An overview of machine learning techniques in constraint solving

Andrei Popescu¹ · Seda Polat-Erdeniz¹ · Alexander Felfernig¹ · Mathias Uta² · Müslüm Atas¹ · Viet-Man Le¹ · Klaus Pils³ · Martin Enzelsberger³ · Thi Ngoc Trang Tran¹

Received: 16 April 2021 / Revised: 5 August 2021 / Accepted: 9 August 2021 /
Published online: 30 August 2021
© The Author(s) 2021

Abstract

Constraint solving is applied in different application contexts. Examples thereof are the configuration of complex products and services, the determination of production schedules, and the determination of recommendations in online sales scenarios. Constraint solvers apply, for example, search heuristics to assure adequate runtime performance and prediction quality. Several approaches have already been developed showing that machine learning (ML) can be used to optimize search processes in constraint solving. In this article, we provide an overview of the state of the art in applying ML approaches to constraint solving problems including constraint satisfaction, SAT solving, answer set programming (ASP) and applications thereof such as configuration, constraint-based recommendation, and model-based diagnosis. We compare and discuss the advantages and disadvantages of these approaches and point out relevant directions for future work.

Keywords Constraint satisfaction · Boolean satisfiability · Constraint solving · Answer set programming · Machine learning · Applications

1 Introduction

Over several decades, *constraint solving* (we use this as a general term referring to specific problem solving approaches such as constraint satisfaction (Freuder, 1997), SAT solving (Gu et al., 1996), and answer set programming (Brewka et al., 2011)) has shown to be a core technology of Artificial Intelligence (Apt, 2003; Brailsford et al., 1999; Rossi et al., 2006). Its widespread use led to the development of algorithms and tools that help to tackle

✉ Andrei Popescu
andrei.popescu@ist.tugraz.at

a variety of combinatorial problems. Basic examples thereof are the *n*-queens problem and car sequencing (Adorf & Johnston, 1990; Tsang, 1993). The *n*-queens problem is related to the task of placing *n* queens on *n* distinct squares of an $n \times n$ chessboard in such a way that no pair of queens attacks each other. This problem has been extensively used to illustrate the modeling of constraint solving tasks and related search algorithms. The *car sequencing problem* is another problem that can be tackled on the basis of constraint solving techniques. This problem is related to the task of scheduling a car production sequence in such a way that capacity constraints of the assembly line are satisfied. Cars can be of different type and thus need different features to be installed and, as a consequence, also different production steps need to be taken into account. For example, a *type-A* car might need a *sunroof installed* but *no skibag*. An assembly line is often composed of different sections, each specialized in installing one or more specific features. Each section, however, has a limited capacity which has to be taken into account by the solver. Another application area of constraint-based knowledge representations and reasoning is knowledge-based configuration (Felfernig et al., 2014). Configuration has become a core technology for businesses depending on the mass customization paradigm (Felfernig et al., 2014). Configuration technologies help to tackle important challenges triggered by mass customization, for example, due to the correctness of configurations, errors in processing orders and related follow-up costs due to the production of faulty configurations can be avoided. Beyond configuration, constraint-based technologies are applied in a variety of further scenarios such as scheduling, vehicle routing, and recommender systems just to mention a few (Burke, 2000; Felfernig & Burke, 2008; Rossi et al., 2006).

As indicated by a couple of research results, combinatorial problems such as configuration problems have no universal best solving approach (Bengio et al., 2021; Kotthoff, 2014; Loreggia et al., 2016; O'Mahony et al., 2013; Spieker & Gotlieb, 2018; Xu et al., 2008; Xu et al., 2009). To solve such problems in an efficient fashion, state of the art techniques exploit *search heuristics* (Arbelaez et al., 2010; Beck et al., 2004; Da Col & Teppan, 2017; Erdeniz et al., 2017; Jannach, 2013; Johnston & Minton, 1994; Mouhoub & Jafari, 2011; Pearl, 1984; Sadeh & Fox, 1996) or try to find the best solver (algorithm) for the problem instance at hand. *Algorithm selection* approaches have shown to be applicable in various scenarios – related approaches to predict a solver's performance become an increasingly active field of research (Kotthoff, 2014). Furthermore, the development of machine learning methods that are capable of *learning search heuristics* in a domain-dependent fashion is relevant in the context of minimizing search efforts and optimizing the prediction quality (Raedt et al., 2011). Machine learning can also be used to predict the satisfiability of a specific problem and thus can help to significantly reduce the overall search effort. We have identified *the following main application areas* of machine learning techniques in the context of constraint solving: (1) *direct solution search*, (2) *constraint learning*, (3) *learning search heuristics (and/or parameters)* (e.g., related to possible variable and value orderings), (4) *satisfiability prediction* (by learning relevant problem features that help to correctly classify problems as (un-)satisfiable), and (5) *algorithm selection* (based on problem properties/features).

There exist different *surveys* related to the topics of *constraint satisfaction and SAT solving* (Apt, 2003; Rossi et al., 2006; Tsang, 1993), *algorithm portfolios* (Kotthoff, 2014), and also *interactions between data mining and constraint programming* (Bessiere et al., 2016). A survey of different algorithmic approaches to solve constraint satisfaction problems is provided, for example, by Kumar (1992). An overview of different approaches to SAT solving is provided, for example, by Gu et al. (1996). Compared to the literature study

presented in this article, commonalities exist with the overviews provided by Kotthoff (2014), Bischl et al. (2015), and Bessiere et al. (2016). Kotthoff (Kotthoff, 2014) provides a survey of algorithm portfolios that allow to select the “best” algorithm for a specific problem instance. Bischl et al. (2015) propose an algorithm benchmark including different datasets for evaluation purposes. Finally, Bessiere et al. (2016) focus on synergies between data mining and constraint satisfaction. They focus on related aspects such as approaches to algorithm selection, constraint learning, and representing data mining problems as constraint satisfaction problems. In this line of research, Alves Pereira et al. (2019) motivate the application of machine learning methods for the purpose of configuration space learning. The authors propose the application of machine learning methods for the purpose of predicting non-functional system properties such as response time or system failure probabilities. Furthermore, overviews of the state of the art in algorithm runtime prediction and selection can be found in Hutter et al. (2014) and Lombardi and Milano (2018). Finally, an overview of recent research related to the application of machine learning methods in the context of combinatorial optimization can be found in Bengio et al. (2021).

