on the HuggingFace platform for Text2Zinc and CP-Bench. This enables the continual development and evaluation of LLM capabilities for CP modeling.

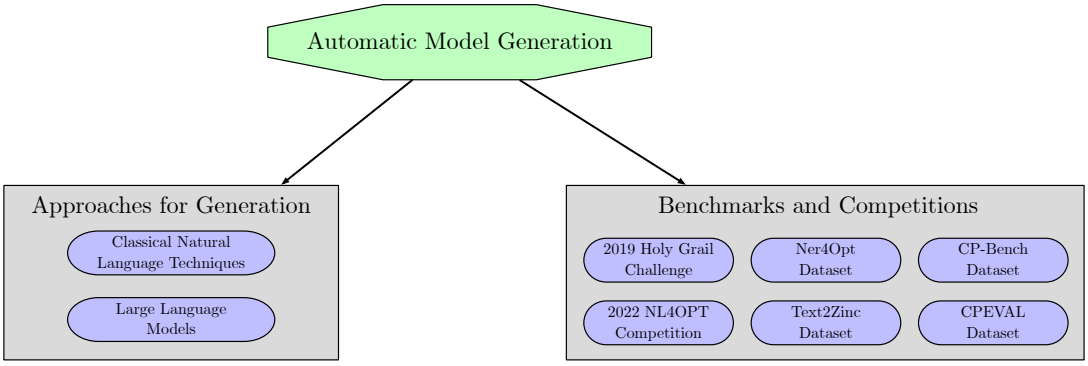

Fig. 4. Overview of Section 3.1.3 - *automatic model generation*.

3.1.4 Learning to Branch. Alongside propagation, branching is a cornerstone of most modern CP solvers. A typical branching strategy involves two key decisions: selecting the next variable to branch on and choosing the value to assign to it. The effectiveness of these decisions can have a significant impact on the efficiency of the search. Unfortunately, identifying the optimal choices is itself an NP-hard problem. As a result, many heuristics, often developed without the use of learning techniques, have been proposed to guide variable and value selection effectively. This section is dedicated to approaches that aim to learn appropriate branching heuristics for CP, from collected samples or training data. We distinguish between two categories: *offline methods*, where learning is performed prior to the CP solving phase, and *online methods*, where learning occurs dynamically during the solving process.

Online variable-selection heuristic. Since they do not require historical data to train a model, online learning-based heuristics were among the first to be explored, in contrast to their offline counterparts. Among these,

those focused on variable selection have received the most attention due to their substantial impact on search pruning and overall solver performance. Broadly speaking, most of these heuristics collect information during the search to dynamically determine an effective variable ordering. In the early 2000s, Boussemart et al. (2004) introduced a dynamic variable ordering heuristic aimed at guiding the search toward inconsistent or difficult regions of the problem. This dynamic heuristic, known as WDEG, leverages simple statistics about past conflicts to prioritize variables, and has since become widely adopted in the CP community. Building on this idea, Grimes and Wallace (2007) showed that restarts can enhance the effectiveness of WDEG, while Wattez, Lecoutre, et al. (2019) further refined it by incorporating more fine-grained information at each conflict. Still focusing on conflict-driven strategies, Habet and Terrioux (2019) introduced the *conflict-history search* heuristic, which records failures during the search to better guide variable selection. Similarly, Audemard, Lecoutre, et al. (2023) proposed a method that learns from conflicts by identifying variables that are directly involved in constraint propagation. Other works, such as Li, M. Yin, et al. (2021), have considered additional failure-related features, like the depth in the search tree at which a failure occurs. Beyond conflicts, R. Wang et al. (2017) explored the concept of *correlation*, defined as the degree to which domain reductions of one variable affect others during propagation. This information is then used to guide variable selection. Multi-armed bandit frameworks have also been explored for online heuristic selection. Xia and Yap (2018) applied bandit algorithms to select the most appropriate heuristic from a portfolio during the search. This idea was extended by Wattez, Koriche, et al. (2020), who integrated restarts, and by Koriche, Lecoutre, et al. (2022), who leveraged Luby's universal restart sequence to better balance exploration and exploitation. Still with bandits, Cherif et al. (2020) proposed to calibrate the conflict-history search heuristic presented previously. Another direction was taken by Doolaard and Yorke-Smith (2022), who proposed to extend existing variable ordering heuristics (i.e. smallest, anti first-fail, and maximum regret) through an initial online probing phase that collects instance-specific samples. Similarly, Li, Wu, et al. (2022) used probing to evaluate and select heuristics from a portfolio before initiating the backtracking search. Most recently, J. Xu, Wu, et al. (2025) proposed a method that learns heuristics from the topology of search trees, by recording the number of successful decisions made before a failure. This highlights that online variable-selection heuristics remain an active area of research in the CP community.

Offline variable-selection heuristic. The offline counterpart to online learning-based heuristics is comparatively less explored. To the best of our knowledge, the first such work is by Driel et al. (2021), who employed a graph neural network trained on an adapted version of the MINIZINC benchmark suite. The network is trained to predict which variables are likely to belong to an unsatisfiable subset of a CP instance. These predictions are then used to initialize the activity scores of the state-independent decaying sum heuristic within the solver CHUFFED. Graph neural networks have also been investigated for this task by W. Song et al. (2022), who combined them with reinforcement learning. Their approach is trained to minimize the expected number of nodes required to reach a leaf node, following the first-fail principle.

Online value-selection heuristic. Introduced more than 20 years ago, *impact-based search* (Refalo 2004) remains one of the most widely used value-selection heuristics in constraint programming. The core idea is to associate each branching decision $x = a$ with a quantitative measure (i.e. its *impact*), that reflects how effectively the decision reduces the search space. This impact is learned dynamically by observing domain reductions during the search. Later, Michel and Van Hentenryck (2012) proposed an alternative strategy based on *activity*, which records how frequently a variable is affected by propagation. Initially designed as a variable-selection heuristic, this approach was extended to value selection by tracking how much activity can be attributed to a specific decision $x = a$. To the best of our knowledge, and in contrast to the growing number of online heuristics for variable selection, there are currently no generic learning-based approaches specifically dedicated to online value selection. This may be a possible direction for future research, for example, by accelerating the computation of computationally expensive heuristics (e.g., (Fages and Prud'Homme 2017)).

Offline value-selection heuristic. Contributions in this area have become more prolific over the past decade with the rise of deep learning. Chu and Stuckey (2015) introduced a scoring function designed to evaluate the quality of an assignment given the current domain. This function is learned through a supervised training phase with a simple linear regression. Later, Cappart, Moisan, et al. (2021) proposed a reinforcement learning approach in which a machine learning model is trained outside the CP solver and then integrated into it post-training. In this setting, no internal solver information, such as the effectiveness of constraint propagation, is used to guide the learning process. To address this limitation, Chalumeau et al. (2021) introduced a learning-based CP solver, SeaPearl, which incorporates the solver into the training loop. However, the approach struggled to achieve competitive execution times. Subsequently, Marty, Boisvert, et al. (2024) and Marty, François, et al. (2023) introduced several enhancements to this solver, including a revised reward scheme and a learning pipeline based on restart strategies.

3.1.5 Learning to Reason. In addition to learning effective branching decisions, one can also enhance the reasoning capabilities of CP solvers. The objective here is to prune unproductive parts of the search tree, thereby accelerating the search process. In practice, it is desirable to guarantee that any pruning operation remains *correct*, meaning it should not mistakenly remove any solutions superior to the best found so far. Unfortunately, providing this guarantee is challenging when relying solely on machine learning approaches. In our view, this difficulty explains why research contributions on learning-based reasoning methods are considerably fewer compared to branching heuristics, where prediction errors can typically be rectified through backtracking. This section presents CP approaches based on lazy clause generation and those learning Lagrangian multipliers.

Deductive learning of nogoods. Lazy clause generation (Ohrimenko et al. 2009) is perhaps the most common learning-based component introduced in CP solvers. Briefly, it consists of recording domain restrictions that have previously led to failures during search. These failures are then integrated into the model as new constraints, commonly referred to as *nogoods*. Intuitively, nogoods can be seen as partial assignments that cannot be extended to a full solution. This process can be interpreted as deductive reasoning (i.e. applying inference rules to reach a particular fact) that will prevent the solver from reaching the same conflict again. The learning process occurs online, without requiring a dedicated training phase, which makes it easier to integrate as a generic mechanism within solvers. It is a core component of solvers such as Chuffed[3] and Pumpkin[4]. Since its introduction, lazy clause generation has been extended and improved in various ways. Moving beyond the purely online setting, Chu and Stuckey (2012) demonstrated how nogoods learned from one problem instance can be reused for solving similar problems. Although this mechanism can significantly reduce the search tree, there are situations where clause generation may negatively impact performance for reasons that were not fully understood. Shishmarev et al. (2016) addressed this issue by profiling the execution of the Chuffed solver to measure the impact of learned clauses. Through this profiling, they identified modifications that could be made to the original constraints to improve solver performance. However, this process was done manually and required deep expertise in the solver's internals. Building on this idea, Zeighami et al. (2018) proposed a first step toward an automatic framework for clause impact analysis. Beyond nogoods, Veksler and Strichman (2016) introduced new inference rules aimed at inferring more general global constraints, such as allDifferent.

Learning Lagrangian multipliers in CP. Inspired by mixed-integer programming solvers and their mathematical toolkit, including duality and Lagrangian methods, some researchers have proposed integrating similar techniques into CP (Focacci et al. 1999; Sellmann 2004). However, these methods generally incur a computational overhead that may limit their practical applicability. As a specific example, Benchimol et al. (2012) explored the use of *minimum 1-trees* as valid relaxations for the traveling salesman problem. Their approach, however, relied on

[3]https://github.com/chuffed/chuffed
[4]https://github.com/ConSol-Lab/Pumpkin

heuristic decisions that critically influenced the tightness of the relaxation. One notable issue was generating tight Lagrangian multipliers, computed using an expensive sub-gradient procedure. To overcome this limitation, Parjadis et al. (2024) introduced a self-supervised learning method designed to directly predict accurate Lagrangian multipliers. Interestingly, these predicted multipliers can also effectively serve as warm-starts for the traditional sub-gradient approach. Another example is the use of Lagrangian decomposition as a general bounding mechanism in CP. The key idea is to relax the original problem by decomposing it into several independent subproblems, each easier to solve individually. Each subproblem is associated with a vector of Lagrangian multipliers. Solving this relaxed formulation provides a dual bound, whose tightness directly depends on the values of these multipliers. Typically, these multipliers are again optimized using a sub-gradient approach. Although the initial study by Hà et al. (2015) demonstrated the potential of this method to significantly enhance the filtering capabilities of CP solvers, it remains computationally intensive, as multiple combinatorial sub-problems must be solved repeatedly during each iteration of the sub-gradient optimization. To address this issue, Bessa et al. (2025) adapted the self-supervised learning method proposed by Parjadis et al. (2024) specifically for this context, enabling direct prediction of effective Lagrangian multipliers. Their approach successfully reduced the number of sub-gradient optimization iterations required for convergence, improved pruning efficiency, and thus substantially decreased the overall execution time of a CP solver.

3.1.6 Reshaping the Solver Engine. The core search engine of standard CP solvers typically operates in a depth-first manner (e.g. Gecode, Choco, MiniCP), using backtracking to handle conflicts, either through trailing or copying mechanisms to restore previous states. However, some outside-the-box solving strategies redefine this standard approach by adopting alternative search strategies, sometimes incorporating learning-based components. This section presents such alternative methods, specifically focusing on three approaches that integrate learning: adaptive large neighborhood search, Monte Carlo tree search, and solvers augmented with belief propagation or with ant colony optimization. We also review approaches that bypass the search procedure, instead leveraging ML to skip the search phase.

Adaptive large neighborhood search. In his paper introducing large neighborhood search, Shaw (1998) already stressed the importance of a good choice of neighborhood. In his case this was the of variables corresponding to *related* customers in a vehicle routing problem. Even though in principle the neighborhood can be chosen at random, adapting this choice to the nature of the combinatorial problem can be more productive. It is therefore natural that some researchers proposed learning to make such good choices automatically. To this end, Laborie and Godard (2007) proposed *self-adapting large neighborhood search* which uses biased-wheel selection from a portfolio of neighborhoods and updates the weight of selected neighborhoods dynamically according to their performance during search. While this proposal was applied to scheduling, it has the potential to be used in other contexts. Thomas and Schaus (2018) built on the previous work and adjusted it to account for some neighborhoods being explored faster and therefore having their weight updated more often, leading to neighborhoods being evaluated at different rates.

Monte-Carlo tree search. Initially introduced as an alternative to alpha-beta search for solving zero-sum games with perfect information, *Monte-Carlo tree search* (MCTS) (Coulom 2006) has gained significant traction in the broader AI community. This interest was further amplified by the success of *AlphaGo* (Silver et al. 2016), which combined MCTS with deep learning to defeat world champion Lee Sedol in the game of Go. Given the combinatorial structure of such games, researchers naturally explored the application of MCTS to combinatorial optimization problems. In the context of CP, one of the earliest efforts was introduced by Loth, Sebag, Hamadi, and Schoenauer (2013), who proposed adapting MCTS to CP by designing a generic reward function tailored to CP search and compatible with a multi-restart strategy. Their approach also used depth-first search as the roll-out procedure and was applied to the job-shop scheduling problem (Loth, Sebag, Hamadi, Schoenauer, and Schulte

2013). To our knowledge, this line of research remained largely unexplored in the CP community for nearly a decade. More recently, Antuori et al. (2021) revisited this idea in the context of a car manufacturing workshop scheduling problem. They employed MCTS to balance exploration and exploitation within the search space while collecting informative data about explored subtrees. One key challenge they identified is that evaluating a search node in CP is considerably more difficult than in classical game settings. Their proposed evaluation scheme is based on the marginal improvement of the primal bound of the objective function across previous simulations, incorporating an exponential decay based on depth. Interestingly, although related works (e.g. Ménard et al. (2023)) have used MCTS as a standalone mechanism to tackle combinatorial problems, its use as a replacement for the traditional CP search engine has not seen further significant development. This may suggest that obtaining competitive performance with such an approach remains a substantial challenge. Notably, the previous methods did not incorporate neural networks such as in *AlphaGo* and its variants.

Solvers augmented with belief propagation. As mentioned in Section 2.1, the core propagation mechanism in CP is a form of message passing over the constraint network. Its messages from constraints are not probabilistic but deterministic, providing a binary information about removing or assigning domain values. Pesant (2019) generalized these messages from 0/1 to real-valued, representing how much support a domain value has in solutions to a given constraint. This support can be interpreted as a probability distribution over the domain of a variable for solutions sampled uniformly at random. Computing such messages requires replacing domain filtering algorithms by weighted counting algorithms. Constraint propagation is also replaced by *belief propagation* (Pearl 1982), a well-known inference algorithm that computes marginal distributions over the variables. That computation is exact in the special case of trees but only approximate for general constraint networks. These learned marginals have been used to guide search (Burlats and Pesant 2023) and to enable some neurosymbolic architectures that are later discussed in Sections 3.2.2 and 3.2.3.

Search with ant colony optimization. Ant Colony Optimization (ACO) (Dorigo, Birattari, et al. 2007; Dorigo, Maniezzo, et al. 1996) has also been explored in the context of CP as a metaheuristic to guide the search process. The central idea is to exploit pheromone trails (i.e. a reinforcement learning mechanism) to bias variable- and value-selection heuristics, thereby combining the adaptive learning ability of ACO with the pruning strength of CP. Early studies by Meyer and Ernst (2004) and Khichane et al. (2008) demonstrated how ACO can be integrated into CP solvers to address scheduling and car sequencing problems, while Khichane et al. (2010) later extended this integration to IBM's CP Optimizer. Several applications have since been proposed, including balancing bicycle-sharing systems (Di Gaspero et al. 2013), group cumulative scheduling (Groleaz et al. 2020), and other combinatorial problems (Solnon 2010). Despite these contributions, recent contributions of ACO within CP remain relatively limited compared to other learning-based approaches, leaving ample room for further exploration, particularly in terms of novel methodological developments.