The major contribution of this article is the following. There does not exist an in-depth overview of machine learning methods aiming at improving constraint solving (in areas such as constraint satisfaction, SAT solving, and answer set programming). Although the interest in the field of applying machine learning to the solution of combinatorial problems is growing (Bengio et al., 2021), an overview specifically focusing on the application of machine learning approaches to constraint solving is still missing. With this article, we provide such an overview and also include an in-depth discussion of challenges and open issues for future work.

The remainder of this article is organized as follows. In Section 2, we present our methodological approach to the literature analysis on machine learning and constraint solving. Section 3 provides an introduction to major relevant concepts from the areas of *constraint solving* and *machine learning* that are regarded as a basis for being able to understand the discussions in Section 4. In Section 4, we summarize the results of our literature analysis on the integration of machine learning techniques into different types of constraint solving and provide selected examples to increase understandability. Specifically, we focus on related work in constraint satisfaction, SAT solving, answer set programming, and related applications in configuration, diagnosis, and constraint-based recommendation. In Section 5, we provide an overview of relevant issues for future research. The article is concluded in Section 6.

2 Research methodology

Our goal is to provide a comprehensive overview of the state of the art in the application of machine learning methods in constraint solving. We want to provide an in-depth overview of the field and thus also contribute to further bridge building between the two research fields of machine learning and constraint solving. This article is based on a literature review comprised of paper selection, review, and a discussion of reviewed approaches. Specifically, our work involved an initial step of *querying* leading *research portals* with relevant keywords, an intermediary phase consisting of *classifying approaches* based on their *contribution* to *Constraint Solving* and used *machine learning methods*, and a final phase with a *discussion of current challenges* and relevant *future research directions*.

We performed queries on well-known research platforms such as Google Scholar¹, ResearchGate², ScienceDirect³, SpringerLink⁴, and Elsevier⁵ with the initial keywords “machine learning” + “constraint satisfaction”. Taking into account the initial query results regarded as relevant, we established further search criteria (in terms of keywords) that combined in a systematic fashion “machine learning” and “constraint satisfaction” with related terms such as “deep learning”, “data mining”, “matrix factorization” and “constraint solving”, “SAT solving”, “Answer Set Programming”, and related meta-criteria such as “overview”, “review”, and “survey”. In addition, we analyzed topic-related work from the last 15 years published in the *International Joint Conference on Artificial Intelligence (IJCAI)*, the *AAAI Conference on Artificial Intelligence*, the *European Conference on Artificial Intelligence (ECAI)*, the *Conference on Theory and Applications of Satisfiability Testing (SAT)*, and the *Constraint Programming (CP)* conference. Following the snowballing technique (Wohlin, 2014), we further analyzed the reference sections of the identified papers with the goal to find further topic-related work. The major criterion for regarding a reference as relevant was a topic-wise match, i.e., the reference needed to cover the topic of integrating machine learning (ML) techniques with constraint solving. We have identified 41 publications as a basis for our overview. We did not include contributions focusing on the integration of constraint solving into machine learning (see, e.g., Lallouet & Legtchenko, 2005). The majority of the identified publications is based on supervised ML (see Table 1).

3 Preliminaries

In the following, we present relevant *constraint solving paradigms*, such as constraint satisfaction (Freuder, 1997), SAT solving (Gu et al., 1996) and answer set programming (Brewka et al., 2011), and key concepts from machine learning which constitute the basic knowledge required to be able to understand the discussions in this overview.

3.1 Constraint satisfaction problems (CSPs)

A constraint on a set of variables defines a restriction on possible combinations of variable assignments (Apt, 2003). Constraint Satisfaction Problems (CSPs) specify a problem in terms of relevant variables and corresponding constraints on possible value combinations (Apt, 2003). Ideally, an assignment should instantiate each variable and satisfy all the constraints defining the problem. Examples of CSPs are the configuration of telecommunication switches (Fleischanderl et al., 1998), the recommendation of financial services (Felfernig et al., 2006), and reconfiguration (Felfernig et al., 2018). As a basis for the following discussions, we now introduce a definition of a constraint satisfaction problem (CSP) (Rossi et al., 2006; Tsang, 1993).

Definition (CSP and Solution). A CSP is defined as a triple (V, D, C) where $V = \{v_1..v_n\}$ is a set of variables, $D = \{domain(v_1)..domain(v_n)\}$ are the corresponding

¹<https://scholar.google.com>

²<https://www.researchgate.net>

³<https://www.sciencedirect.com>

⁴<https://link.springer.com>

⁵<https://www.elsevier.com>

Table 1 Overview of example integrations of machine learning with constraint solving categorized based on problem type (CSP, SAT, and ASP) problems and related applications such as configuration, diagnosis, and constraint-based recommendation

Problem type	Paper	ML Method	Application	Evaluation Metrics
CSP	(Gent et al., 2010)	Decision Trees	Constraint Learning	Runtime
	(Adorf & Johnston, 1990)	Neural Networks	Solution Search	Completeness
	(Galassi et al., 2018)	Deep Learning	Solution Search	Completeness
	(Xu et al., 2018)	Deep Learning	Satisfiability Prediction	Accuracy
	(Gent et al., 2010)	Classification	Algorithm Selection	Runtime
	(Wang & Tsang, 1991)	Neural Networks	Solution Search	Runtime
	(Xu et al., 2009)	Reinforcement Learning	Learning Heuristics	Runtime
	(Erdeniz & Felfernig, 2018)	Clustering, Genetic Algorithms	Learning Heuristics	Runtime
	(Arbelaez et al., 2010)	Classification	Learning Heuristics	Runtime
	(Guerri & Milano, 2004)	Decision Trees	Algorithm Selection	Runtime
	(Bonfietti et al., 2015)	Decision Trees and Random Forests	Constraint Learning	Runtime
	(Bello et al., 2017)	Reinforcement Learning	Solution Search	Optimality
	(Cappart et al., 2020)	Reinforcement Learning	Learning Heuristics	Optimality
	(Nareyek, 2004)	Reinforcement Learning	Learning Heuristics	Optimality
	(Loreggia et al., 2016)	Deep Learning	Algorithm Selection	Accuracy
SAT	(O'Mahony et al., 2013)	Case-based Reasoning	Algorithm Selection	Completeness
	(Xu et al., 2008)	Ridge Regression	Algorithm Selection	Runtime
	(Bünz & Lamm, 2017)	Graph Neural Networks	Satisfiability Prediction	Accuracy
	(Selsam et al., 2019)	Neural Networks	Satisfiability Prediction	Accuracy
	(Hutter et al., 2006)	Linear Regression	Parameter Tuning	Runtime
	(Samulowitz & Memisevic, 2007)	Logistic Regression	Learning Heuristics	Runtime