Skipping the search phase. Intuitively, this family of approaches bypasses the traditional reasoning and branching loop illustrated in Figure 1 by directly predicting either the full solution to a combinatorial problem or constructing it incrementally, assignment by assignment. This is arguably the simplest and most widely explored application of learning to combinatorial optimization. In this section, we focus specifically on such end-to-end approaches designed to solve CSPs (rather than a specific problem) or those that integrate CP techniques at some stage in their pipeline. A first study in this direction was conducted by Galassi et al. (2018), who investigated the ability of neural networks to construct solutions for CSPs without relying on any explicit symbolic representation of the problem constraints. Their approach involved systematically extending partial solutions by making single assignments that were globally consistent. In the same year, H. Xu et al. (2018) proposed using deep learning to predict the satisfiability of CSPs. To address the challenge of limited labeled data for training, they introduced a method for randomly generating new CSP instances to populate the training set. However, their experiments were limited

to random binary CSPs, which are solvable in polynomial time and do not reflect the complexity of standard NP-hard CSPs typically encountered in practice. This line of work was later extended by Zheng et al. (2023), who applied a different architecture, FastMapSVM, to the satisfiability prediction task. Additionally, reinforcement learning has been explored in this context by Tassel et al. (2023), who proposed an end-to-end learning approach for the job-shop scheduling problem. In their framework, CP is used to model the RL environment: CP variables represent the state of the environment, and the agent selects variables to fix. State transitions include constraint propagation to reduce variable domains. It is worth noting that CP is not used in the final solving phase, but rather serves to structure the environment for learning. Finally, some papers also used CSP problems to test the limits of reasoning LLMs, e.g. the ability of pretrained LLMs to solve Zebra puzzles (Lin et al. 2025) or other logic puzzles (Giadikiaroglou et al. 2024).

3.1.7 Learning to Configure a CP Solver. The performance of a CP solver can depend significantly on how it is configured. For example, a user may choose between different search heuristics (e.g. variable and value selection), different branching schemes (e.g. one value versus all, or domain splitting), or even radically different search approaches (e.g. LNS instead of the standard search). Similarly, many global constraints feature multiple propagators, offering different trade-offs in terms of their effectiveness at pruning values and their computational cost. Finally, completely different CP solvers might be better suited at tackling different problems. The impact of the solver configuration presents both an opportunity and a challenge: on the one hand, proper tuning can significantly enhance solver performance for a given problem; on the other hand, selecting the right configuration can be difficult even for expert users, thereby raising the barrier to the effective use of CP technology.

Combining learning and optimization has long been recognized as a promising direction for addressing this challenge. The underlying idea is that, similarly to how experts are more adept at configuring solvers thanks to their experience, so an automated system could replicate this feat by relying on past experiments. In the literature, this task has been framed either in terms of *algorithm selection* (Rice 1976) or *algorithm configuration* (Adenso-Diaz and Laguna 2006). The former consists in choosing from a small pool of candidates, the best solver to address a given problem instance; the latter involves calibrating the parameters of a given algorithm so as to maximize its performance on a target distribution of instances. In practice, the distinction between algorithm selection and configuration has blurred considerably over time, due to cross-fertilization between research in the two fields and to the fact that different parameterizations of the same algorithm can always be viewed as distinct algorithms.

Algorithm configuration is of interest for many research fields besides CP, such as operations research, evolutionary computation, and machine learning. As a result, most approaches in this area are designed with a general-purpose mindset and can, in principle, be applied to arbitrary optimization algorithms. Nevertheless, only a few methods in the literature have been specifically tailored to constraint programming, even though the topic has long been recognized as important within the CP community. Providing extensive coverage of the literature in this area is beyond the scope of this survey. Instead, we limit ourselves to discussing some key ideas in the existing methods, highlighting papers that have focused on CP.

Optimized configurations for entire problem classes. Approaches for algorithm configuration have traditionally focused on improving the performance of a solver *for a given distribution of instances*, represented through a training sample. This task can be framed as an optimization problem, where the search space includes the possible configurations and the goal is minimizing (or maximizing) a performance metric, such as the runtime to find an optimal solution. For example, Minton (1996) proposed a system that can specialize generic algorithms for a given class of CSPs. The algorithms considered are primarily backtracking schemes, similar to those employed by many CP solvers. The specialization lies in synthesizing variable and value ordering heuristics by combining known rules, a process carried out either through search or heuristic techniques.

Instance-specific methods. Algorithm selection methods have conversely focused on achieving optimal performance *on individual instances*. This can be tackled by first solving a training benchmark via multiple approaches, then training a ML classifier to identify the top performing algorithm, based on feature vectors that represent the properties of each instance. This is the approach taken for example by Guerri and Milano (2004), which choose between a mixed-integer programming or a CP approach for a specific combinatorial problem. The selection is done by means of a decision tree. Decision trees are also employed by Gebruers et al. (2005), which select between multiple solution strategies for a given CP problem. Such solution strategies can make use of different model formulations, algorithms and search heuristics. A second example of algorithm selection based on modeling is the Proteus system (Hurley et al. 2014), which targets CP problems but may choose to encode them into SAT instead of solving them directly with a CP solver. Another illustrative case is the work by Gent et al. (2010), who use multiple types of machine learning classifiers to select among different propagators for the AllDifferent constraint. Additionally, CP-Hydra (O'Mahony et al. 2008), SUNNY (Amadini et al. 2014), and SUNNY-CP (Amadini et al. 2015) address both CSPs and COPs using a k-nearest neighbor algorithm. Unlike many algorithm selection methods that choose a single solver, these approaches output a set of solution algorithms. This distinctive characteristic is discussed later in a subsequent paragraph. More recently, SUNNY was extended by Liu et al. (2021) to tackle general optimization problems. Besides, Kadioglu, Malitsky, Sellmann, et al. (2010) were among the first to combine exploration of large parameter spaces on par with classical algorithm configuration, which resulted in the ISAC system and the ability to output instance-specific parameter vectors. The method operates by first clustering a set of instances based on features, and then applying a standard configurator to optimize solver parameters for each cluster. Methods relying on sequential model based optimization, such as the SMAC algorithm by Hutter et al. (2011), manage to be instance-specific by learning surrogate models on a shared input that includes both the parameters to be optimized and the instance features.

Configuration from instance descriptions. All instance-specific methods rely on a representation of the problem instance to select an appropriate configuration. This representation typically takes the form of a vector of *features*, which serve as input to the ML model driving the selection process. Most approaches use *syntactic* features designed to capture structural properties of the instance, such as the number of constraints, their arity, or domain sizes. Some methods, like CP-Hydra discussed earlier, supplement syntactic features with dynamic ones obtained by running a solver for a limited amount of time and collecting statistics from its behavior. In contrast, Collautti et al. (2013) incorporate information from past solver performance to enhance the performance of ISAC. A few works are notable due to their unique approach to feature extraction. In particular, Pulatov et al. (2022) assume access to the source code of the algorithms to be optimized, and rely on code analysis tools to obtain additional information and make more accurate predictions. Finally, Loreggia et al. (2016) replaced the feature extraction by a procedure that encodes the textual representation of the considered problem instance as a greyscale image, which is then used as the input for a neural network.

Portfolio-based algorithm selection. While traditional approaches to algorithm configuration have focused on selecting a single algorithm or configuration, extensive research has shown that leveraging multiple algorithms can be beneficial. This is particularly relevant for NP-hard problems, where the performance of a single algorithm can vary significantly across instances. By running a diverse set of algorithms, one can mitigate the risk associated with this performance variance. Consequently, several methods have been developed to produce a *portfolio* of algorithms, rather than relying on a single choice. The earliest approaches in this family identify a set of algorithm to be executed sequentially, in a preemptive fashion, with non-uniform computational budgets, for instance, proportional to their estimated success probability. Such *sequential portfolios* are generated by CP-Hydra, by CP-SUNNY, and their subsequent evolutions. The 3s system (Kadioglu, Malitsky, Sabharwal, et al. 2011) is notable in this space because it provides a declarative formalization for the task of computing an optimal sequential schedule.

Research on *parallel portfolios* emerged shortly after the development of sequential ones, aiming to leverage the increasing number of cores available in modern CPUs. A representative method in this area is the work by Malitsky et al. (2012), which similarly to Kadioglu, Malitsky, Sabharwal, et al. (2011), provides a declarative formulation for computing an optimal parallel schedule. An interesting contribution by Lindauer, Hoos, et al. (2015) enables to re-use the ML models from sequential selectors to build a parallel portfolio. This is achieved by generating a solver ranking and then using the top-k solvers. More recently, Kashgarani and Kotthoff (2023) introduced what is likely the first approach trained on true parallel executions, as previous methods relied on simulated parallelism. The distinction is relevant, since concurrent access to shared resources (e.g. memory and cache) can have a significant impact on the performance of a solver.

Acceleration techniques. All such approaches must contend with the high computational cost of evaluating the objective function they aim to optimize (e.g. running a solver to observe runtime). Consequently, extensive research has focused on developing techniques to accelerate or circumvent these evaluations. *Surrogate models*, such as those used by SMAC (Hutter et al. 2011) and its more recent version SMAC3 (Lindauer, Eggensperger, et al. 2022), aim to reduce the number of costly evaluations of the performance metric by approximating it with a predictive model. In the context of CP, Bleukx et al. (2022) combined adaptive capping (i.e. a time out matching that of the best approach found so far) with surrogate-based optimization and decomposed surrogate models. Additionally, priors are associated with each parameter to warm-start the optimization process. However, a key limitation of this approach is that it currently supports parameter optimization only on a single instance. Further, Kuş et al. (2024) employed three distinct acceleration techniques: a surrogate model with uncertainty quantification; a timeout predictor; and an active learning strategy that selects only a subset of instances for training.

3.1.8 Learning for Generic and Specific Combinatorial Problems. This final subsection on ML for CP first summarizes approaches tailored to specific combinatorial problems. Notable examples are scheduling problems where CP solvers achieve strong performances. Such problem-specific approaches leverage particular structural insights and domain-dependent features, enabling more efficient solutions. However, this advantage comes with the drawback that design choices made in these methods typically cannot be generalized to other problems. As a result, some research focuses instead on developing methodologies that are broadly applicable to *any* combinatorial problem.

Resource constrained project scheduling problem (RCPSP). This problem has recently attracted significant interest in the development of hybrid CP and ML approaches. Briefly, the goal of this problem is to schedule a set of tasks within a given time horizon, respecting resource capacities and precedence constraints between tasks. In this context, Teichteil-Königsbuch et al. (2023) introduced an approach using a graph neural network to imitate solutions produced by a CP solver. To guarantee feasibility, the solutions generated by the GNN are subsequently refined using a dedicated scheduling algorithm. The resulting method achieved superior solutions compared to those produced directly by the CP solver within an equivalent computational time budget. Subsequently, Verhaeghe, Cappart, et al. (2024) also utilized a GNN, albeit with a different strategy. Their model aims instead at identifying additional precedence relations between tasks, which can then be integrated either as redundant constraints or employed to guide variable selection heuristics in the CP solver.

Job shop scheduling problem (JSSP). The objective of the JSSP is to schedule multiple jobs across several machines, subject to operational constraints, while optimizing a given objective function, such as makespan. Lee and Kim (2022) proposed an approach where CP serves as an expert in an imitation learning framework, producing high-quality solutions to the JSSP. These solutions then serve as training labels for a GNN, enabling rapid generation of new solutions. Similarly, Tassel et al. (2023) later explored reinforcement learning methods for the JSSP, using feedback and examples from a CP solver as the training signal. Rather than employing CP

solutions as supervised training data for heuristics, Sun et al. (2024) took a different approach by leveraging learning techniques to optimize variable ordering heuristics within the CP solver itself.

Cyclic hoist scheduling problem (CHSP). Although less studied compared to the RCPSP and JSSP, the hoist scheduling problem has also been addressed through hybrid methods. The objective of this problem is to determine a feasible schedule for a hoist tasked with transferring objects between tanks, minimizing the cyclic period. In this context, Efthymiou and Yorke-Smith (2023) explored how machine learning approaches could accelerate exact solution methods. Specifically, their models are trained to predict the optimal cyclic period, serving as an effective bound. This predicted value is subsequently used within a CP solver to significantly reduce computation times without compromising the optimality of the obtained solutions.

Generic representation of combinatorial problems. The previously discussed approaches lack generality because their representations cannot easily be transferred between different combinatorial problems. From a user's perspective, this poses a significant issue, as solving a new combinatorial problem requires designing an entirely new architecture and representation. Motivated by the need for greater generality, several works have progressed towards a generic representation of combinatorial problems suitable for learning-based approaches. Among these, the bipartite representation introduced by Gasse et al. (2019), although not directly related to CP and limited to binary mixed-integer programs, is one of the pioneering efforts. They proposed modeling the problem as a bipartite graph, connecting variables and constraints whenever a variable participates in a constraint. Within a CP context, Chalumeau et al. (2021) expanded upon this idea by introducing a tripartite graph representation, where variables, values, and constraints each form distinct vertex types. However, their approach still lacks complete generality since it requires retraining whenever the number of variables changes. Building on this limitation, Marty, Boisvert, et al. (2024) and Marty, François, et al. (2023) augmented each vertex type with dedicated features. This advancement theoretically allows encoding any combinatorial problem but comes with the drawback of losing some information during encoding. Seeking to avoid such information loss, Boisvert et al. (2024) introduced a fully generic representation of combinatorial problems. Their core idea involves decomposing constraints into abstract syntax trees, where similar elements (e.g. identical variables or constraints) are connected by edges. Subsequently, learning is performed using graph neural networks in a supervised manner. In a related yet distinct approach, Tönshoff et al. (2023) employed a recurrent graph neural network to learn from a tripartite representation, but leveraged reinforcement learning instead. Interestingly, nearly all these methods encode combinatorial problems as graphs and utilize graph neural networks for learning. A recent exception is presented by Y. Xu et al. (2025), who introduced a self-supervised transformer capable of iteratively improving solutions to constraint satisfaction problems. Notably, their method handles various global constraints, such as AllDifferent and Cardinality.

3.2 Improving ML with CP

Generally speaking, machine learning involves two phases: *learning* and *inference*. We intentionally omit the hyperparameter optimization phase, as it primarily involves model selection and offers fewer opportunities for improvement through CP techniques. As detailed in Section 2.2, the learning phase consists of training a specific type of model (e.g. neural networks or decision trees) using historical data. Depending on the application, certain desirable properties might be required for the model. For example, applications demanding transparency, such as medical diagnosis, might require models that are explainable. Unfortunately, achieving these properties can be challenging depending on the type of model. This presents an opportunity for integrating CP approaches. Once trained, the learned model is applied to generate predictions for new instances, a process commonly referred to as inference. CP has also been leveraged at this stage to filter predictions that fail to adhere to properties. Some

of these properties can be expressed by means of constraints. Figure 5 provides an overview of these concepts together with pointers to the next subsections.

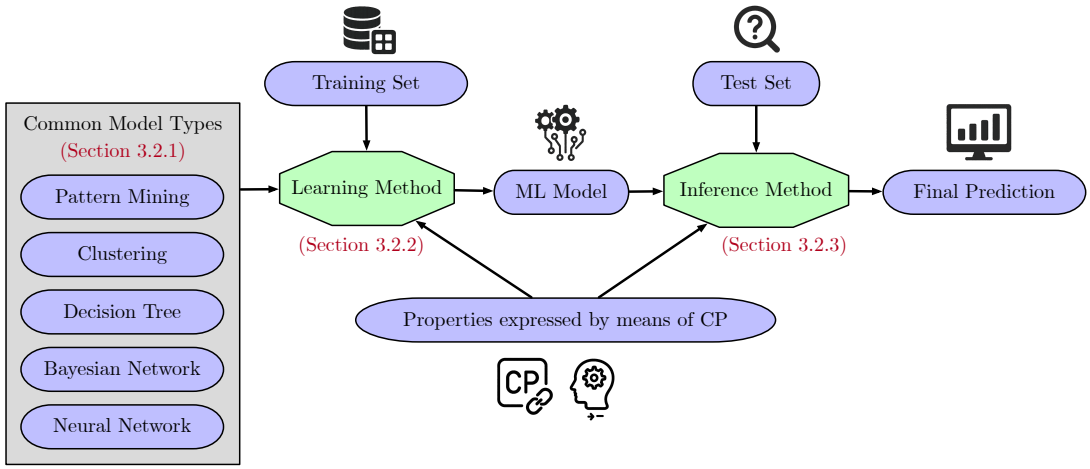

Fig. 5. Overview of a typical machine learning pipeline and highlight specific stages where CP techniques can be effectively integrated. In particular, we examine opportunities for integration at the model level, during the learning process, and at the inference stage. The following subsections correspond to each of these three integration points.

3.2.1 CP-based Learning for Different ML Model Types. Within the field of machine learning and data mining, the learning task typically corresponds to an optimization or enumeration problem. It should hence not be a surprise that CP techniques have been used for these tasks, especially when dealing with constraints, e.g. leading to *constrained* optimization and enumeration problems. We survey how CP has been successfully used for various types of machine models and data mining tasks.

Pattern mining. Within the field of data mining (Aggarwal 2015), pattern mining focuses on extracting meaningful patterns from data. The type of patterns mined typically depends on the intended downstream tasks.

(1) *Itemset patterns*: given a database of transactions, each transaction consists of a set of items; an itemset is a set of items that frequently occurs in the database of transactions (Fournier-Viger et al. 2017). Finding itemsets is a constrained enumeration problem. While itemsets are usually required to appear frequently in the data, a range of other constraints have been studied, including closedness, maximality, cost-based, discriminativeness, etc. The work of De Raedt, Guns, et al. (2010) and of Guns, Nijssen, et al. (2011) was the first to show that generic CP systems can be effectively used for constraint-based pattern mining. As efficiency is key in pattern mining on large databases, Lazaar et al. (2016) introduced the closedPattern global constraint for more efficient frequent closed pattern mining with CP solvers. Schaus, Aoga, et al. (2017) then introduced the coverSize global constraint, for general counting and constraining the number of transactions covered by the itemset decision variables. Its propagator has a strong correspondence to the highly efficient bitset-based table constraint propagator. Results show that it outperforms other CP approaches, while still being able to model a large range of constraint-based mining tasks. Other works focused on expanding the reach of constraint-based itemset mining problems that can be solved using constraint programming. Aiming for maximum generality, Guns, Dries, et al. (2013) extended the MiniZinc constraint modeling language to MiningZinc, introducing custom functions and predicates for

itemset mining. They also extended the MiniZinc toolchain to analyze the problem specifications, and use specialized itemset mining algorithms, general purpose constraint solvers or a combination of the two to compute the patterns.

(2) *Sequence patterns*: Negrevergne and Guns (2015) introduced CP formulations for constraint-based sequence mining, and Kemmar et al. (2015) introduced a global constraint that incorporates techniques from classic sequence mining algorithms, achieving similar performance with a CP solver. Extensions include an efficient global propagator for time-constrained sequence mining (Aoga et al. 2017), as well as for mining time-constrained *episodes* in a single sequence (Cappart, Aoga, et al. 2018).

(3) *Additional patterns:* The flexibility that CP offers to include additional constraints continues to make it an appealing framework for pattern mining, also leading to new constraint-based pattern mining tasks. Examples include n-ary pattern mining (Khiari et al. 2010), k-pattern set mining (Guns, Nijssen, et al. 2013), skypattern mining (Ugarte et al. 2017), the mining of diverse sets of patterns with constraint programming (Dzyuba et al. 2017; Hien et al. 2024), and mining closed interval patterns on numeric data (Bekkoucha et al. 2024). Finally, Koptelov et al. (2023) focus on explainability in the context of pattern mining for sensitive domains such as drug discovery, and propose using a CSP-based approach to guide the mining process and to explain why certain patterns are not found.

Clustering. Also for the task of clustering, and more specifically *constrained clustering*, the benefit of constraint programming has been investigated. Dao et al. (2015) proposed a global constraint for the widely used 'within-cluster sum of squares' objective function, and showed how it outperforms other exact methods. This framework is improved and extended by Dao et al. (2017), who included the ability to provide bounds on the number of clusters rather than fixing them and the support for various other constraints. Furthermore, the flexibility of the CP-based approach is shown in a novel bi-objective constrained clustering setup. To increase scalability, Guns, Dao, et al. (2016) hybridized this CP-based constrained clustering approach with an exact repetitive branch-and-bound approach, making a CP-based constrained version of the latter while maintaining its computational strengths. For the class of cardinality-constrained clustering problems, Haouas et al. (2020) showed that a CP-based approach with appropriate lower bounds, domain filtering, and branching can improve the efficiency further. Other constrained clustering problems have been tackled with constraint programming too, e.g. minimal cluster modification (Kuo et al. 2017), *conceptual* clustering where the features are binary like in itemset mining (Chabert and Solnon 2017; Guns, Nijssen, et al. 2013; Khiari et al. 2010), and density-based clustering (Grossi et al. 2015).

Decision trees. Decision trees are popular interpretable classification models. While the use of greedy algorithms to build them is widespread, such approaches fail to provide any guarantees on classification accuracy or decision tree complexity (such as a maximum depth). To address this issue, several exact approaches have been proposed to build optimal decision trees according to various criteria, including a few based on CP. Bessiere, Hebrard, et al. (2009) minimize the number of nodes in the tree subject to achieving perfect accuracy on the training data. Taking a dual perspective, Verhaeghe, Nijssen, et al. (2020) maximize the accuracy on the training data subject to a maximum tree depth, using the coverSize global constraint for itemset mining, and/or search, and caching.

Bayesian networks. Following the successful use of integer linear programming for structure learning of Bayesian networks from discrete data (Bartlett and Cussens 2017), Van Beek and Hoffmann (2015) investigated a CP approach augmented with dominance constraints, symmetry-breaking constraints, cost-based pruning rules, and an acyclicity constraint to further increase the effectiveness of the constraint-based structure learning approach. This was later extended by Trösser et al. (2021) with improved handling of the acyclicity constraints, leading to significant performance improvements. Constraint programming has also been used for answering exploratory queries of Bayesian Networks (Babaki et al. 2015).

Neural networks. As early as the 1990s, Kok et al. (1996) considered using constraint programming to train *constrained* neural networks. More specifically, they were motivated by cases where backpropagation would not be applicable, e.g. due to a recurrent structure or non-differentiable losses, and where genetic algorithms could be used for training. They then used the ECLiPsE constraint solver to restrict the architecture and weight representation used by the genetic algorithm. More recently, Toro Icarte et al. (2019) investigated optimal training of binarized neural networks with CP, including a hybridization with mixed-integer programming techniques. While these binarized neural networks only have activations and weights of +1/-1, their typical training is done with gradient descent followed by rounding. Toro Icarte et al. (2019) instead explored model-based approaches, e.g. formulating the learning problem as a constrained optimization problem and using constraint solvers, with good results in the low data regime.

3.2.2 CP-based Loss Functions. Because CP often expresses desired properties as combinatorial constraints, which are typically non-differentiable, it presents a technical challenge when defining a suitable loss function for end-to-end learning to satisfy such properties. One solution, adopted by some *layer-based* approaches, is to incorporate a differentiable layer representing the combinatorial structure within the network architecture. Examples include *SATNet* (P. Wang et al. 2019), a differentiable maximum satisfiability solver, and *semantic probabilistic layers* (Ahmed et al. 2022) which translates hard symbolic constraints into a differentiable circuit. However, most CP-related research has focused on *loss-based* approaches, adding a regularization term to the loss function.

This has been approached in two ways. Given the current (partial) assignment, one can either assess whether a constraint is violated, or identify domain values for each variable that lack support under the current constraints. Silvestri et al. (2021) studied the injection of information from a CP solver in order to learn globally consistent assignments. Given a CSP with some variables already assigned, the authors define an indicator function $\mathbf{1}_D$ over all possible assignments to the remaining variables. This function takes the value 0 for assignments that have been filtered out by the CP solver, and 1 otherwise. They also investigated several regularization terms $\mathcal{L}(\mathbf{1}_D, f_\theta(x))$ based on which domain values are still supported after constraint propagation. These include penalties for unsupported assignments, binary cross-entropy, and mean squared error. Their experiments on the *partial latin square problem* showed low sensitivity to the size of the training set size but also reveal scalability challenges.

The same year, Detassis et al. (2021) loosely combined a supervised learning method and a symbolic solver (CP or other) in a teacher-learner architecture called *moving-targets*. The method alternates between the learner and the teacher (i.e. the solver), the latter modifying the target vector used by the former. The new target y^{k+1} is selected so that it satisfies the constraints while being *in between* the current target y^k and the prediction, minimizing $\mathcal{L}(z, y^k) + \frac{1}{\alpha}\mathcal{L}(z, f_\theta(x))$ over all satisfying z (with hyper-parameter α controlling the combination of the two terms). Despite not guaranteeing constraint satisfaction, it performed well on tasks requiring fairness such as classification with protected features (e.g. ethnicity, gender).

On another front, D. Chen et al. (2020) designed the end-to-end framework *deep reasoning nets* (DRNets) combining deep learning and constraint reasoning for unsupervised image-based combinatorial games such as *visual Sudoku* and *visual mixed Sudoku* for pattern de-mixing. Combinatorial constraints are relaxed into continuous differentiable loss functions based on the entropy of probability distributions over variables and that act as regularization terms. In the case of Sudoku, these loss functions serve to minimize the entropy of each cell's digits distribution (encouraging the convergence to one digit) and to maximize the entropy of the average distribution of digits in each row, column, and sub-square (encouraging all digits to appear). The framework was later improved by Bai et al. (2021) through the use of curriculum learning and restarts on a pool of trained models. However, it is unclear whether this approach to constraint relaxation based on entropy can be extended to other types of constraints.

Turning now to reinforcement learning, the reward signal can be interpreted as a form of loss function guiding the learning process. In this context, Lafleur et al. (2022) applied RL to music generation, using it to adjust a pre-trained RNN by enforcing arbitrary constraints through a CP solver enhanced with belief propagation (see Section 3.1.6). They designed a reward function combining the RNN's learned probabilities, constraint violations, and marginals. As a result, they provided empirical evidence that the agent preserves the behavior of the pre-trained RNN while successfully incorporating the specified constraints. Later, C. Yin et al. (2024) simplified this architecture and designed a reward signal based on the change in expected value of a constraint violation variable. This was made possible thanks to the availability of a marginal distribution over its domain. They showed that this simplification led to faster convergence (i.e. improved sample efficiency), a higher constraint satisfaction rate, and better generalization. Finally, C. Yin et al. (2025) formalized that line of work as a *potential-based reward shaping* and confirmed the previously-observed advantages on a planning task in artificial intelligence. However, the sample efficiency from this line of work is somewhat offset by the additional computational cost of belief propagation.

3.2.3 CP-based Inference. Despite all the progress in training neural models to exhibit desired output properties, whether through CP or other methods, these models may still occasionally produce output that does not satisfy the output properties. This can be highly problematic in contexts where satisfying these properties is a hard constraint, such as in *safe robotics* (Brunke et al. 2022). Consequently, researchers have naturally turned to CP to enforce desired properties during inference. The types of properties typically addressed by CP involve structured outputs defined over multiple interdependent variables, subject to logical or numerical constraints. The core challenge is not only enforcing the properties, but doing so while maintaining proximity to the outputs of the neural model. In other words, we want to stay faithful to the learned distribution from the training corpus.

In this context, Dragone et al. (2021) proposed NESTER, namely for *NEuro-Symbolic consTrainEd structuRed predictor*, an architecture built using the modeling language MiniZinc. The goal is to enable the integration of hard constraints that describe desired properties with both the low-level inputs and the neural network predictions. They experimented on handwritten equation recognition, an image-based combinatorial problem, where a convolutional neural network predicts digits and operators from images and the COP is part of a structured predictor that drives the final prediction to remain close to the network's prediction while enforcing syntactic and semantic rules. This approach can be trained end-to-end, and they achieved better results than either of the two components on its own. The authors also reported that NESTER was particularly effective in a low data regime.

Mulamba, Mandi, Canoy, et al. (2020) considered image-based combinatorial problems such as *visual Sudoku* and investigated a hybrid architecture involving a neural image classifier and a CP solver. The poor performance of the solver in completing the puzzle when using classified digits from the neural model, due to classification errors that make the puzzle unsolvable, motivated their proposal to instead retrieve the (log) class probabilities for each digit and use them as costs over the domain values of the corresponding variables in the CP model of a Sudoku puzzle. Optimizing over these costs yields a maximum likelihood solution which performs much better than the looser integration using the predicted digits. Mulamba, Mandi, Mahmutogullari, et al. (2024) later incorporated the ability to handle real-life images of the puzzle and to correct human mistakes in partly-filled grids. The authors encouraged the further integration of predicted probabilities from the neural model with constraint solving.

In that vein, Manibod et al. (2025) introduced GeAI-BLAnC (*Generative AI using BeLief-Augmented Constraints*) which combines the predicted probabilities from a sequential generative model with the marginal probabilities computed by a CP model encoding the desired properties (see Section 3.1.6). At inference time, the next token is sampled from the resulting probability distribution. In particular, the focus was on imposing long-term structure to the generated sequence, which requires balancing foresight over future tokens and immediacy of next-token

generation. The experiments on molecule design and melody generation under constraints showed a high satisfaction rate while maintaining close adherence to the learned distribution.

The specific domain of natural language generation poses an additional challenge for CP. The very large vocabulary of LLMs, easily tens of thousands of words and subwords, induces equally large domains for the variables, which may be overwhelming for traditional CP solvers. To circumvent this, recent works either reduced the context to a much smaller vocabulary, implemented strong domain filtering, or gradually defined domains restricted to the most likely tokens. Bonlarron, Calabrèse, et al. (2023) considered a highly constrained case study related to vision screening, requiring a precise number of characters and lines, inter-word spaces of limited width, and a small prescribed vocabulary. Because of this severe restriction, they managed to represent all possible sentences in a multi-valued decision diagram and then used an LLM to rank each, reporting the best ones. Bonlarron and J. Régin (2024) worked later with an n-gram model in which the sequence of variables have n-grams as domain values (for a small n, here 4 or 5) and introduced the NgramMarkov constraint ensuring that the chaining of n-grams is consistent. In addition, they performed various forms of cost-based filtering on the cumulative perplexity of the sequence. The individual perplexity score of n-grams were computed offline by an LLM. Interestingly, all the filtering performed by this constraint allowed the generation process to remain manageable for a CP solver. Still in the context of natural language generation, F. Régin et al. (2024) applied *on-the-fly CP*, which gradually builds a CSP one variable at a time. They solved tasks from the COLLIE benchmark, which involve requirements such as mandatory words and constraints on the number of words or characters. To do so, they used an LLM to populate the next variable's domain with the most likely vocabulary given the text generated so far. This approach allows for controlled domain sizes but limits anticipation and filtering, particularly for long-term constraints, since some of the variables in their scope have yet to be defined. In an effort to compensate for that lack of anticipation that can lead to dead-ends and wasteful search, Bonlarron, F. Régin, et al. (2025) propose replacing the LLM by a *masked language model* (MLM) in order to predict token values both from the preceding and the following contexts. In practice they use it to preview the domain of the next few tokens, thus working with a slightly extended though still partial CSP of the sequence.