Table 1 (continued)

Problem type	Paper	ML Method	Application	Evaluation Metrics
ASP	(Lagoudakis & Littman, 2001)	Reinforcement Learning	Learning Heuristics	Runtime
	(Yolcu & Pó, 2019)	Reinforcement Learning	Learning Heuristics	Completeness
	(Ansótegui et al., 2009)	Genetic Algorithms	Hyper Parameter Tuning	Runtime
	(Cameron et al., 2020)	Deep Learning	Satisfiability Prediction	Accuracy
	(Kurin et al., 2019)	Reinforcement Learning	Learning Heuristics	Runtime
	(Xu et al., 2012)	Decision Trees	Satisfiability Prediction	Accuracy
	(Haim & Walsh, 2009)	Logistic Regression	Algorithm Selection	Runtime
	(Liang et al., 2016)	Reinforcement Learning	Learning Heuristics	Runtime
	(Gebser et al., 2011)	Support Vector Regression	Hyper Parameter Tuning	Runtime
	(Maratea et al., 2012)	Nearest-Neighbour Classification	Algorithm Selection	Runtime
Configuration	(Yang et al., 2020)	Neural Networks	Learning Heuristics	Accuracy
	(Temple et al., 2017; Temple et al., 2016)	Decision Trees	Problem Model Learning	Accuracy
	(Acher et al., 2018)	Decision Trees	Constraint Learning	Accuracy
	(Da Col & Teppan, 2017)	Genetic Algorithms	Learning Heuristics	Runtime
	(Uta & Felfernig, 2020)	Neural Networks	Solution Search	Accuracy
Diagnosis	(Erdeniz et al., 2018)	Genetic Algorithms	Learning Heuristics	Runtime
	(Erdeniz et al., 2019)	Matrix Factorization	Learning Heuristics	Accuracy
Constraint-based Recommendation	(Erdeniz et al., 2019)	Matrix Factorization	Learning Heuristics	Accuracy
	(Zanker, 2008)	Collaborative Filtering	Constraint Learning	Accuracy

domain definitions, and $C = \{c_1..c_k\}$ is a set of constraints. A solution to a CSP is a variable assignment $A = \{v_1 = val_{v_1}..v_n = val_{v_n}\}$ where val_{v_j} denotes the value of variable v_j with $val_{v_j} \in domain(v_j)$ and $consistent(C \cup A)$.

A basic approach to solve a CSP is to apply *backtracking search* combined with *forward checking* (analyzing whether a variable value assignment has an impact on the set of possible values of other variables) and different approaches assuring consistency properties such as *node consistency* (each value of a variable domain has to be consistent with all unary constraints defined on that variable) and *arc consistency* (for each binary constraint c_a connecting two variables, e.g., x_1 and x_2 , it must be the case that each value of $domain(x_1)$ must have a corresponding values in $domain(x_2)$ such that the corresponding assignment is consistent with c_a). For a general overview of different approaches to solve constraint satisfaction problems we refer to Apt (2003). Example evaluation criteria that can be used for evaluating the performance of solvers are *runtime* (how long does it take to find a solution), *optimality* (solution that achieves an optimum with regard to a predefined optimization function), *accuracy* (share of correct classifications), and *completeness* (degree to which existing solutions can be found).

3.2 Boolean satisfiability problems (SAT & ASP problems)

A Boolean Satisfiability (SAT) problem is a specific type of CSP where variables have Boolean domains and constraints are represented as Boolean formulas (Rossi et al., 2006). As a consequence, CSPs and SAT problems are in a close relationship which allows for a reformulation of CSPs into corresponding SAT representations (Gu et al., 1996). In the line of the definition of a CSP, a SAT problem is defined by a set of variables $V = \{v_1..v_n\}$ where each variable represents a specific domain value. For example, if $V = \{x_1\}$ and $domain(x_1) = \{a, b\}$ in the CSP representation, the corresponding SAT representation would be $V = \{x_a, x_b\}$ with $domain(x_a) = domain(x_b) = \{true, false\}$. Constraints in SAT solving are represented in terms of clauses where each clause is represented by a disjunction of literals (a positive literal represents the assumption that the variable value is “true”, a negative literal represents the assumption that the variable value is “false”) (Gu et al., 1996). A solution to a SAT problem is a consistent variable assignment for the equivalent conjunctive normal form (CNF) which represents a conjunction of the mentioned clauses. For an overview of different approaches to solve SAT problems, we refer to Gu et al. (1996). Importantly, answer set programming (ASP) (Brewka et al., 2011) is an expressive declarative knowledge representation that allows to model complex domains in a predicate logic fashion. On the reasoning level, answer set programs are translated into a SAT representation, i.e., ASP problems can be solved on the basis of SAT algorithms. It is worth mentioning that one of the first real-world applications of answer set programming was *product configuration* – for details we refer to Myllärniemi et al. (2014) and Simons et al. (2002).

3.3 Machine learning

The basic idea of machine learning is to exploit data from past events (represented in terms of training data) to build a (hopefully) reliable model that is able to generalize the given data. On a high level, machine learning is used to solve two basic tasks. First, *classification* is used to figure out for a given test case if it falls under a given category or not. For example, in the context of a constraint-based configuration scenario, an estimation whether a user will be interested in a specific potentially new product component (or a new service) can be

regarded as a classification task, i.e., to figure out whether an item is relevant or irrelevant. Second, *prediction* is used to estimate a specific value. For example, the estimation of the upper price limit of a user can be regarded as a prediction task (predicting user-individual price limits). Depending on the machine learning task to be solved (classification or prediction), corresponding evaluation metrics can be applied, i.e., classification metrics such as *precision* and *prediction metrics* such as *mean squared error* (MSE) (Gunawardana & Shani, 2015; Uta et al., 2021). In addition to the basic differentiation between classification and prediction, machine learning algorithms can be further differentiated as follows. First, *supervised learning* is based on the idea of learning from labeled data (often denoted as “datasets”) where datasets are divided into a training dataset (used for learning purposes) and a test dataset (used for evaluation purposes). Examples of machine learning algorithms implementing supervised learning are *linear regression*, *decision tree based classification*, *matrix factorization*, and *neural networks* (Bishop, 2006). Second, *unsupervised learning* focuses on the identification of patterns in unlabeled data. A basic example of an algorithmic approach supporting unsupervised learning are different *clustering algorithms*. Third, *reinforcement learning* learns from its past experience by trying to maximise a reward function with respect to the next state and actions available for selection (Sutton & Barto, 1998).