Finally, another type of inference on a trained model is to extract explanations. Here, the goal is to identify a small set of features that jointly determine the model's predictions, either exactly (Marques-Silva 2024) or with minimal error. Koriche, Lagniez, et al. (2024) focused on the general task of interpretable black-box ML models with CP. Their approach frames the problem of model-agnostic explanation as a COP: given a black-box classifier and a specific input instance, we seek an explanation of minimal error. Importantly, this CP-based approach offers guarantees for the output explanation in the same spirit as the *probably approximately correct* learning framework.

4 Limitations and Future Opportunities

Despite the growing interest and potential of the various hybrid approaches presented earlier, there remain significant limitations and fundamental challenges that must be addressed to achieve competitive performance and ensure practical applicability. The goal of this section is to outline what we identify as the key obstacles in building such hybrid systems, what has been currently achieved, and to propose a set of open research directions aimed at inspiring future work. For clarity, we distinguish between theoretical, methodological, and practical challenges. A summary of these directions is depicted in Table 1.

4.1 Theoretical Challenges

This section presents the challenges associated with the lack of theoretical understanding of several key considerations that arise when building CP+ML hybrid approaches. We believe that addressing these fundamental

Table 1. Summary of the open challenges and future opportunities described in Section 4.

Theoretical aspect	
Specific field	Challenge
General open challenge	Dealing with generalization
ML for CP (general)	Designing sufficiently expressive ML models for CP
ML for CP (general)	Identifying what to learn
ML for CP (modeling)	Learning constraint networks
ML for CP (modeling)	Designing a unified representation for learning
ML for CP (branching and reasoning)	Ensuring formal guarantees in learning-based CP systems
ML for CP (configuration)	Tailoring algorithmic configuration to CP
CP for ML (CP-based loss functions)	Integrating a discrete solver into a differentiable pipeline
Methodological aspect	
Specific field	Challenge
General open challenge	Exploring under-investigated integration strategies
ML for CP (general)	Harnessing the potential of foundation models
ML for CP (general)	Leveraging opportunities offered by large language models
ML for CP (general)	Preserving the core philosophy of constraint programming
ML for CP (branching and reasoning)	Deciding how learning should be invoked in a solver
CP for ML (CP-based inference)	Designing faster CP approaches for ML inference
Practical aspect	
Specific field	Challenge
General open challenge	Addressing the lack of appropriate benchmarks
General open challenge	Finding relevant industrial case studies
ML for CP (general)	Improving the support for learning in CP solvers

questions is essential for enabling the development of novel and promising hybrid architectures that effectively combine constraint programming and machine learning.

Dealing with generalization. A key aspect of ML is *generalization*. Training an ML model is not just about optimizing the loss over the training set, rather it is about generalizing to unseen examples. This has two major implications. First, over-optimization can lead to over-fitting; therefore, suboptimal or stochastic learning techniques, as well as proxy loss functions, are often sufficient. When improving ML with CP, this requires a careful consideration of the optimal solving capabilities of constraint programming for training, with the application-specific needs in terms of constraint satisfaction as well as generalization. This includes the objective function used and the need or non-need for finding the optimal solution versus high-quality solutions. Second, one has to be mindful of the distribution from which the training data is drawn, as well as the (possibly different) distribution on which the method is expected to perform well at test time. This is especially pronounced in improving CP with ML. While a CP solver can be expected to behave consistently across smaller and larger problems, a solver with a learning component might behave markedly different for smaller or larger problem instances due to being trained on only one of the two. There should hence be a clear distinction between approaches that aim to be *distribution-specific* (and typically application specific), versus *general-purpose*. Advances on either trajectory can

improve the other, and we expect the two to become more and more overlapping, e.g. through the development of foundation models for CP.

Designing sufficiently expressive ML models for CP. When designing a machine learning model, a key consideration is determining the appropriate level of expressivity required for the task. While simple models such as linear regression may suffice for basic tasks or situations where prediction errors are tolerable, it remains unclear what degree of expressivity is needed to effectively handle combinatorial tasks, which are often NP-hard. Although neural networks are known to be universal approximators (Cybenko 1989), achieving this generality may require an exponential number of neurons, which is not practical for many applications. Furthermore, while standard GNNs have become widely used for combinatorial reasoning, they are limited in expressive power: their discriminative capacity is bounded by the 1-dimensional Weisfeiler–Leman algorithm, making them unable to distinguish certain non-isomorphic graphs (Morris et al. 2019). Consequently, an important theoretical open question is how to determine whether a given ML model is expressive enough to solve a target combinatorial task related to CP (e.g. branching).

Identifying what to learn. This survey has illustrated that learning can augment multiple components of the CP toolbox in a variety of ways. However, across the reviewed approaches, the learning signal, whether it be labels in supervised learning, rewards in reinforcement learning, or loss functions in constraint-aware predictions, is often manually designed. This highlights a fundamental challenge: there is no clearly established ground truth for guiding the learning process in CP contexts. Automating the selection of what to learn is particularly challenging due to the multitude of potentially conflicting objectives involved in solving combinatorial problems. These objectives might include finding the best solution, maximizing pruning, minimizing the primal-dual integral, or reducing solving time. On the modeling side, there is also a tension between efficiency and interpretability, especially when comparing automatically generated models. Consequently, an open question is how to determine what information should be learned in a solver, and how to formulate this objective in a way that can effectively guide the learning process.

Learning constraint networks. Despite recent progress in learning constraint networks, several fundamental theoretical challenges remain unresolved. Current constraint acquisition methods typically require users to explicitly specify the problem variables and domains, together with a predefined constraint language from which candidate constraints are selected. However, this approach quickly becomes impractical when considering global constraints or nested expressions with potentially unbounded recursion, as the number of candidate constraints grows exponentially. On the other hand, language-free learning methods, while circumventing this combinatorial explosion, often produce constraint networks that lack interpretability, thus deviating from the user-friendly and explicit model philosophy central to CP. Moreover, in interactive constraint learning, the exact learning with (partial) membership queries imposes significant theoretical limitations on the number of queries required to accurately learn a constraint network. Extending this framework to incorporate richer forms of queries and user interaction (E. C. Freuder 2024), as well as integrating standard ML techniques to enhance generalization capabilities, represents a promising research direction. Finally, a critical open question is how to rigorously assess the quality of learned constraint networks. For the key evaluation criterion of correctness this is well understood, but perhaps other aspects should be evaluated as well, such as readability, maintainability, and efficiency of the constraint network. Addressing these theoretical challenges is essential for advancing the integration of ML and CP towards more robust, interpretable, and practically useful constraint learning frameworks.

Designing a unified representation for learning. A related question, discussed in Section 3.1.8 and also advocated in the field of neuro-symbolic methods (X. Zhang and Sheng 2024), concerns the feasibility of designing a generic representation of combinatorial problems that can be reused across different problem classes without additional modeling efforts. Despite recent progress (Boisvert et al. 2024; Marty, Boisvert, et al. 2024; Tönshoff et al. 2023;

Y. Xu et al. 2025), current approaches still face important limitations, either in terms of expressivity or in the scalability of the resulting representation. Designing such a unified and reusable representation therefore remains an open research challenge.

Ensuring formal guarantees in learning-based CP systems. Regardless of its predictive accuracy, any learning-based approach is inherently prone to prediction errors. This limitation is theoretically grounded in the notion of irreducible error, which captures the minimal level of uncertainty that no model can eliminate. In the context of combinatorial tasks, a critical question is how robust the solving process can be in the face of such errors. In many applications, we do not want to compromise essential solver properties, such as the ability to find optimal solutions or to prove optimality. While only a few of the approaches discussed in this survey offer such guarantees, they demonstrate encouraging progress. For example, backtracking can be employed to recover from poor branching decisions, and learning can serve as a sub-module within a formally valid reasoning scheme. However, these guarantees remain limited, often reduced to binary indicators (e.g. whether a property is preserved or not), without providing insight into the magnitude of the potential degradation caused by prediction errors. A valuable research direction is to establish quantitative relationships between prediction errors and solver performance, such as how errors affect the number of search nodes pruned. This would allow for more informed design choices and enable the integration of learning with clearer expectations on solver behavior under imperfect models.

Tailoring algorithmic configuration to CP. While the discussion in this survey has emphasized contributions involving or targeting CP, the vast majority of research in algorithm selection and configuration has not been tailored to CP. For example, while the design of reliable features for CSPs and COPs has received some attention, this is still only a fraction of the effort that has been put on finding good features for SAT. As a result, investigating effective features for CP problems might be a research direction with significant improvement potential. A related idea would be to design solvers that can be easily instrumented (e.g. via callbacks) to collect advanced statistics on the solution process Taking into account recent developments in the ML field, a second promising research direction would concern the development of *feature-free* approaches. The idea from Gebruers et al. (2005) of adjusting problem models, rather than just choosing or configuring algorithms, may also deserve further attention. In particular, taken to an extreme, this research direction might connect algorithm selection and configuration with approaches that learn the objective function or the constraints. Finally, in contexts where the configuration space is complex (e.g. different solvers with different sets of parameters), CP may represent a good candidate for searching for an effective configuration. This might be achieved, for example, by designing surrogate models, or local approximations, that can be efficiently handled using CP techniques.

Integrating a discrete solver into a differentiable pipeline. When learning to predict solutions, to infer the coefficients of an objective function, or to learn under constraints, a common challenge lies in integrating the solver (i.e. its constraints and objective function) into the learning process, especially in the widely used gradient-descent based optimization frameworks. While a number of general techniques exists, e.g. through differentiable optimization, perturbation-based methods or through knowledge compilation, the most suitable technique for CP with its integer variables and rich constraints has yet to emerge. Achieving such integration in a generic and principled way could have a significant impact, enabling differentiable CP models to attain some of the desirable properties such as exact reasoning, explainability, and control thanks to a combinatorial structure. J. Xu, Z. Zhang, et al. (2018) introduced the concept of *semantic loss*, a differentiable loss function that measures how well a predicted distribution over Boolean variables satisfies a logical formula by summing the probabilistic weights of all solutions to that formula. Its purpose is *"to fill in the gap between the continuous world of feedforward DNNs and the symbolic world of propositional logic"* (Gajowniczek et al. 2020). That definition can be extended to finite-domain variables. Computing the semantic loss exactly is intractable in general because the

number of solutions to σ may grow exponentially with n, the number of variables. A natural question then arises: *"Can we compute an effective approximate semantic loss tailored to CP?"*

4.2 Methodological Challenges

In contrast to the previous section, which focused on theoretical concerns, we now highlight challenges that are primarily methodological and engineering-oriented. These include issues related to the design of concrete approaches, as well as questions around how to efficiently and effectively combine ML with CP.

Exploring under-investigated integration strategies. This survey has revealed several imbalances in the way learning has been integrated into CP. Notably, prior work has focused much more on learning branching heuristics from historical data than on improving constraint propagation through learning. This trend likely stems from the fact that incorrect branching decisions can be easily corrected via backtracking, whereas incorporating learning into the filtering modules risks compromising correctness, which is far more difficult to detect and recover from. Nonetheless, we argue that learning to reason holds substantial promise in the context of CP. Recent efforts (Bessa et al. 2025; Parjadis et al. 2024) have demonstrated this potential by enhancing cost-based filtering and Lagrangian decomposition using learned components. However, these approaches remain limited to specific contexts, and it is still unclear how to generalize these ideas to broader propagation mechanisms within CP solvers. Another compelling example of imbalance is the absence of approaches combining Monte-Carlo tree search with deep learning in the context of CP. The closest related work to date is that of Antuori et al. (2021), which marks an initial step in this direction. This is particularly surprising given the remarkable success of such integrations in combinatorial games, most notably in systems like AlphaGo. We believe that exploring this direction for CP is a promising and worthwhile research avenue, as a redefinition of the standard depth-first backtracking search. Similar observations apply to the improvement of ML through CP. For example, since CP has been successfully applied to the construction of decision trees, it may also hold potential for enhancing related tree-based models, such as random forests or gradient-boosted trees.

Harnessing the potential of foundation models. Figure 1 shows that ML models are already being investigated for use in many different components of the solving process. This raises the question whether there can be such a thing as a *foundation model* for constraint programming: a representation of the problem formulation (and additional contextual data) that can be used to make predictions for all these components. That requires identifying each of these tasks, e.g. predicting satisfiability, runtime, branching strategies, value orderings, solutions. It also requires identifying a common input format, ML model architecture, an evaluation framework across each of these tasks and suitable datasets. While ambitious, any advancement across multiple tasks could lead to more generally applicable ML models in the solver.

Leveraging opportunities offered by large language models. Large language models are emerging as general-purpose assistants that can help with summarization, programming, translation, test generation and more. This offers opportunities for LLMs to help new users with formulating constraint programming models as discussed in Section 3.1.3. However, increasingly capable coding LLMs could also have potential for helping write more efficient constraint models, or to help write efficient propagators for application-specific global constraints, or to co-design new branching strategies or heuristics, suggest algorithm configuration parameters, large neighborhood strategies and more. Key in this will be to build appropriate infrastructure and datasets, to let LLM agents explore such design decisions in a controlled environment.

Preserving the core philosophy of constraint programming. Despite the clear benefits of integrating learning components into the CP toolbox, this integration risks making the technology less accessible to end users, particularly those unfamiliar with machine learning. This stands in contrast to one of the core philosophies of CP:

making combinatorial problem solving accessible and user-friendly. As such, integrating learning should be done with this in mind. On the modeling side, learning is often used to simplify the process of formulating a CP model, shifting the user's responsibility toward expressing the problem in high-level or natural language. However, this introduces new challenges, most notably, ensuring the correctness of the generated model, which is particularly difficult due to the inherent ambiguity of natural language. On the solving side, learning is typically employed to improve efficiency. Ideally, this enhancement should be transparent to the users, allowing them to benefit from improved performance without requiring additional expertise. However, this creates a significant challenge: the training phase must occur before the actual problem is known, meaning the model will likely be applied to out-of-distribution instances, where its performance may degrade. Addressing this issue is non-trivial. One possible direction is to explore online learning or fine-tuning strategies that adapt the model during solving. Another promising avenue, particularly for constraint filtering, is to train models that are specific to individual constraints and embed them directly into the implementation of those constraints, thus maintaining modularity across different models and minimizing the need for user intervention.

Deciding how learning should be invoked in a solver. A somewhat unexpected yet critical concern for learning-based CP solvers is deciding when a learning module should be invoked during the solving process. Take, for example, the case of branching decisions. A natural idea might be to use a learned model at every node in the search tree. However, given the exponential size of the tree, this would require running model inference a large number of times, which can be computationally prohibitive. Inference time generally grows with model complexity, with larger models tending to offer better performance but come at the cost of higher inference time. This trade-off has a direct impact on solver runtime. As a result, many learning-based branching heuristics surveyed here succeed in reducing the number of explored nodes, but fail to improve overall execution time. This highlights a key trade-off between model efficiency and inference cost, which was often overlooked in early work. We propose two practical recommendations to address this challenge. First, we advocate prioritizing replacing time-consuming tasks rather than lightweight ones. This aligns with our earlier observation that learning to replace or accelerate expensive filtering algorithms may offer more impactful benefits than learning-based branching heuristics, which are often computationally cheap. Second, we recommend avoiding the use of learning at every search node. Instead, it should be applied only at the top of the tree or at strategically selected nodes, where decisions are expected to have a significant impact on the structure and quality of the resulting subtrees. However, this latter suggestion raises a new research question: *"how do we identify strategic nodes where learning would have the greatest benefit?"*.