4 Machine learning in constraint solving

The majority of research contributions related to the integration of machine learning with constraint solving are based on *supervised machine learning*. In the following, we will discuss these approaches in detail and – where applied – also discuss the corresponding unsupervised learning and reinforcement learning approaches. Table 1 provides an overview of relevant research contributions related to the topic of machine learning and constraint solving. In the line of the basic classification of machine learning methods as discussed in Carbonell et al. (1983), we categorize the identified research contributions according to the used machine learning method/algorithm (e.g., decision trees and neural networks), the application in the constraint solving context (e.g., algorithm selection and learning of heuristics), and the used evaluation metrics (e.g., runtime and accuracy).

4.1 Constraint satisfaction

Research contributions related to the integration of machine learning and constraint satisfaction problem solving (Freuder, 1997; Tsang, 1993) support different goals. In the context of solution search, *constraint learning* supports the learning of constraints that will help to avoid redundant search and thus improve the search performance. *Solution prediction* is related to the goal of directly applying machine learning to derive a solution for a given constraint satisfaction problem (CSP). Furthermore, *satisfiability prediction* focuses on predicting the satisfiability of a CSP without the need of activating a solver. *Heuristics learning* focuses on the learning of search heuristics that improve the performance of a constraint solver with regard to criteria such as runtime and prediction quality. Finally, *constraint learning* in the context of knowledge acquisition supports the derivation of constraints from a given dataset or on the basis of the feedback from expert users. Research contributions related to these aspects will be discussed in the following paragraphs.

The idea of *constraint learning* is to identify previously unknown constraints of a CSP and – through the inclusion of these constraints – speed up the search process. Gent et al. (2010) introduce a *classification model* that helps to identify CSP instances on

which constraint learning is likely to improve a solver's performance. The motivation behind this work is that not every CSP instance is amenable of constraint learning which means, it cannot be guaranteed that constraint learning helps to improve performance in every case. In order to better understand the utility of constraint learning, Gent et al. (2010) introduce a decision tree based approach that helps to evaluate a given CSP instance. Specifically, the authors use *lazy learning* in the context of *nogood learning* where a nogood can be interpreted as a combination of attribute values that do not allow the determination of a solution. Such nogoods can be interpreted as additional constraints to be taken into account by the solver. Examples of features used in the decision tree based approach of Gent et al. (2010) are *number of problem constraints*, *number of variables*, *number of attributes*, *number of constraints that refer to a specific variable*, and the *proportion of tuples not allowed by a specific constraint* (constraint tightness).

A *neural network architecture* that helps to solve constraint satisfaction problems is proposed in Wang and Tsang (1991). CSP variables and potential values are encoded as nodes of the neural network. Based on the results of their performance evaluation, the authors point out that CSP solution search based on neural network approaches is feasible also for CSPs with up to 170 variables. As a working example, the authors present the domain of *car scheduling*. A neural network can learn relevant variable assignments on the basis of a training data set which is represented by a collection of already solved CSPs. In order to escape local minima (e.g., in terms of the number of still violated constraints), the authors propose a so-called heuristic repair method. More in-depth evaluations are needed to have a clearer view in which contexts neural network based approaches can outperform standard constraint solving. A multi neural network architecture for solving binary CSPs is proposed by Adorf and Johnston (1990) where the problem of escaping local minima is solved by the integration of so-called auxiliary guard networks.

A *deep learning based approach* to predict the satisfiability of constraint satisfaction problems is presented by Xu et al. (2018). Their approach exploits basic properties of CSPs (e.g., individual variable values) as input for the neural network which has to decide whether the given CSP is satisfiable or not. Besides satisfiability, further labels of data points are *k-consistency*, *best algorithm to solve*, and *required time to solve the problem*. The input for the neural network is the *CSP represented in matrix form* where each entry indicates whether the corresponding value combination is allowed or not. The rationale behind this representation is its similarity to a 2-D representation of a grayscale image which is also used as a basis for pattern recognition in image analysis. The major motivation behind the work of Xu et al. (2018) is to accurately predict the satisfiability of CSPs to achieve the goal of minimizing the need of backtracking in solution search. In the context of the presented evaluation settings (synthesized dataset that includes data points labeled in terms of satisfiability or unsatisfiability), the authors report to have achieved an accuracy rate of higher than 99.99%. A similar line of research is followed by Wen et al. (2020) who apply deep learning for improving the performance of constraint solving in the context of symbolic execution. Symbolic execution is used, for example, in *software testing* and *test case generation* with the goal to identify in an efficient fashion executable paths in a program.

The objective of *continuous search* is to permanently improve the search behavior of a solver (Arbelaez et al., 2010). Related approaches include *two basic modes*: first, the so-called *functioning mode* which is used to solve concrete user problems (ako productive environment) whereas the *exploration mode* focuses on continuously improving the search

performance by reusing the information from already completed solver sessions for *training the heuristics model*. The underlying machine learning approach is based on a binary classification model. As mentioned by Arbelaez et al. (2010), the proposed approach follows the idea of lifelong learning which means to successively become an expert in the application domain. Based on a set of *static features* (e.g., *#variables*, *variable domain sizes*, and *constraint degrees*) and *dynamic features* (e.g., *used heuristics for variable selection*, *variable domain size during search*, and *involvement degree of variables in failed constraints*), the underlying machine learning task is defined as a binary classification problem. A related neural network approach to the training of heuristics is introduced in Galassi et al. (2018).

Permanently improving the search performance is also the idea of *reinforcement learning* which follows the idea of continuously exploring new problem solving approaches with the overall goal of continuously improving problem solving behavior and corresponding outputs (Bello et al., 2017; Lagoudakis & Littman, 2001; Lederman et al., 2020). Xu et al. (Xu et al., 2009) and Spieker and Gotlieb (2018) introduce reinforcement learning approaches for efficiently solving constraint satisfaction problems. Adaptive behavior is achieved, for example by the continuous adaptation of individual variable ordering heuristics with the goal to improve the performance of the problem solving behavior. Reinforcement learning does not primarily rely on datasets used for evaluation purposes but focuses more on an *instance by instance training* with adaptations based on the feedback of *reward functions*. A simple example of such a reward function is the *validity of a solution* or the *share of violated constraints* in a proposed solution. Consequently, *two major roles* can be observed in reinforcement learning based approaches: first, *actors* are responsible for the *selection of the next actions* (e.g., adaptation of the domain ordering heuristic for a specific variable), second, *critiquing agents* are responsible for *estimating the reward after an action has been performed* (Nareyek, 2004; Spieker & Gotlieb, 2018).