Designing faster CP approaches for ML inference. Involving CP during ML inference can unfortunately slow it down: some of the approaches surveyed require solving a COP, which can be much slower than a CSP; others involve potentially expensive weighted counting. In case of LLM inference, they have a large vocabulary, inducing large domains that need to be represented explicitly in CP. There is an opportunity here either to accelerate these approaches or to find inherently faster ones. Another issue is hardware: large neural models have billions of parameters yet run comparatively faster because they use GPUs. Advances to have CP runnable on GPUs would level the playing field (Campeotto et al. 2014). Tardivo, Dovier, et al. (2023, 2024) and Tardivo, Michel, et al. (2024) made progress in this direction by proposing the use of GPUs to accelerate the filtering and propagation of certain global constraints. Conceptually pushing this idea even further, Talbot et al. (2022) implemented a complete, though minimalist, CP solver designed specifically for execution on GPUs.

4.3 Practical Challenges

This section summarizes the challenges that are more closely tied to the current landscape of the field and the limitations of existing tools for conducting research in this area.

Addressing the lack of appropriate benchmarks. There is a shortage of standardized and widely accepted benchmarks specifically designed to evaluate learning-based approaches for CP solvers. Currently, the community primarily relies on the XCSP3 library (Audemard, Boussemart, et al. 2020) and MiniZinc (Stuckey et al. 2010). However, these frameworks are not well suited for evaluating methods that leverage historical data or learning components. As a result, research teams often resort to designing problem-specific benchmarks and generating synthetic instances tailored to their approaches. This makes fair comparisons difficult, as observed performance improvements may stem from external factors, such as larger or more tailored training datasets, rather than the methodology itself. To address this issue, we advocate for the development and adoption of benchmarks specifically designed for learning-based CP methods. A notable example of such an initiative can be found in the planning community, which introduced a learning track in the *international planning competition* (Taitler et al. 2024). This track provides standardized training and test sets, enabling a fair and reproducible evaluation of learning-based approaches.

Finding relevant industrial case studies. Despite the potential of leveraging historical data to enhance the overall efficiency of a solver, this approach introduces several challenges from a business perspective. These include the availability of sufficient and exploitable data, the need to ensure customer data privacy, and the requirement to frequently update the model with the latest data. As a result, to the best of our knowledge, only a few such hybrid approaches have been successfully deployed in industrial settings. We believe that identifying the characteristics of ideal case studies for such deployments is an important and worthwhile question to explore.

Improving the support for learning in CP solvers. Historically, the CP and ML communities have diverged in the technologies used for implementation. In pursuit of maximal performance, CP researchers have typically developed solvers in high-performance languages such as C++ (e.g. Gecode, Chuffed) or Java (e.g. Choco, MiniCP, ACE). In contrast, the ML community has favored the flexibility and ease of use offered by Python (e.g. scikit-learn, PyTorch, Keras), with performance-critical components often written in lower-level languages. This mismatch in technology stacks unfortunately increases the integration effort required to leverage state-of-the-art approaches from both fields, posing a barrier to developing efficient hybrid CP+ML systems. Several efforts have recently been made to bridge the gap between CP and ML toolchains. For example, SeaPearl (Chalumeau et al. 2021) is a CP solver implemented in Julia that integrates ML routines to learn branching decisions via reinforcement learning. However, its performance currently falls significantly short of what is achieved by traditional CP solvers. Another notable effort is CPMpy (Guns 2019), a Python-based modeling library that provides solver-agnostic access to multiple backends in a similar NumPy compatible way as ML libraries do. While it greatly improves accessibility and interoperability for ML researchers, allowing to add predictions declaratively to the problem specification and hance to reason over them; it does not support fine-grained interaction with the solver during the solving process, which limits the applicability to control the search or control other internal decision procedures/heuristics for which we could use machine learning.

5 Conclusion

Combining constraint programming and machine learning has recently attracted significant interest within the AI community. On the one hand, learning techniques can be used to automate parts of the CP modeling process, making it more accessible to users, or to accelerate the solving process, with the goal of achieving new levels of performance. On the other hand, constraint programming, thanks to its strong foundations in search, logic, and reasoning, offers a promising direction to addressing some of the fundamental limitations of machine learning approaches. The aim of this survey is to provide a comprehensive overview of recent advances at the intersection of CP and ML, with the intent of bridging and informing both research communities. While this hybridization holds great potential, it also introduces a wide range of challenges that remain open. In this survey, we have

identified these challenges and shared our perspectives on what we believe to be promising directions for future research in this rapidly evolving area. Finally, we believe that fostering a stronger presence of CP within the ML community, and reciprocally, of ML within the CP community, through dedicated workshops, tutorials, or special tracks, could promote meaningful synergies and advance both fields.

Acknowledgments

This work was initially motivated by discussions held during the Dagstuhl Seminar 24441 on Machine *Learning Augmented Algorithms for Combinatorial Optimization Problems*.

References

H. A. Addi, C. Bessiere, R. Ezzahir, and N. Lazaar. 2018. "Time-Bounded Query Generator for Constraint Acquisition." In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 1–17.

H. A. Addi and R. Ezzahir. 2019. "Pa-QUACQ: Algorithm for Constraint Acquisition System." In: *Smart Data and Computational Intelligence*. Springer International Publishing, 249–256.

B. Adenso-Diaz and M. Laguna. 2006. "Fine-tuning of algorithms using fractional experimental designs and local search." *Operations research*, 54, 1, 99–114.

C. C. Aggarwal. 2015. *Data Mining: The Textbook*. Springer Publishing Company, Incorporated. ISBN: 3319141414.

K. Ahmed, S. Teso, K.-W. Chang, G. Van den Broeck, and A. Vergari. 2022. "Semantic probabilistic layers for neuro-symbolic learning." *Advances in Neural Information Processing Systems*, 35, 29944–29959.

D. Ajwani, B. Dilkina, T. Guns, and U. C. Meyer. 2025. "Machine Learning Augmented Algorithms for Combinatorial Optimization Problems (Dagstuhl Seminar 24441)." *Dagstuhl Reports*, 14, 10, 76–100.

D. Allouche, S. De Givry, G. Katsirelos, T. Schiex, and M. Zytnicki. 2015. "Anytime hybrid best-first search with tree decomposition for weighted CSP." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 12–29.

R. Amadini, M. Gabbrielli, and J. Mauro. 2015. "SUNNY-CP: a sequential CP portfolio solver." In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 1861–1867.

R. Amadini, M. Gabbrielli, and J. Mauro. 2014. "SUNNY: a lazy portfolio approach for constraint solving." *Theory and Practice of Logic Programming*, 14, 4-5, 509–524.

D. Angluin. 1988. "Queries and concept learning." *Machine learning*, 2, 4, 319–342.

V. Antuori, E. Hébrard, M.-J. Huguet, S. Essodaigui, and A. Nguyen. 2021. "Combining Monte Carlo tree search and depth first search methods for a car manufacturing workshop scheduling problem." In: *International Conference on Principles and Practice of Constraint Programming*.

J. O. Aoga, T. Guns, and P. Schaus. 2017. "Mining time-constrained sequential patterns with constraint programming." *Constraints*, 22, 548–570.

R. Arcangioli, C. Bessiere, and N. Lazaar. 2016. "Multiple Constraint Acquisition." In: *IJCAI: International Joint Conference on Artificial Intelligence*, 698–704.

G. Audemard, F. Boussemart, C. Lecoutre, C. Piette, and O. Roussel. 2020. "XCSP3 and its ecosystem." *Constraints*, 25, 47–69.

G. Audemard, C. Lecoutre, and C. Prud'Homme. 2023. "Guiding Backtrack Search by Tracking Variables During Constraint Propagation." In: *International Conference on Principles and Practice of Constraint Programming*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

B. Babaki, T. Guns, S. Nijssen, and L. De Raedt. 2015. "Constraint-Based Querying for Bayesian Network Exploration." In: *Advances in Intelligent Data Analysis XIV*. Ed. by E. Fromont, T. De Bie, and M. van Leeuwen. Springer International Publishing, Cham, 13–24. ISBN: 978-3-319-24465-5.

Y. Bai, D. Chen, and C. P. Gomes. 2021. "CLR-DRNets: Curriculum Learning with Restarts to Solve Visual Combinatorial Games." In: *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021* (LIPIcs). Ed. by L. D. Michel. Vol. 210. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 17:1–17:14. doi:10.4230/LIPICS.CP.2021.17.

G. Bakir, T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, and S. Vishwanathan. July 2007. *Predicting Structured Data*. The MIT Press, (July 2007). doi:10.7551/mitpress/7443.001.0001.

V. Balafas, D. Tsouros, N. Ploskas, and K. Stergiou. 2024. "The Impact of Solution Diversity on Passive Constraint Acquisition." In: *Proceedings of the 13th Hellenic Conference on Artificial Intelligence*, 1–10.

V. Balafas, D. C. Tsouros, N. Ploskas, and K. Stergiou. 2024. "Enhancing Constraint Acquisition through Hybrid Learning: An Integration of Passive and Active Learning Strategies." *International Journal on Artificial Intelligence Tools*, 33, 06, 2450020.

H. Barral, M. Gaha, A. Dems, A. Côté, F. Nguewouo, and Q. Cappart. 2024. "Acquiring Constraints for a Non-linear Transmission Maintenance Scheduling Problem." In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer.

R. Barták. 1999. "Constraint programming: In pursuit of the holy grail." In: *Proceedings of the Week of Doctoral Students (WDS99)*. Vol. 4. MatFyzPress Prague, 555–564.

M. Bartlett and J. Cussens. 2017. "Integer Linear Programming for the Bayesian network structure learning problem." *Artificial Intelligence*, 244, 258–271. doi:https://doi.org/10.1016/j.artint.2015.03.003.

A. Bartolini, M. Lombardi, M. Milano, and L. Benini. 2011. "Neuron constraints to model complex real-world problems." In: *Principles and Practice of Constraint Programming–CP 2011: 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings 17*. Springer, 115–129.

D. Bekkoucha, A. Ouali, P. Boizumault, and B. Crémilleux. 2024. "Efficiently Mining Closed Interval Patterns with Constraint Programming." In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 51–67.

M.-B. Belaid, N. Belmecheri, A. Gotlieb, N. Lazaar, and H. Spieker. 2022. "GEQCA: Generic qualitative constraint acquisition." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 4. Vol. 36, 3690–3697.

M.-B. Belaid, N. Belmecheri, A. Gotlieb, N. Lazaar, and H. Spieker. 2024. "Query-driven Qualitative Constraint Acquisition." *Journal of Artificial Intelligence Research*, 79, 241–271.

N. Beldiceanu, M. Carlsson, S. Demassey, and T. Petit. 2007. "Global constraint catalogue: Past, present and future." *Constraints*, 12, 21–62.

N. Beldiceanu and H. Simonis. 2011. "A constraint seeker: Finding and ranking global constraints from examples." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 12–26.

N. Beldiceanu and H. Simonis. 2012. "A model seeker: Extracting global constraint models from positive examples." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 141–157.

N. Beldiceanu and H. Simonis. 2016. "Modelseeker: Extracting global constraint models from positive examples." In: *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*. Springer, 77–95.

N. Benabbou and T. Lust. 2019. "An interactive polyhedral approach for multi-objective combinatorial optimization with incomplete preference information." In: *International Conference on Scalable Uncertainty Management*. Springer, 221–235.

P. Benchimol, W.-J. v. Hoeve, J.-C. Régin, L.-M. Rousseau, and M. Rueher. 2012. "Improved filtering for weighted circuit constraints." *Constraints*, 17, 205–233.

Y. Bengio, A. Lodi, and A. Prouvost. 2021. "Machine learning for combinatorial optimization: a methodological tour d'horizon." *European Journal of Operational Research*, 290, 2, 405–421.

S. Bessa, D. Dabert, M. Bourgeat, L.-M. Rousseau, and Q. Cappart. 2025. "Learning Valid Dual Bounds in Constraint Programming: Boosted Lagrangian Decomposition with Self-Supervised Learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 11. Vol. 39, 11113–11121.

C. Bessiere, C. Carbonnel, A. Dries, et al.. 2023. "Learning constraints through partial queries." *Artificial Intelligence*, 319, 103896.

C. Bessiere, C. Carbonnel, and A. Himeur. 2023. "Learning constraint networks over unknown constraint languages." In: *IJCAI 2023-32nd International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 1876–1883.

C. Bessiere, R. Coletta, A. Daoudi, N. Lazaar, Y. Mechqrane, and E.-H. Bouyakhf. 2014. "Boosting Constraint Acquisition via Generalization Queries." In: *ECAI*, 99–104.

C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C.-G. Quimper, T. Walsh, et al.. 2013. "Constraint Acquisition via Partial Queries." In: *IJCAI*. Vol. 13, 475–481.

C. Bessiere, R. Coletta, F. Koriche, and B. O'Sullivan. 2005. "A SAT-based version space algorithm for acquiring constraint satisfaction problems." In: *European Conference on Machine Learning*. Springer, 23–34.

C. Bessiere, R. Coletta, B. O'Sullivan, M. Paulin, et al.. 2007. "Query-Driven Constraint Acquisition." In: *IJCAI*. Vol. 7, 50–55.

C. Bessiere, A. Daoudi, E. Hebrard, G. Katsirelos, N. Lazaar, Y. Mechqrane, N. Narodytska, C.-G. Quimper, and T. Walsh. 2016. "New approaches to constraint acquisition." *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, 51–76.

C. Bessiere, E. Hebrard, and B. O'Sullivan. 2009. "Minimising Decision Tree Size as Combinatorial Optimisation." In: *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings* (Lecture Notes in Computer Science). Ed. by I. P. Gent. Vol. 5732. Springer, 173–187. doi:10.1007/978-3-642-04244-7_16.

C. Bessiere, F. Koriche, N. Lazaar, and B. O'Sullivan. 2017. "Constraint acquisition." *Artificial Intelligence*, 244, 315–342.

B. Bischl et al.. 2016. "ASlib: A benchmark library for algorithm selection." *Artificial Intelligence*, 237, 41–58.

I. Bleukx, S. Berden, L. Coenen, N. Decleyre, and T. Guns. 2022. "Model-Based Algorithm Configuration with Adaptive Capping and Prior Distributions." In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 64–73.

L. Boisvert, H. Verhaeghe, and Q. Cappart. 2024. "Towards a Generic Representation of Combinatorial Problems for Learning-Based Approaches." In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 99–108.

A. Bonfietti, M. Lombardi, and M. Milano. 2015. "Embedding decision trees and random forests in constraint programming." In: *Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings 12*. Springer, 74–90.

A. Bonlarron, A. Calabrèse, P. Kornprobst, and J. Régin. 2023. "Constraints First: A New MDD-based Model to Generate Sentences Under Constraints." In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*. ijcai.org, 1893–1901. doi:10.24963/IJCAI.2023/210.

A. Bonlarron, F. Régin, E. D. Maria, and J.-C. Régin. 2025. "Large Language Model Meets Constraint Propagation." In: *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, 16th-22nd August 2025, Montreal, Canada*. ijcai.org.

A. Bonlarron and J. Régin. 2024. "Markov Constraint as Large Language Model Surrogate." In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*. ijcai.org, 1844–1852. https://www.ijcai.org/proceedings/2024/204.

N. Bourdache, P. Perny, and O. Spanjaard. 2020. "Bayesian preference elicitation for multiobjective combinatorial optimization." In: *DA2PL 2020-From Multiple Criteria Decision Aid to Preference Learning*.

F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. 2004. "Boosting systematic search by weighting constraints." In: *Proceedings of the 16th European Conference on Artificial Intelligence*, 146–150.

C. Boutilier, R. Brafman, C. Geib, and D. Poole. 1997. "A constraint-based approach to preference elicitation and decision making." In: *AAAI Spring Symposium on qualitative decision theory*. Citeseer, 19–28.