Combinatorial optimization is related to the task of *finding an optimal solution* for a given (constrained) problem (Cappart et al., 2020; Korte & Vygen, 2000). Example algorithmic approaches to solve such problems range from solving integer linear programs to basic formulations as constraint satisfaction problems. Guerri and Milano (2004) apply decision tree learning in the context of the *Bid Evaluation Problem* (BEP) which is regarded as a combinatorial optimization problem. They propose the application of a decision tree to decide about the solving approach to be used for a problem instance. There are two basic problem solving approaches taken into account in the reported evaluation scenario which are *constraint programming* (CP) and *integer programming* (IP). Each entry in the machine learning training set consists of 25 features and – as an annotation – the best algorithm to solve the problem.

In the context of combinatorial optimization, Bonfietti et al. (2015) propose an approach to the integration of *Decision Trees* (DTs) and *Random Forests* (RFs) in a constraint programming model. The major motivation for this integration is that in many domains constraints and heuristics part of the model are not completely clear. The overall idea is to learn additional constraints that are then integrated into the constraint model. This approach can be regarded as a kind of empirical model learning where data from past solutions can be used to infer the mentioned classification models. In the work of Bonfietti et al. (2015), different embedding techniques are used which basically represent different ways of integrating the knowledge from decision trees and random forests into the constraint-based representation. An example thereof is a *rule-based embedding* where rules become part of the constraint-based representation.

Runtime prediction and optimization is important in different contexts. For example, predicting the performance of a solver given a set of solver settings and a corresponding problem instance can help to avoid suboptimal solver parametrizations but also support the optimization of algorithm parametrization, i.e., parameter tuning. Furthermore, runtime predictions in the context of solution space learning can help to efficiently estimate the applicability of a specific (constraint-based) knowledge base in interactive settings. With the goal to support the prediction of solver runtimes, Hutter et al. (2006) propose a regression-based prediction model where empirical hardness aspects derived from a problem instance are used as basic indicators for runtime estimation.

Gent et al. (2010) introduce a classification based approach to select the optimal implementation of the *alldifferent* constraint with the overall goal of solver performance optimization. The semantics of the *alldifferent* constraint associated with a set of variables $V = \{x_1, x_2, \dots, x_n\}$ can be defined as $\forall_{(x_i, x_j)} : val(x_i) \neq val(x_j)$ with $i \neq j$. The features used by the classification approach are *constraint and variable statistics* directly derived from each given CSP instance. The training data was automatically labeled on the basis of the performance of a specific *alldifferent* implementation given a set of problem-specific constraint and variable statistics (features). Examples of such features are *average constraint arity*, *proportion of constraint pairs that share more than one variable*, and *constraint tightness in terms of the portion of disallowed tuples*. The approach showed to clearly outperform CSP implementations with one specific *alldifferent* variant.

In contrast to the previously discussed work, Erdeniz and Felfernig (2018) show how to make the learning of CSP heuristics more applicable to interactive settings. In their approach, a dataset of already collected user requirements is forwarded to a *k*-means clustering process (Burkardt, 2009) which is used to group similar sets of user requirements. Thereafter, in the line of the work of Da Col and Teppan (2017), a genetic algorithm is used to determine cluster-specific heuristics that cause the best runtime behavior for the problem instances (user requirements) in the individual cluster. After having determined the cluster-specific heuristics, the resulting models (clusters) can be applied in interactive settings in the context of new user sessions: if a user specifies a set of preferences, the system can determine the most similar cluster and use the cluster-specific heuristics for determining a solution. Based on example map coloring problems (Fritsch et al., 1998; Leighton, 1979), Erdeniz and Felfernig (2018) show that their cluster-based approach outperforms settings with globally defined variable and value ordering heuristics. For visualization purposes, a simple example of learning of variable value orderings in the context of constraint-based recommendation scenarios is depicted in Tables 6, 7, 8 and 9.

An application of case-based reasoning (CBR) for the *selection of well-performing constraint solvers* is introduced by O'Mahony et al. (2013). CBR is based on the idea of exploiting existing problem solving knowledge from previous cases for solving new problems. In this context, old cases are retrieved that are similar to the new problem and the problem solving practices of the old cases are reused to solve the new one. In the approach proposed by O'Mahony et al. (2013), solvers are recommended which have a high probability of efficiently solving the given problem. The similarity between old cases and the new ones is determined on the basis of similarity metrics defined on features such as *maximum constraint arity*, *maximum variable domain size*, *number of variables*, *number of constraints*, and *ratio of different types of constraints* (e.g., extensional and intensional). Table 2 depicts a simple *solver recommendation scenario* where individual *sessions* reflect stored data about already completed constraint solving sessions and the task is to find an appropriate solver for the current session ($s_{current}$).

Table 2 A sketch of a case-based reasoning scenario focusing on the identification of appropriate solver parameter settings (p_i represent problem features such as #constraints, average variable domain sizes, and used variable ordering heuristics), *solver* denotes the selected solver, *runtime* denotes the time (in *msec*) needed by the solver to determine a solution

session	p_1	p_2	p_3	p_4	p_5	solver	runtime
s_1	1	2	1	3	2	<i>solver₅</i>	191.2
s_2	3	10	1	5	1	solver₁	1012.7
s_3	3	12	1	7	1	solver₂	78.1
s_4	1	2	1	3	2	<i>solver₅</i>	33.82
s_{current}	3	10	1	5	1	?	?

The nearest neighbor in terms of the problem features p_i of the *current* session is session s_2 , the second nearest neighbor is session s_3 . Since the solver chosen for session s_3 showed a significantly better performance, solver *solver₂* (used in session s_3) should be recommended/used for/in the current session.

Constraint learning Raedt et al. (2018) in the context of knowledge acquisition scenarios (i.e., not during solution search) can be interpreted as inductive learning task of identifying a constraint set that supports a training/test dataset. This problem is different from learning processes within the scope of a solver session where the learned constraints are primarily used to improve solver performance (Gent et al., 2010). Learning constraints from data is a relevant approach to tackle issues in the context of the *knowledge acquisition bottleneck* which can be a major obstacle when deploying constraint-based systems in industrial contexts. Another example task of constraint learning is the *learning of preferences* which is used, for example, when learning search heuristics for improving the prediction quality of a solver (Erdeniz et al., 2019). Constraints can also be learned in an interactive fashion within the scope of so-called *active learning* processes where the machine learning component is not limited to the analysis of a given dataset but also interacts with expert users who give feedback on the relevance of learned constraints. In this line of research, also human computation based approaches to constraint learning (Ulz et al., 2016) are used to exploit the feedback of expert users.