C. Boutilier, R. Patrascu, P. Poupart, and D. Schuurmans. 2006. "Constraint-based optimization and utility elicitation using the minimax decision criterion." *Artificial Intelligence*, 170, 8, 686–713. doi:https://doi.org/10.1016/j.artint.2006.02.003.

C. Brouard, S. de Givry, and T. Schiex. 2020. "Pushing data into CP models using graphical model learning and solving." In: *Principles and Practice of Constraint Programming: 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7–11, 2020, Proceedings 26*. Springer, 811–827.

T. Brown et al.. 2020. "Language models are few-shot learners." *Advances in neural information processing systems*, 33, 1877–1901.

L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig. 2022. "Safe learning in robotics: From learning-based control to safe reinforcement learning." *Annual Review of Control, Robotics, and Autonomous Systems*, 5, 1, 411–444.

N. H. Bshouty. 1995. "Exact learning boolean functions via the monotone theory." *Information and Computation*, 123, 1, 146–153.

N. H. Bshouty, R. Cleve, S. Kannan, and C. Tamon. 1994. "Oracles and queries that are sufficient for exact learning." In: *Proceedings of the seventh annual conference on Computational learning theory*, 130–139.

A. Burlats and G. Pesant. 2023. "Exploiting Entropy in Constraint Programming." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 20th International Conference, CPAIOR 2023, Nice, France, May 29 - June 1, 2023, Proceedings* (Lecture Notes in Computer Science). Ed. by A. A. Ciré. Vol. 13884. Springer, 320–335. doi:10.1007/978-3-031-33271-5_21.

F. Campeotto, A. Dal Palu, A. Dovier, F. Fioretto, and E. Pontelli. 2014. "Exploring the use of GPUs in constraint solving." In: *International Symposium on Practical Aspects of Declarative Languages*. Springer, 152–167.

R. Canoy, V. Bucarey, J. Mandi, and T. Guns. 2023. "Learn and route: learning implicit preferences for vehicle routing." *Constraints*, 28, 3, 363–396.

R. Canoy and T. Guns. 2019. "Vehicle routing by learning from historical solutions." In: *Principles and Practice of Constraint Programming: 25th International Conference, CP 2019, Stamford, CT, USA, September 30–October 4, 2019, Proceedings 25*. Springer, 54–70.

Q. Cappart, J. O. Aoga, and P. Schaus. 2018. "Episodesupport: A global constraint for mining frequent patterns in a long sequence of events." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*. Springer, 82–99.

Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković. 2023. "Combinatorial optimization and reasoning with graph neural networks." *Journal of Machine Learning Research*, 24, 130, 1–61.

Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, and A. A. Cire. 2021. "Combining reinforcement learning and constraint programming for combinatorial optimization." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 5. Vol. 35, 3677–3687.

M. Chabert and C. Solnon. 2017. "Constraint programming for multi-criteria conceptual clustering." In: *Principles and Practice of Constraint Programming: 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 23*. Springer, 460–476.

F. Chalumeau, I. Coulon, Q. Cappart, and L.-M. Rousseau. 2021. "Seapearl: A constraint programming solver guided by reinforcement learning." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*. Springer, 392–409.

T. C. Chan, R. Mahmood, and I. Y. Zhu. 2025. "Inverse optimization: Theory and applications." *Operations Research*, 73, 2, 1046–1074.

D. Chen, Y. Bai, W. Zhao, S. Ament, J. M. Gregoire, and C. P. Gomes. 2020. "Deep Reasoning Networks for Unsupervised Pattern De-mixing with Constraint Reasoning." In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event* (Proceedings of Machine Learning Research). Vol. 119. PMLR, 1500–1509. http://proceedings.mlr.press/v119/chen20a.html.

M. Chen et al.. 2021. "Evaluating large language models trained on code." *arXiv preprint arXiv:2107.03374*.

M. S. Cherif, D. Habet, and C. Terrioux. 2020. "On the Refinement of Conflict History Search Through Multi-Armed Bandit." In: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 264–271. doi:10.1109/ICTAI50040.2020.00050.

F. Chollet. 2019. "On the measure of intelligence." *arXiv preprint arXiv:1911.01547*.

G. Chu and P. J. Stuckey. 2012. "Inter-instance nogood learning in constraint programming." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 238–247.

G. Chu and P. J. Stuckey. 2015. "Learning value heuristics for constraint programming." In: *Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings 12*. Springer, 108–123.

J. Claes, B. Bogaerts, R. Canoy, and T. Guns. 2019. "User-oriented solving and explaining of natural language logic grid puzzles." In: *The Third Workshop on Progress Towards the Holy Grail*. Vol. 14.

M. Collautti, Y. Malitsky, D. Mehta, and B. O'Sullivan. 2013. "SNNAP: Solver-based nearest neighbor for algorithm portfolios." In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*. Springer, 435–450.

M. Collins. 2002. "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms." In: *Proceedings of the 2002 conference on empirical methods in natural language processing (EMNLP 2002)*, 1–8.

R. Coulom. 2006. "Efficient selectivity and backup operators in Monte-Carlo tree search." In: *International conference on computers and games*. Springer, 72–83.

C. Coulombe and C.-G. Quimper. 2022. "Constraint acquisition based on solution counting." In: *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 15–1.

G. Cybenko. 1989. "Approximation by superpositions of a sigmoidal function." *Mathematics of control, signals and systems*, 2, 4, 303–314.

T.-B.-H. Dao, K.-C. Duong, and C. Vrain. 2017. "Constrained clustering by constraint programming." *Artificial Intelligence*, 244, 70–94. doi:https://doi.org/10.1016/j.artint.2015.05.006.

T.-B.-H. Dao, K.-C. Duong, and C. Vrain. 2015. "Constrained minimum sum of squares clustering by constraint programming." In: *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*. Springer, 557–573.

A. Daoudi, N. Lazaar, Y. Mechqrane, C. Bessiere, and E. H. Bouyakhf. 2015. "Detecting types of variables for generalization in constraint acquisition." In: *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 413–420.

A. Daoudi, Y. Mechqrane, C. Bessiere, N. Lazaar, and E. H. Bouyakhf. 2016. "Constraint Acquisition Using Recommendation Queries." In: *IJCAI: International Joint Conference on Artificial Intelligence*, 720–726.

L. De Raedt, T. Guns, and S. Nijssen. 2010. "Constraint programming for data mining and machine learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 1. Vol. 24, 1671–1675.

L. De Raedt, A. Passerini, and S. Teso. 2018. "Learning constraints from examples." In: *Proceedings of the AAAI conference on Artificial Intelligence* 1. Vol. 32.

M. Defresne, S. Barbe, and T. Schiex. 2023. "Scalable Coupling of Deep Learning with Logical Reasoning." In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*. ijcai.org, 3615–3623. doi:10.24963/IJCAI.2023/402.

S. Demassey, G. Pesant, and L.-M. Rousseau. 2006. "A cost-regular based hybrid column generation approach." *Constraints*, 11, 4, 315–333.

F. Detassis, M. Lombardi, and M. Milano. 2021. "Teaching the old dog new tricks: Supervised learning with constraints." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 5. Vol. 35, 3742–3749.

L. Di Gaspero, A. Rendl, and T. Urli. 2013. "A hybrid ACO+ CP for balancing bicycle sharing systems." In: *International Workshop on Hybrid Metaheuristics*. Springer, 198–212.

F. Doolaard and N. Yorke-Smith. 2022. "Online learning of variable ordering heuristics for constraint optimisation problems." *Annals of Mathematics and Artificial Intelligence*, 1–30.

M. Dorigo, M. Birattari, and T. Stutzle. 2007. "Ant colony optimization." *IEEE computational intelligence magazine*, 1, 4, 28–39.

M. Dorigo, V. Maniezzo, and A. Colorni. 1996. "Ant system: optimization by a colony of cooperating agents." *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, 26, 1, 29–41.

P. Dragone, S. Teso, and A. Passerini. 2018. "Constructive preference elicitation over hybrid combinatorial spaces." In: *Proceedings of the AAAI conference on artificial intelligence* 1. Vol. 32.

P. Dragone, S. Teso, and A. Passerini. 2021. "Neuro-Symbolic Constraint Programming for Structured Prediction." In: *Proceedings of the 15th International Workshop on Neural-Symbolic Learning and Reasoning as part of the 1st International Joint Conference on Learning & Reasoning (IJCLR 2021), Virtual conference, October 25-27, 2021* (CEUR Workshop Proceedings). Ed. by A. S. d'Avila Garcez and E. Jiménez-Ruiz. Vol. 2986, 6–14.

R. van Driel, E. Demirović, and N. Yorke-Smith. 2021. "Learning variable activity initialisation for lazy clause generation solvers." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*. Springer, 62–71.

V. Dzyuba, M. van Leeuwen, and L. De Raedt. 2017. "Flexible constrained sampling with guarantees for pattern mining." *Data Mining and Knowledge Discovery*, 31, 1266–1293.

N. Efthymiou and N. Yorke-Smith. 2023. "Predicting the Optimal Period for Cyclic Hoist Scheduling Problems." In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 238–253.

J.-G. Fages and C. Prud'Homme. 2017. "Making the first solution good!" In: *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 1073–1077.

A. O. Fajemisin, D. Maragno, and D. den Hertog. 2024. "Optimization with constraint learning: A framework and survey." *European Journal of Operational Research*, 314, 1, 1–14.

F. Focacci, A. Lodi, and M. Milano. 1999. "Cost-based domain filtering." In: *Principles and Practice of Constraint Programming–CP'99: 5th International Conference, CP'99, Alexandria, VA, USA, October 11-14, 1999. Proceedings 5*. Springer, 189–203.

P. Fournier-Viger, J. C. Lin, B. Vo, T. C. Truong, J. Zhang, and H. B. Le. 2017. "A survey of itemset mining." *WIREs Data Mining Knowl. Discov.*, 7, 4. doi:10.1002/WIDM.1207.

E. Frejinger, A. Lodi, M. Lombardi, and N. Yorke-Smith. 2023. "Data-Driven Combinatorial Optimisation (Dagstuhl Seminar 22431)." *Dagstuhl Reports*, 12, 10, 166–174.

E. Freuder. 1996. "In pursuit of the holy grail." *ACM Computing Surveys (CSUR)*, 28, 4es, 63–es.

E. C. Freuder. 2024. "Conversational modeling for constraint satisfaction." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 20. Vol. 38, 22592–22597.

E. C. Freuder. 2018. "Progress towards the holy grail." *Constraints*, 23, 2, 158–171.

K. Gajowniczek, Y. Liang, T. Friedman, T. Zabkowski, and G. V. den Broeck. 2020. "Semantic and Generalized Entropy Loss Functions for Semi-Supervised Deep Learning." *Entropy*, 22(3), 334. doi:10.3390/e22030334.

A. Galassi, M. Lombardi, P. Mello, and M. Milano. 2018. "Model agnostic solution of CSPs via deep learning: A preliminary study." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*. Springer, 254–262.

A. S. Garcez, L. C. Lamb, and D. M. Gabbay. 2008. *Neural-symbolic cognitive reasoning*. Springer Science & Business Media.

M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. 2019. "Exact combinatorial optimization with graph convolutional neural networks." *Advances in neural information processing systems*, 32.

C. Gebruers, B. Hnich, D. Bridge, and E. Freuder. 2005. "Using CBR to select solution strategies in constraint programming." In: *International Conference on Case-Based Reasoning*. Springer, 222–236.

M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. 2010. "Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies." *Artificial Intelligence*, 174, 3, 270–294. doi:https://doi.org/10.1016/j.artint.2009.11.015.

I. Gent, L. Kotthoff, I. Miguel, and P. Nightingale. 2010. "Machine learning for constraint solver design–A case study for the alldifferent constraint." *arXiv preprint arXiv:1008.4326*.

P. Giadikiaroglou, M. Lymperaiou, G. Filandrianos, and G. Stamou. Nov. 2024. "Puzzle Solving using Reasoning of Large Language Models: A Survey." In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Y. Al-Onaizan, M. Bansal, and Y.-N. Chen. Association for Computational Linguistics, Miami, Florida, USA, (Nov. 2024), 11574–11591. doi:10.18653/v1/2024.emnlp-main.646.

L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. 2018. "Explaining explanations: An overview of interpretability of machine learning." In: *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE, 80–89.

D. Grimes and R. J. Wallace. 2007. "Learning to Identify Global Bottlenecks in Constraint Satisfaction Search." In: *FLAIRS*, 592–597.

L. Groleaz, S. N. Ndiaye, and C. Solnon. 2020. "Solving the Group Cumulative Scheduling Problem with CPO and ACO." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 620–636.

V. Grossi, A. Monreale, M. Nanni, D. Pedreschi, and F. Turini. 2015. "Clustering Formulation Using Constraint Optimization." In: *Software Engineering and Formal Methods*. Ed. by D. Bianculli, R. Calinescu, and B. Rumpe. Springer Berlin Heidelberg, Berlin, Heidelberg, 93–107. ISBN: 978-3-662-49224-6.

J. W. Grzymala-Busse. 2023. "Rule induction." In: *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*. Springer, 55–74.

A. Guerri and M. Milano. 2004. "Learning techniques for automatic algorithm portfolio selection." In: *ECAI*. Vol. 16, 475.

T. Guns. 2019. "Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example." In: *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*. Vol. 19.

T. Guns, T.-B.-H. Dao, C. Vrain, and K.-C. Duong. 2016. "Repetitive branch-and-bound using constraint programming for constrained minimum sum-of-squares clustering." In: *ECAI 2016*. IOS Press, 462–470.

T. Guns, A. Dries, G. Tack, S. Nijssen, and L. De Raedt. 2013. "Miningzinc: A modeling language for constraint-based mining." In: *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. AAAI Press, 1365–1372.

T. Guns, S. Nijssen, and L. De Raedt. 2011. "Itemset mining: A constraint programming perspective." *Artificial Intelligence*, 175, 12-13, 1951–1983.

T. Guns, S. Nijssen, and L. De Raedt. 2013. "k-Pattern Set Mining under Constraints." *IEEE Transactions on Knowledge and Data Engineering*, 25, 2, 402–418. doi:10.1109/TKDE.2011.204.

M. H. Hà, C.-G. Quimper, and L.-M. Rousseau. 2015. "General bounding mechanism for constraint programs." In: *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*. Springer, 158–172.

D. Habet and C. Terrioux. 2019. "Conflict history based search for constraint satisfaction problem." In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 1117–1122.

M. N. Haouas, D. Aloise, and G. Pesant. 2020. "An Exact CP Approach for the Cardinality-Constrained Euclidean Minimum Sum-of-Squares Clustering Problem." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21-24, 2020, Proceedings* (Lecture Notes in Computer Science). Ed. by E. Hebrard and N. Musliu. Vol. 12296. Springer, 256–272. doi:10.1007/978-3-030-58942-4_17.

A. Hien, N. Aribi, S. Loudni, Y. Lebbah, A. Ouali, and A. Zimmermann. 2024. "Mining diverse sets of patterns with constraint programming using the pairwise Jaccard similarity relaxation." *Constraints*, 1–32.

B. Hurley, L. Kotthoff, Y. Malitsky, and B. O'Sullivan. 2014. "Proteus: A hierarchical portfolio of solvers and transformations." In: *Integration of AI and OR Techniques in Constraint Programming: 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings 11*. Springer, 301–317.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. 2011. "Sequential model-based optimization for general algorithm configuration." In: *Learning and intelligent optimization: 5th international conference, LION 5, rome, Italy, January 17-21, 2011. selected papers 5*. Springer, 507–523.

H. Islah and Y. Mechqrane. 2024. "Machine Learning Based Recommendation Queries for Constraint Acquisition." In: *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 159–165.

E. Jabrayilzade and S. Tekir. Nov. 2020. "LGPSolver - Solving Logic Grid Puzzles Automatically." In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Ed. by T. Cohn, Y. He, and Y. Liu. Association for Computational Linguistics, Online, (Nov. 2020), 1118–1123. doi:10.18653/v1/2020.findings-emnlp.100.

J. Jumper et al.. 2021. "Highly accurate protein structure prediction with AlphaFold." *Nature*, 596, 7873, 583–589.

V. Jung and J.-C. Régin. 2021. "Checking constraint satisfaction." In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 332–347.

S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. 2011. "Algorithm selection and scheduling." In: *Principles and Practice of Constraint Programming–CP 2011: 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings 17*. Springer, 454–469.

S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. 2010. "ISAC–Instance-Specific Algorithm Configuration." In: *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, 751–756.

S. Kadıoğlu, P. Pravin Dakle, K. Uppuluri, R. Politi, P. Raghavan, S. Rallabandi, and R. Srinivasamurthy. Nov. 2024. "Ner4Opt: named entity recognition for optimization modelling from natural language." *Constraints*, 29, 3, (Nov. 2024), 261–299. doi:10.1007/s10601-024-09376-5.

H. Kashgarani and L. Kotthoff. 2023. "Automatic Parallel Portfolio Selection." In: *ECAI 2023*. IOS Press, 1215–1222.

A. Kemmar, S. Loudni, Y. Lebbah, P. Boizumault, and T. Charnois. 2015. "PREFIX-PROJECTION global constraint for sequential pattern mining." In: *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*. Springer, 226–243.

P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. 2019. "Automated algorithm selection: Survey and perspectives." *Evolutionary computation*, 27, 1, 3–45.

M. Khiari, P. Boizumault, and B. Crémilleux. 2010. "Constraint programming for mining n-ary patterns." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 552–567.

M. Khichane, P. Albert, and C. Solnon. 2008. "Integration of ACO in a constraint programming language." In: *International Conference on Ant Colony Optimization and Swarm Intelligence*. Springer, 84–95.

M. Khichane, P. Albert, and C. Solnon. 2010. "Strong combination of ant colony optimization with constraint programming optimization." In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 232–245.

Z. Kiziltan, M. Lippi, and P. Torroni. 2016. "Constraint detection in natural language problem descriptions." In: *IJCAI*. Vol. 2016. International Joint Conferences on Artificial Intelligence, 744–750.

J. N. Kok, E. Marchiori, M. Marchiori, and C. Rossi. 1996. *Evolutionary training of CLP-constrained neural networks*. Practical Application Company.

D. Koller and N. Friedman. 2009. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press. ISBN: 978-0-262-01319-2. http://mitpress.mit.edu/catalog/item/default.asp?ttype=2%5C&tid=11886.

M. Koptelov, A. Zimmermann, P. Boizumault, R. Bureau, and J.-L. Lamotte. 2023. "Explanations for Itemset Mining by Constraint Programming: A Case Study Using ChEMBL Data." In: *International Symposium on Intelligent Data Analysis*. Springer, 208–221.

F. Koriche, J.-M. Lagniez, S. Mengel, and C. Tran. 2024. "Learning Model Agnostic Explanations via Constraint Programming." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 437–453.

F. Koriche, C. Lecoutre, A. Paparrizou, and H. Wattez. 2022. "Best heuristic identification for constraint satisfaction." In: *31st International Joint Conference on Artificial Intelligence (IJCAI'22)*. International Joint Conferences on Artificial Intelligence Organization, 1859–1865.

J. Kotary, F. Fioretto, P. van Hentenryck, and B. Wilder. 2021. "End-to-End Constrained Optimization Learning: A Survey." In: *30th International Joint Conference on Artificial Intelligence, IJCAI 2021*. International Joint Conferences on Artificial Intelligence, 4475–4482.

L. Kotthoff. 2016. "Algorithm selection for combinatorial search problems: A survey." In: *Data mining and constraint programming: Foundations of a cross-disciplinary approach*. Springer, 149–190.

M. Kumar, S. Kolb, and T. Guns. 2022. "Learning constraint programming models from data using generate-and-aggregate." In: *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

C.-T. Kuo, S. Ravi, C. Vrain, and I. Davidson. 2017. "A framework for minimal clustering modification via constraint programming." In: *Proceedings of the AAAI conference on artificial intelligence* 1. Vol. 31.

E. Kuş, Ö. Akgün, N. Dang, and I. Miguel. 2024. "Frugal Algorithm Selection." In: *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)* (Leibniz International Proceedings in Informatics (LIPIcs)). Ed. by P. Shaw. Vol. 307. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 38:1–38:16. ISBN: 978-3-95977-336-2. doi:10.4230/LIPIcs.CP.2024.38.

P. Laborie and D. Godard. 2007. "Self-adapting large neighborhood search: Application to single-mode scheduling problems." *Proceedings MISTA-07, Paris*, 8.

P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. 2018. "IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG." *Constraints*, 23, 210–250.

D. Lafleur, S. Chandar, and G. Pesant. 2022. "Combining Reinforcement Learning and Constraint Programming for Sequence-Generation Tasks with Hard Constraints." In: *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel* (LIPIcs). Ed. by C. Solnon. Vol. 235. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 30:1–30:16. doi:10.4230/LIPICS.CP.2022.30.

A. Lallouet, M. Lopez, L. Martin, and C. Vrain. 2010. "On learning constraint problems." In: *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*. Vol. 1. IEEE, 45–52.

C. Lawless, J. Schoeffer, L. Le, K. Rowan, S. Sen, C. St. Hill, J. Suh, and B. Sarrafzadeh. 2024. ""I Want It That Way": Enabling Interactive Decision Support Using Large Language Models and Constraint Programming." *ACM Transactions on Interactive Intelligent Systems*, 14, 3, 1–33.

N. Lazaar. 2021. "Parallel constraint acquisition." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 5. Vol. 35, 3860–3867.

N. Lazaar, Y. Lebbah, S. Loudni, M. Maamar, V. Lemière, C. Bessiere, and P. Boizumault. 2016. "A global constraint for closed frequent pattern mining." In: *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings 22*. Springer, 333–349.

Y. LeCun, Y. Bengio, and G. Hinton. 2015. "Deep learning." *Nature*, 521, 7553, 436–444.

J.-H. Lee and H.-J. Kim. 2022. "Imitation learning for real-time job shop scheduling using graph-based representation." In: *2022 Winter Simulation Conference (WSC)*. IEEE, 3285–3296.

H. Li, Y. Wu, M. Yin, and Z. Li. 2022. "A portfolio-based approach to select efficient variable ordering heuristics for constraint satisfaction problems." In: *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

H. Li, M. Yin, and Z. Li. 2021. "Failure based variable ordering heuristics for solving CSPs." In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

B. Y. Lin, R. L. Bras, K. Richardson, A. Sabharwal, R. Poovendran, P. Clark, and Y. Choi. 2025. "ZebraLogic: On the Scaling Limits of LLMs for Logical Reasoning." In: *Forty-second International Conference on Machine Learning*.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. 2022. "SMAC3: A versatile Bayesian optimization package for hyperparameter optimization." *Journal of Machine Learning Research*, 23, 54, 1–9.

M. Lindauer, H. Hoos, and F. Hutter. 2015. "From sequential algorithm selection to parallel portfolio selection." In: *International Conference on Learning and Intelligent Optimization*. Springer, 1–16.

T. Liu, R. Amadini, M. Gabbrielli, and J. Mauro. 2021. "sunny-as2: Enhancing SUNNY for algorithm selection." *Journal of Artificial Intelligence Research*, 72, 329–376.

A. Lodi and G. Zarpellon. 2017. "On learning and branching: a survey." *Top*, 25, 207–236.

M. Lombardi and S. Gualandi. 2016. "A lagrangian propagator for artificial neural networks in constraint programming." *Constraints*, 21, 435–462.

M. Lombardi and M. Milano. 2018. "Boosting combinatorial problem modeling with machine learning." In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 5472–5478.

M. Lombardi, M. Milano, and A. Bartolini. 2017. "Empirical decision model learning." *Artificial Intelligence*, 244, 343–367.

A. Loreggia, Y. Malitsky, H. Samulowitz, and V. Saraswat. 2016. "Deep learning for algorithm portfolios." In: *Proceedings of the AAAI conference on artificial intelligence* 1. Vol. 30.

M. Loth, M. Sebag, Y. Hamadi, and M. Schoenauer. 2013. "Bandit-based search for constraint programming." In: *Principles and Practice of Constraint Programming: 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings 19*. Springer, 464–480.

M. Loth, M. Sebag, Y. Hamadi, M. Schoenauer, and C. Schulte. 2013. "Hybridizing constraint programming and monte-carlo tree search: Application to the job shop problem." In: *Learning and Intelligent Optimization: 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers 7*. Springer, 315–320.

A. K. Mackworth. 1977. "Consistency in networks of relations." *Artificial intelligence*, 8, 1, 99–118.
Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. 2012. "Parallel SAT solver selection and scheduling." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 512–526.
J. Mandi, V. Bucarey, M. M. K. Tchomba, and T. Guns. 2022. "Decision-focused learning: Through the lens of learning to rank." In: *International conference on machine learning*. PMLR, 14935–14947.
J. Mandi, R. Canoy, V. Bucarey, and T. Guns. 2021. "Data Driven VRP: A Neural Network Model to Learn Hidden Preferences for VRP." In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.
J. Mandi, J. Kotary, S. Berden, M. Mulamba, V. Bucarey, T. Guns, and F. Fioretto. 2024. "Decision-focused learning: Foundations, state of the art, benchmark and future opportunities." *Journal of Artificial Intelligence Research*, 80, 1623–1701.
V. Manibod, D. Saikali, and G. Pesant. 2025. "Constrained Sequential Inference in Machine Learning Using Constraint Programming." In: *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, 16th-22nd August 2025, Montreal, Canada*. ijcai.org.
J. Marques-Silva. 2024. "Logic-based explainability: past, present and future." In: *International Symposium on Leveraging Applications of Formal Methods*. Springer, 181–204.
J. Marques-Silva, I. Lynce, and S. Malik. 2021. "Conflict-driven clause learning SAT solvers." In: *Handbook of satisfiability*. ios Press, 133–182.
T. Marty, L. Boisvert, T. François, P. Tessier, L. Gautier, L.-M. Rousseau, and Q. Cappart. 2024. "Learning and fine-tuning a generic value-selection heuristic inside a constraint programming solver." *Constraints*, 1–27.
T. Marty, T. François, P. Tessier, L. Gautier, L.-M. Rousseau, and Q. Cappart. 2023. "Learning a Generic Value-Selection Heuristic Inside a Constraint Programming Solver." In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
Y. Mechqrane, C. Bessiere, and I. Elabbassi. 2024. "Using large language models to improve query-based constraint acquisition." In: *IJCAI 2024-33rd International Joint Conference on Artificial Intelligence*, 1916–1925.
M.-A. Ménard, M. Morin, M. Khachan, J. Gaudreault, and C.-G. Quimper. 2023. "Learn, Compare, Search: One Sawmill's Search for the Best Cutting Patterns Across and/or Trees." In: *International Conference on Learning and Intelligent Optimization*. Springer, 552–566.
G. Menguy, S. Bardin, A. Gotlieb, and N. Lazaar. 2025. "A Query-Based Constraint Acquisition Approach for Enhanced Precision in Program Precondition Inference." *Journal of Artificial Intelligence Research*, 82, 901–936.
G. Menguy, S. Bardin, N. Lazaar, and A. Gotlieb. 2023. "Active disjunctive constraint acquisition." In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* 1. Vol. 19, 512–520.
B. Meyer and A. Ernst. 2004. "Integrating ACO and constraint propagation." In: *International Workshop on Ant Colony Optimization and Swarm Intelligence*. Springer, 166–177.
K. Michailidis, D. Tsouros, and T. Guns. 2024. "Constraint modelling with LLMs using in-context learning." In: *30th International conference on principles and practice of constraint programming*.
K. Michailidis, D. Tsouros, and T. Guns. 2025. "CP-Bench: Evaluating Large Language Models for Constraint Modelling." *arXiv preprint arXiv:2506.06052*.
L. Michel and P. Van Hentenryck. 2012. "Activity-based search for black-box constraint programming solvers." In: *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimization Problems: 9th International Conference, CPAIOR 2012, Nantes, France, May 28–June 1, 2012. Proceedings 9*. Springer, 228–243.
S. Minton. 1996. "Automatically configuring constraint satisfaction programs: A case study." *Constraints*, 1, 7–43.
T. M. Mitchell. 1997. *Machine learning*. 9. Vol. 1. McGraw-hill New York.
C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. 2019. "Weisfeiler and Leman go neural: Higher-order graph neural networks." In: *Proceedings of the AAAI conference on artificial intelligence* 01. Vol. 33, 4602–4609.
M. Mouhoub, H. Al Marri, and E. Alanazi. 2023. "Exact learning of qualitative constraint networks from membership queries." *International Journal of Software Engineering and Knowledge Engineering*, 33, 06, 837–863.
M. Mulamba, J. Mandi, R. Canoy, and T. Guns. 2020. "Hybrid Classification and Reasoning for Image-Based Constraint Solving." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21-24, 2020, Proceedings* (Lecture Notes in Computer Science). Ed. by E. Hebrard and N. Musliu. Vol. 12296. Springer, 364–380. doi:10.1007/978-3-030-58942-4_24.
M. Mulamba, J. Mandi, A. I. Mahmutogullari, and T. Guns. 2024. "Perception-based constraint solving for sudoku images." *Constraints*, 29, 1, 112–151.
K. P. Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT press.
K. P. Murphy. 2023. *Probabilistic Machine Learning: Advanced Topics*. MIT Press. http://probml.github.io/book2.
K. P. Murphy. 2022. *Probabilistic Machine Learning: An introduction*. MIT Press. http://probml.github.io/book1.
B. Negrevergne and T. Guns. 2015. "Constraint-based sequence mining using constraint programming." In: *Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings 12*. Springer, 288–305.

N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. 2007. "MiniZinc: Towards a standard CP modelling language." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 529–543.

E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan. 2008. "Using case-based reasoning in an algorithm portfolio for constraint solving." In: *Irish conference on artificial intelligence and cognitive science*, 210–216.

O. Ohrimenko, P. J. Stuckey, and M. Codish. 2009. "Propagation via lazy clause generation." *Constraints*, 14, 357–391.

F. Pachet and P. Roy. 2011. "Markov constraints: steerable generation of Markov sequences." *Constraints*, 16, 2, 148–172.

A. Parjadis, Q. Cappart, B. Dilkina, A. Ferber, and L.-M. Rousseau. 2024. "Learning Lagrangian Multipliers for the Travelling Salesman Problem." In: *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)* (Leibniz International Proceedings in Informatics (LIPIcs)). Ed. by P. Shaw. Vol. 307. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 22:1–22:18. ISBN: 978-3-95977-336-2. doi:10.4230/LIPIcs.CP.2024.22.

J. Pearl. 1982. "Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach." In: *Proceedings of the National Conference on Artificial Intelligence. Pittsburgh, PA, August 18-20, 1982.* 133–136. http://www.aaai.org/Library/AAAI/1982/aaai82-032.php.

G. Pesant. 2004. "A regular language membership constraint for finite sequences of variables." In: *International conference on principles and practice of constraint programming*. Springer, 482–495.

G. Pesant. 2015. "Achieving Domain Consistency and Counting Solutions for Dispersion Constraints." *INFORMS J. Comput.*, 27, 4, 690–703. doi:10.1287/IJOC.2015.0654.

G. Pesant. 2019. "From support propagation to belief propagation in constraint programming." *Journal of Artificial Intelligence Research*, 66, 123–150.