4.2 SAT solving

In the SAT solving literature, there are two basic method categories how to integrate machine learning for the purpose of problem solving (Zhang et al., 2020). First, machine learning can be used directly to solve individual SAT instances, i.e., given a SAT problem as input, the machine learning algorithm learns to solve the problem instance itself – see, for example, Galassi et al. (2018). Second, so-called heuristic methods use machine learning for the purpose of predicting relevant heuristics which are then used by solvers to find a solution for a given instance (Zhang et al., 2020). Beyond solution search, there are further applications of machine learning techniques in SAT solving which include the aspects of *predicting satisfiability* and *selecting the best algorithm* for solving a given problem instance. Related scientific contributions will be discussed in the following paragraphs.

The task of algorithm selection is to select the best algorithm to solve a specific problem instance. An in-depth survey of algorithm selection techniques is given in Kotthoff (2014). One of the most prominent environments supporting algorithm selection is SATZILLA

(Xu et al., 2008). It can be regarded as one of the first systems propagating the idea of *algorithm portfolios* as a reasonable alternative to the optimization of individual algorithms. The application context of SATZILLA is SAT solving scenarios where *runtime per problem instance* is used as evaluation criteria (see also Haim & Walsh, 2009 where machine learning is applied to determine solver restart strategies). Algorithm portfolios can be *static* (a portfolio is comprised of a fixed set of algorithms which comes along with limitations regarding flexibility) or *dynamic* (algorithms in a portfolio can be modified, e.g., on the basis of parametrization). A dynamic approach proposed in the context of constraint solving is the ADAPTIVE CONSTRAINT ENGINE (ACE) (Epstein & Freuder, 2001) which supports the adaptive selection of variable orderings based on a multi-tier adaptive voting architecture with the goal to optimize solution search. Related approaches to heuristic selection in SAT solving are proposed in Ansótegui et al. (2009) and Samulowitz and Memisevic (2007). Finally, Da Col and Teppan (Da Col & Teppan, 2017) show how to apply genetic algorithms to automatically figure out optimal search heuristics (in terms of variable orderings, value orderings, and pruning strategies) for a set of predefined benchmark problems.

An approach to the application of neural networks for *classifying SAT problems in terms of satisfiability* is introduced by Selsam et al. (2019). In this context, a SAT problem is encoded as a undirected graph where each node represents one literal of the SAT problem. Furthermore, each clause is represented as a node of the neural work which is connected to those literals that are part of the clause. In this scenario, the neural network can be applied for two purposes: first, it can be applied for *predicting the satisfiability* of a given SAT problem. Second, it can be applied for *determining a solution*. If one exists, it can be directly derived from the individual activation level of the nodes in the neural network. The proposed approach is also flexible in terms of solving SAT problems in different domains such as *graph coloring* (e.g., in map coloring, no two adjacent map areas have the same color) and *clique detection* (e.g., in social network analysis, the goal could be to identify subgraphs of people where everybody knows each other). Another classification approach for estimating the satisfiability of a given SAT problem is discussed in Bünz and Lamm (2017). Regarding the used structural properties of SAT formulae, the authors point out that neural networks fail to produce a different output given certain highly particular symmetries between two different inputs (SAT problems).

In many application contexts, there is no single solver that provides the optimal performance for a range of different problem settings (Loreggia et al., 2016). The goal of *algorithm selection* is to identify the best problem solving approach (algorithm) for a given problem at hand. Such an algorithm selection is often based on a manually defined set of features which are then used by different machine learning techniques to predict the optimal solver for a given problem. The deep learning approach presented in Loreggia et al. (2016) supports the automated determination of an informative set of machine learning features thus making the algorithm selection a completely automated process. Interestingly, the deep learning approach is based on the idea of transforming input text files representing problem instances into corresponding grayscale square images that serve as an input for the training of the neural network. The presented approach outperforms single solvers. A basic example of a related neural network architecture supporting the selection of solvers on the basis of a given set of problem properties (features) is depicted in Fig. 2.

The term *phase transition* denotes the transition from a region of search spaces where the large majority of problems have many solutions (the problems are under-constrained) to a region where the large majority of problems has no solution (the problems become

increasingly over-constrained). Xu et al. (2012) introduce a decision tree based classification approach that supports the classification of SAT problems with regard to satisfiability. The classification approach is based on a set of features derived from a given SAT problem. An example of such a feature is the average *imbalance of the non-negated and negated occurrences of SAT variables*. Following a similar idea, Yolcu and Pó (2019), Kurin et al. (2019), and Liang et al. (2016) introduce reinforcement learning approaches with the goal of improving the performance of SAT solving through the repeated adaptation of search heuristics.

With the focus of *predicting the satisfiability of individual 3-SAT problems* (maximum three literals per clause), Cameron et al. (2020) introduce a deep neural network based approach based on the concept of end-to-end learning (all model parameters are learned at the same time). In this context, the authors introduce two different network architectures, one focusing on the prediction of satisfiability, the other one supporting the prediction of satisfiable variable assignments. These variants were evaluated on SAT instances ranging from 100 to 600 variables and showed an increased prediction accuracy compared to related work (Xu et al., 2012). For visualization purposes, we provide a simple example of a neural network architecture (see Fig. 1).

4.3 Answer set programming (ASP)

An approach to *solver selection in the context of ASP scenarios* is introduced by Gebser et al. (2011). More precisely, the authors show how to select a specific solver configuration given a specific answer set program (ASP). The problem features used by the machine learning approach (support vector regression) are, for example, *number of constraints*, *number of variables*, *average backjump length*, and *length of learned clauses*. These features are used to map a given problem to the most promising solver configuration.

Yang et al. (2020) present a framework (NeurASP) for the integration of neural networks into answer set programs. The underlying idea is to exploit the output of a neural network for the purpose of ranking atomic facts in answer set programs. As mentioned by the authors, this integration is a good example of approaches combining symbolic AI (ASP) and sub-symbolic AI (neural networks). It helps to exploit synergy effects by combining the strengths of both worlds. For example, in the context of solving Sudoku, neural networks can be used to recognize the digits on the game board whereas ASP logic can be used to represent the constraints of the game and to find solutions. The approach of Yang et al. (2020) easily generalizes to different *Sudoku* instances.

Fig. 1 A simple neural network architecture with one hidden layer that can classify a SAT problem as satisfiable or not. Examples of properties $prop_i$ (features) are problem size, variables per clause, and balance between negated and non-negated occurrences of SAT variables

As mentioned, there is rarely one specific problem solving approach/algorithm that performs best in all of a given set of problem instances. More often, the best-performing algorithm differs depending on the problem setting. In the line of the research of Gebser et al. (2011) and Maratea et al. (2012) present a classification-based (a.o., nearest neighbor based classification) machine learning approach to *per problem instance solver selection* in a multi ASP solver environment. Examples of features used by the classification approach are the following ones: *number of rules*, *number of atoms*, *share of unary/binary/ternary rules*, and *number of true facts*. For visualization purposes, we provide a simple example of a related neural network architecture (see Fig. 2).

4.4 Configuration

Highly-variant products, services, and systems allow for a potentially huge number of possible configurations (Felfernig et al., 2014; Felfernig et al., 2021). The corresponding configuration space could also allow faulty configurations simply due to the fact that some relevant constraints have not been integrated into the configuration model. Examples of such faulty configurations are non-compilable software components and non-producible products. Configuration problems are typically represented on the basis of the already discussed approaches of constraint satisfaction, SAT solving, and answer set programming (ASP). In the context of *feature model development*, Temple et al. (2016) introduce a classification-based (decision trees) learning approach that supports the derivation of additional constraints that help to reduce the space of possible configurations. In their approach, an oracle is used to estimate the correctness of generated configurations. These annotated configurations are then used to build a decision tree that classifies correct and faulty configurations depending on the configuration-individual attribute values. This decision tree is then used for the derivation of the mentioned additional constraints. Intuitively, new relevant constraints can be extracted by negating those decisions that led to faulty configurations (Temple et al., 2016). In a related work, Temple et al. (2017) propose a machine learning approach that helps to relate context information (represented in terms of feature selections) with corresponding parts of a software and thus replaces manual modeling efforts. Based

Fig. 2 A simple neural network architecture with one hidden layer that can be used for learning solver selection ($solver_i$) based on basic properties ($prop_j$) of a given ASP. Examples of such properties (features) are: number of rules in the ASP, number of atoms, share of unary/binary/ternary rules, and number of true facts

on the criteria of runtime performance of generated configurations, these problem instances (configurations) can be evaluated as acceptable or unacceptable. On the basis of this annotated dataset, decision trees can be learned which again serve as a basis for the derivation of additional constraints that reduce the search space.

In the context of *configuring highly-variant products* with around 40 different product properties, Uta and Felfernig (2020) introduce a machine learning approach that helps to predict variable values a user would regard as relevant. The learning approach is based on a three-layer neural network with one input, output, and hidden layer. In the presented approach, variable value predictions do not take into account domain constraints which can result in situations where the recommendations can become inconsistent with the underlying product model. Related approaches based on recommendation techniques are summarized a.o. in Falkner et al. (2011) where techniques such as collaborative filtering (Konstan et al., 1997) are applied to predict variable values of relevance for a user. A basic method to avoid inconsistencies between recommended variable values and the underlying product domain knowledge is to avoid direct variable value predictions but forward value selection probabilities to a configurator (e.g., a constraint solver) which can integrate this information, for example, in terms of variable value ordering heuristics.

Writing a paper, a project proposal, or a curriculum vitae that meets different criteria such obligatory contents, maximum and minimum sizes of paragraphs, upper and lower bounds in terms of pages and words used in a document is often a challenging task. Reasons for making such tasks challenging are time pressure when submitting the document or simply cognitive overheads that are triggered by complex rules regarding the correct format of a submitted document. In order to tackle these challenges, Acher et al. (2018) introduce VARYLATEX which enriches L^AT_EX document editing tasks with the concepts of variability modeling (in terms of feature models) and binary classification which classifies a configured document as acceptable or unacceptable. Features used by classification are all *variability properties* defined for the document under consideration.

For an overview of the application of machine learning and recommender systems techniques in configuration scenarios (especially in feature-modeling related scenarios), we refer to Felfernig et al. (2021). The example depicted in Table 3 sketches a basic approach for *value prediction in configuration scenarios*. For example, a user interacting with a configurator has already specified some parameters and the task is to recommend values for parameters that have not been specified up to now. Following the principles of collaborative filtering (Konstan et al., 1997), the *nearest neighbor* of the current user ($u_{current}$) is the

Table 3 A simple example of applying collaborative filtering (Konstan et al., 1997) in configuration contexts

user	$x_1(a, b)$	$x_2(u, v)$	$x_3(1, 2, 3)$	$x_4(1, 2)$	$x_5(3, 4, 5)$
u_1	a	u	2	1	3
u_2	b	v	3	2	5
u_3	b	v	1	2	4
$u_{current}$	a	u	2	2	? → 3

The nearest neighbor (in terms of an overlap in the individual preferences) of the current user $u_{current}$ is the user u_1 . In this scenario, we are interested in a prediction of the value of x_5 to be recommended to user $u_{current}$. Following the preferences of the nearest neighbor u_1 , the recommended value for $u_{current}$ for variable x_5 would be 3

user u_1 . His/her preference with regard to the variable x_5 is 3. Consequently, 3 is taken as recommendation for variable x_5 for user $u_{current}$.

4.5 Diagnosis

If no solution can be identified for a given constraint set $C = \{c_1, c_2, \dots, c_n\}$, one or more conflicts exist in C . In this context, a minimal conflict (set) $CS \subseteq C$ represents a minimal set of constraints which is inconsistent (Junker, 2004), i.e., $\text{inconsistent}(CS)$. If CS is minimal, only one constraint has to be deleted from CS to restore consistency. If C entails more than one conflict, each individual conflict has to be resolved to restore consistency in C . A minimal set of constraints that needs to be deleted from C to restore consistency is also denoted as *minimal hitting set* (for the given conflict sets) or *minimal diagnosis* (Felfernig et al., 2012; Reiter, 1987). A *diagnosis* (often denoted as Δ) is minimal if $\neg \exists \Delta' : \Delta' \subset \Delta$ and Δ' is still a diagnosis.