G. Pesant and J.-C. Régin. 2005. "Spread: A balancing constraint based on statistics." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 460–474.

É. Picard-Cantin, M. Bouchard, C.-G. Quimper, and J. Sweeney. 2016. "Learning parameters for the sequence constraint from solutions." In: *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings 22*. Springer, 405–420.

É. Picard-Cantin, M. Bouchard, C.-G. Quimper, and J. Sweeney. 2017. "Learning the parameters of global constraints using branch-and-bound." In: *Principles and Practice of Constraint Programming: 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 23*. Springer, 512–528.

A. Popescu, S. Polat-Erdeniz, A. Felfernig, M. Uta, M. Atas, V.-M. Le, K. Pilsl, M. Enzelsberger, and T. N. T. Tran. 2022. "An overview of machine learning techniques in constraint solving." *Journal of Intelligent Information Systems*, 58, 1, 91–118.

S. Prestwich. 2022. "Extrapolating Constraint Networks by Symbolic Classification." In: *5th IJCAI Workshop on Data Science Meets Optimization*.

S. Prestwich. 2020. "Robust constraint acquisition by sequential analysis." In: *ECAI 2020*. IOS Press, 355–362.

S. Prestwich. 2021. "Unsupervised constraint acquisition." In: *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 256–262.

S. Prestwich and N. Wilson. 2024. "A statistical approach to learning constraints." *International Journal of Approximate Reasoning*, 109184.

S. D. Prestwich, E. C. Freuder, B. O'Sullivan, and D. Browne. 2021. "Classifier-based constraint acquisition." *Annals of Mathematics and Artificial Intelligence*, 89, 7, 655–674.

C. Prud'homme and J.-G. Fages. 2022. "Choco-solver." *Journal of Open Source Software*, 7, 78, 4708.

J.-F. Puget. 2004. "Constraint programming next challenge: Simplicity of use." In: *Principles and Practice of Constraint Programming–CP 2004: 10th International Conference, CP 2004, Toronto, Canada, September 27-October 1, 2004. Proceedings 10*. Springer, 5–8.

D. Pulatov, M. Anastacio, L. Kotthoff, and H. Hoos. 2022. "Opening the black box: Automated software analysis for algorithm selection." In: *International Conference on Automated Machine Learning*. PMLR, 6–1.

R. Ramamonjison et al.. 2023. "NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions." In: *NeurIPS 2022 Competition Track*. PMLR, 189–203.

P. Refalo. 2004. "Impact-based search strategies for constraint programming." In: *Principles and Practice of Constraint Programming–CP 2004: 10th International Conference, CP 2004, Toronto, Canada, September 27-October 1, 2004. Proceedings 10*. Springer, 557–571.

F. Régin, E. D. Maria, and A. Bonlarron. 2024. "Combining Constraint Programming Reasoning with Large Language Model Predictions." In: *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2-6, 2024, Girona, Spain* (LIPIcs). Ed. by P. Shaw. Vol. 307. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 25:1–25:18. doi:10.4230/LIPICS.CP.2024.25.

J.-C. Régin. 2011. "Solving problems with CP: Four common pitfalls to avoid." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 3–11.

J. R. Rice. 1976. "The algorithm selection problem." In: *Advances in computers*. Vol. 15. Elsevier, 65–118.

F. Rossi, P. Van Beek, and T. Walsh. 2006. *Handbook of constraint programming*. Elsevier.

U. Sadana, A. Chenreddy, E. Delage, A. Forel, E. Frejinger, and T. Vidal. 2025. "A survey of contextual optimization methods for decision-making under uncertainty." *European Journal of Operational Research*, 320, 2, 271–289.

L. Scavuzzo, K. Aardal, A. Lodi, and N. Yorke-Smith. 2024. "Machine learning augmented branch and bound for mixed integer linear programming." *Mathematical Programming*, 1–44.

P. Schaus, J. O. Aoga, and T. Guns. 2017. "Coversize: A global constraint for frequency-based itemset mining." In: *Principles and Practice of Constraint Programming: 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 23*. Springer, 529–546.

P. Schaus, Y. Deville, and P. Dupont. 2007. "Bound-Consistent Deviation Constraint." In: *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings* (Lecture Notes in Computer Science). Ed. by C. Bessiere. Vol. 4741. Springer, 620–634. doi:10.1007/978-3-540-74970-7_44.

P. Schaus, Y. Deville, P. Dupont, and J.-C. Régin. 2007. "The deviation constraint." In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 260–274.

P. Schaus and J. Régin. 2014. "Bound-consistent spread constraint." *EURO Journal on Computational Optimization*, 2, 3, 123–146. doi:10.1007/S13675-013-0018-8.

E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney. 2022. "A survey of methods for automated algorithm configuration." *Journal of Artificial Intelligence Research*, 75, 425–487.

T. Schiex, H. Fargier, and G. Verfaillie. 1995. "Valued constraint satisfaction problems: Hard and easy problems." *IJCAI (1)*, 95, 631–639.

C. Schulte, M. Lagerkvist, and G. Tack. 2006. "Gecode." *Software download and online material at the website: http://www.gecode.org*, 11–13.

M. Sellmann. 2004. "Theoretical foundations of CP-based lagrangian relaxation." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 634–647.

P. Shaw. 1998. "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems." In: *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings* (Lecture Notes in Computer Science). Ed. by M. J. Maher and J. Puget. Vol. 1520. Springer, 417–431. doi:10.1007/3-540-49481-2_30.

M. Shishmarev, C. Mears, G. Tack, and M. Garcia de la Banda. 2016. "Learning from learning solvers." In: *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings 22*. Springer, 455–472.

P. Shivaswamy and T. Joachims. 2015. "Coactive learning." *Journal of Artificial Intelligence Research*, 53, 1–40.

P. Shojaee, I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar. 2025. *The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity*. (2025). https://ml-site.cdn-apple.com/papers/the-illusion-of-thinking.pdf.

D. Silver et al.. 2016. "Mastering the game of Go with deep neural networks and tree search." *Nature*, 529, 7587, 484–489.

M. Silvestri, M. Lombardi, and M. Milano. 2021. "Injecting Domain Knowledge in Neural Networks: A Controlled Experiment on a Constrained Problem." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5-8, 2021, Proceedings* (Lecture Notes in Computer Science). Ed. by P. J. Stuckey. Vol. 12735. Springer, 266–282. doi:10.1007/978-3-030-78230-6_17.

A. Singirikonda, S. Kadioglu, and K. Uppuluri. 2025. "Text2Zinc: A Cross-Domain Dataset for Modeling Optimization and Satisfaction Problems in MiniZinc." *arXiv preprint arXiv:2503.10642*.

K. A. Smith. 1999. "Neural networks for combinatorial optimization: a review of more than a decade of research." *Informs journal on Computing*, 11, 1, 15–34.

C. Solnon. 2010. *Ant colony optimization and constraint programming*. Wiley Online Library.

W. Song, Z. Cao, J. Zhang, C. Xu, and A. Lim. 2022. "Learning variable ordering heuristics for solving constraint satisfaction problems." *Engineering Applications of Artificial Intelligence*, 109, 104603.

Y. Song and E. Cohen. 2025. "Do LLMs Understand Constraint Programming? Zero-Shot Constraint Programming Model Generation Using LLMs." In: *The 19th Learning and Intelligent Optimization Conference (LION19)*.

P. J. Stuckey, R. Becket, and J. Fischer. 2010. "Philosophy of the MiniZinc challenge." *Constraints*, 15, 307–316.

Y. Sun, S. Nguyen, D. Thiruvady, X. Li, A. T. Ernst, and U. Aickelin. 2024. "Enhancing constraint programming via supervised learning for job shop scheduling." *Knowledge-Based Systems*, 293, 111698.

A. Taitler et al.. 2024. "The 2023 international planning competition." In: Wiley Online Library.

P. Talbot, F. G. Pinel, and P. Bouvry. 2022. "A Variant of Concurrent Constraint Programming on GPU." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 4. Vol. 36, 3830–3839.

F. Tardivo, A. Dovier, A. Formisano, L. Michel, and E. Pontelli. 2023. "Constraint propagation on GPU: A case study for the AllDifferent constraint." *Journal of Logic and Computation*, 33, 8, 1734–1752.

F. Tardivo, A. Dovier, A. Formisano, L. Michel, and E. Pontelli. 2024. "Constraint propagation on GPU: A case study for the cumulative constraint." *Constraints*, 1–23.

F. Tardivo, L. Michel, and E. Pontelli. 2024. "CP for Bin Packing with Multi-Core and GPUs." In: *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 28–1.

P. Tassel, M. Gebser, and K. Schekotihin. 2023. "An end-to-end reinforcement learning approach for job-shop scheduling problems based on constraint programming." In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 33, 614–622.

F. Teichteil-Königsbuch, G. Povéda, G. G. de Garibay Barba, T. Luchterhand, and S. Thiébaux. 2023. "Fast and robust resource-constrained scheduling with graph neural networks." In: *Proceedings of the International Conference on Automated Planning and Scheduling* 1. Vol. 33, 623–633.

S. Teso, A. Passerini, and P. Viappiani. 2016. "Constructive preference elicitation by setwise max-margin learning." In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2067–2073.

C. Thomas and P. Schaus. 2018. "Revisiting the self-adaptive large neighborhood search." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*. Springer, 557–566.

J. Tönshoff, B. Kisin, J. Lindner, and M. Grohe. 2023. "One model, any CSP: graph neural networks as fast global search heuristics for constraint satisfaction." In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 4280–4288.

R. Toro Icarte, L. Illanes, M. P. Castro, A. A. Cire, S. A. McIlraith, and J. C. Beck. 2019. "Training binarized neural networks using MIP and CP." In: *Principles and Practice of Constraint Programming: 25th International Conference, CP 2019, Stamford, CT, USA, September 30–October 4, 2019, Proceedings 25*. Springer, 401–417.

F. Trösser, S. de Givry, and G. Katsirelos. Aug. 2021. "Improved Acyclicity Reasoning for Bayesian Network Structure Learning with Constraint Programming." In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Z.-H. Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, (Aug. 2021), 4250–4257. doi:10.24963/ijcai.2021/584.

D. Tsouros, S. Berden, S. Prestwich, and T. Guns. 2025. "Generalizing Constraint Models in Constraint Acquisition." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 11. Vol. 39, 11362–11371.

D. Tsouros, S. Berden, and T. Guns. 2024. "Learning to learn in interactive constraint acquisition." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 8. Vol. 38, 8154–8162.

D. C. Tsouros, S. Berden, and T. Guns. 2023. "Guided Bottom-Up Interactive Constraint Acquisition." In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 36–1.

D. C. Tsouros and K. Stergiou. 2020. "Efficient multiple constraint acquisition." *Constraints*, 25, 3, 180–225.

D. C. Tsouros and K. Stergiou. 2021. "Learning Max-CSPs via active constraint acquisition." In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

D. C. Tsouros, K. Stergiou, and C. Bessiere. 2020. "Omissions in Constraint Acquisition." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 935–951.

D. C. Tsouros, K. Stergiou, and C. Bessiere. 2019. "Structure-Driven Multiple Constraint Acquisition." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 709–725.

D. C. Tsouros, K. Stergiou, and P. G. Sarigiannidis. 2018. "Efficient Methods for Constraint Acquisition." In: *24th International Conference on Principles and Practice of Constraint Programming*.

W. Ugarte, P. Boizumault, B. Crémilleux, A. Lepailleur, S. Loudni, M. Plantevit, C. Raıˆssi, and A. Soulet. 2017. "Skypattern mining: From pattern condensed representations to dynamic constraint satisfaction problems." *Artificial Intelligence*, 244, 48–69.

P. Van Beek and H.-F. Hoffmann. 2015. "Machine learning of Bayesian networks using constraint programming." In: *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*. Springer, 429–445.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017. "Attention is all you need." *Advances in neural information processing systems*, 30.

B. Véjar, G. Aglin, A. İ. Mahmutoğulları, S. Nijssen, P. Schaus, and T. Guns. 2024. "An Efficient Structured Perceptron for NP-Hard Combinatorial Optimization Problems." In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 253–262.

M. Veksler and O. Strichman. 2016. "Learning general constraints in CSP." *Artificial Intelligence*, 238, 135–153.

H. Verhaeghe, S. Nijssen, G. Pesant, C.-G. Quimper, and P. Schaus. 2020. "Learning optimal decision trees using constraint programming." *Constraints*, 25, 226–250.

H. Verhaeghe, Q. Cappart, G. Pesant, and C.-G. Quimper. 2024. "Learning Precedences for Scheduling Problems with Graph Neural Networks." In: *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

D. L. Waltz. 1975. "Understanding line drawings of scenes with shadows." *The psychology of computer vision*, 19–91.

P. Wang, P. L. Donti, B. Wilder, and J. Z. Kolter. 2019. "SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver." In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (Proceedings of Machine Learning Research). Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. PMLR, 6545–6554. http://proceedings.mlr.press/v97/wang19e.html.

R. Wang, W. Xia, and R. H. Yap. 2017. "Correlation heuristics for constraint programming." In: *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 1037–1041.

H. Wattez, F. Koriche, C. Lecoutre, A. Paparrizou, and S. Tabary. 2020. "Learning variable ordering heuristics with multi-armed bandits and restarts." In: *ECAI 2020*. IOS Press, 371–378.

H. Wattez, C. Lecoutre, A. Paparrizou, and S. Tabary. 2019. "Refining constraint weighting." In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 71–77.

B. Wiegand, D. Klakow, and J. Vreeken. 2024. "What are the rules? Discovering constraints from data." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 8. Vol. 38, 8182–8190.

W. Xia and R. Yap. 2018. "Learning robust search strategies using a bandit-based approach." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 1. Vol. 32.

H. Xu, S. Koenig, and T. S. Kumar. 2018. "Towards effective deep learning for constraint satisfaction problems." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 588–597.

J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. V. den Broeck. 2018. "A Semantic Loss Function for Deep Learning with Symbolic Knowledge." In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (Proceedings of Machine Learning Research). Ed. by J. G. Dy and A. Krause. Vol. 80. PMLR, 5498–5507. http://proceedings.mlr.press/v80/xu18h.html.

J. Xu, Y. Wu, H. Li, and M. Yin. 2025. "Prediction-Based Adaptive Variable Ordering Heuristics for Constraint Satisfaction Problems." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 11. Vol. 39, 11390–11398.

Y. Xu, W. Li, S. Sanner, and E. B. Khalil. 2025. "Self-Supervised Transformers as Iterative Solution Improvers for Constraint Satisfaction." In: *Forty-second International Conference on Machine Learning*.

C. Yin, Q. Cappart, and G. Pesant. 2024. "An Improved Neuro-Symbolic Architecture to Fine-Tune Generative AI Systems." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part II* (Lecture Notes in Computer Science). Ed. by B. Dilkina. Vol. 14743. Springer, 279–288. doi:10.1007/978-3-031-60599-4_19.

C. Yin, Q. Cappart, and G. Pesant. 2025. "Shaping Reward Signals in Reinforcement Learning Using Constraint Programming." In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 22nd International Conference, CPAIOR 2025, Melbourne, Australia, November 10-13, 2025, Proceedings, Part II* (Lecture Notes in Computer Science). Ed. by G. Tack. Vol. 15763. Springer.

K. Zeighami, K. Leo, G. Tack, and M. G. de la Banda. 2018. "Towards semi-automatic learning-based model transformation." In: *Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24*. Springer, 403–419.

X. Zhang and V. S. Sheng. 2024. "Neuro-symbolic AI: Explainability, challenges, and future trends." *arXiv preprint arXiv:2411.04383*.

K. Zheng, A. Li, H. Zhang, and T. Kumar. 2023. "FastMapSVM for Predicting CSP Satisfiability." In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

Received 13 June 2025; accepted 5 November 2025