In most scenarios, there exists more than one diagnosis for a given inconsistent constraint set C (e.g., a CSP). In such cases, we need to find the most relevant diagnosis, for example, a diagnosis that will be with a high probability of relevance for a user. Erdeniz et al. (2018) introduce an approach based on *clustering and genetic algorithms* that help to optimize the prediction quality for diagnoses which is especially relevant in interactive settings. Assuming the existence of a dataset comprised of sets of inconsistent user requirements, clustering can be applied to form individual groups of similar (inconsistent) user requirements. On each of the individual clusters, a genetic algorithm can determine a cluster-specific importance ranking of constraints (user requirements) that helps to maximize the diagnosis prediction quality.

For diagnosis purposes, Erdeniz et al. (2018) apply a specific variant of FASTDIAG (Felfernig et al., 2012) which is a divide-and-conquer based direct diagnosis algorithm. FASTDIAG is based on a constraint ordering that has an impact on the determined diagnosis. For example, if we assume $C = [c_1, c_2, c_3]$ (ordered set notation) and there exists a conflict $CS = \{c_1, c_3\}$, the resulting diagnosis would be $\Delta = \{c_1\}$. If we assume $C = [c_3, c_2, c_1]$, the corresponding diagnosis determined by FASTDIAG would be $\Delta = \{c_3\}$. In the approach of Erdeniz et al. (2018), optimal cluster-specific constraint orderings are determined by a genetic algorithm. The optimization criteria in this context is prediction quality, for example, in terms of *precision* which is $\frac{|\text{correctly predicted diagnoses}|}{|\text{predictions}|}$. In a new user session where user requirements become inconsistent, the cluster most similar to the current set of user requirements has to be identified. Using the cluster-specific constraint ordering, FASTDIAG can be activated with the current set of inconsistent user requirements. More details about the used diagnostic approach can be found in Felfernig et al. (2012) and Felfernig et al. (2018). A further development of the approach of Erdeniz et al. (2018) is introduced in (Erdeniz et al., 2019) where the task of adaptive constraint ordering (in diagnosis contexts) is represented as a matrix factorization task (Koren et al., 2009).

A simple example of the impact of different constraint orderings on the outcome of a diagnosis task is depicted in Tables 4 and 5. In this example, we assume a knowledge base consisting of the constraints $C = \{c_1 : x_1 \neq x_2, c_2 : x_2 \neq x_3\}$ and the additional constraints $\{c_3, c_4, c_5\}$ representing, for example, user requirements. In Table 4, the constraint with the lowest importance is assumed to be c_3 whereas in Table 5, the constraint with the lowest importance is assumed to be c_5 . The different rankings result in different corresponding diagnosis (Δ_1, Δ_2). In Tables 4 and 5, $\times$ indicates that the corresponding constraint has been selected for conflict resolution, i.e., is part of a diagnosis Δ_i .

Table 4 Constraint ranking with c_3 having the lowest importance and c_5 the highest

conflict(CS)	$c_3 : x_1 = 1$	$c_4 : x_2 = 1$	$c_5 : x_3 = 1$
CS_1	$\underline{\times}$	$\times$	—
CS_2	—	$\underline{\times}$	$\times$

Resolving the assumed minimal conflicts from left to right results in diagnosis $\Delta_1 = \{c_3, c_4\}$

4.6 Constraint-based recommendation

Constraint-based recommendation (Felfernig & Burke, 2008; Gräsch et al., 2013; Zanker et al., 2007) is based on the idea of encoding recommendation knowledge into a set of constraints or rules which has to decide about the acceptability of individual items. Consequently, in their pure form, constraint-based recommender systems are quite similar to knowledge-based configuration systems. A major difference is the underlying solution space which is typically quite large in the context of configurator applications (Alves Pereira et al., 2019) but rather small in recommendation scenarios where the different items (products) are enumerated in terms of constraints describing a product catalog (Felfernig & Burke, 2008). Constraint-based recommender systems in their basic form do not provide a means of ranking individual recommendation candidates. In order to support the ranking of recommendation candidates, Felfernig et al. (2006) show how to integrate the concepts of utility-based recommendation with constraint-based reasoning by adding an additional evaluation/ranking phase that follows an initial candidate item selection process performed, for example, by a constraint solver. Following a similar goal, Zanker (2008) proposes an approach to derive preference knowledge from a collaborative filtering process (preferences are derived from a users nearest neighbors) and exploit the learned preferences in a semantic-level recommender system.

More advanced approaches are able to rank individual solutions where recommendation knowledge is *encoded in the search heuristics* of the used constraint solver. Erdeniz et al. (2019) introduce a novel approach to preference learning in the context of constraint-based recommendation scenarios. In this work, value ordering heuristics are learned on the basis of previous user interactions. The underlying idea is to apply the concepts of *matrix factorization* (Koren et al., 2009) to support the determination of individual attribute value recommendations for the current user. Matrix factorization (Koren et al., 2009) is based on the idea of approximating the original (potentially sparse) matrix R by two lower-rank matrices U and V where $U \times V = R'$ and R' (the resulting dense matrix) approximates R . The idea introduced by Erdeniz et al. (2019) is to learn individual user preferences on the basis of historical data and to apply the acquired knowledge for determining recommendations for a new user. After a new user has specified a couple of preferences, the nearest

Table 5 Constraint ranking with c_5 having the lowest importance and c_3 the highest

conflict(CS)	$c_5 : x_3 = 1$	$c_4 : x_2 = 1$	$c_3 : x_1 = 1$
CS_1	$\underline{\times}$	$\times$	—
CS_2	—	$\underline{\times}$	$\times$

Resolving the assumed minimal conflicts from left to right results in diagnosis $\Delta_2 = \{c_5, c_4\}$

Table 6 The original (potentially sparse) matrix R

user	$x_1.v_1$	$x_1.v_2$	$x_2.v_1$	$x_2.v_2$	$x_3.v_1$	$x_3.v_2$	$x_3.v_3$
u_1	0	1	1	0	1	0	0
u_2	1	0	0	1	0	1	0
u_3	0	1	1	0	0	1	0
u_4	1	0	1	0	1	0	0

In this context, $x_i.v_j = 1$ denotes the fact that a user selected the value v_j contained in the domain of variable x_i . Vice-versa, $x_i.v_j = 0$ denotes the fact that a user did not select the corresponding value

neighbors of the current user can be determined by comparing the preferences of the current user with the corresponding attribute value selections of the nearest neighbors.

In the following, we will explain the approach of learning search heuristics proposed by Erdeniz et al. (2019) on the basis of a simple example. Table 6 represents the history of already completed user sessions where $x_i.v_j = 1$ denotes the fact that a user selected the attribute value v_j of variable x_i (in machine learning, also denoted as *one hot encoding*).

We are now interested in performing a factorization between two low-rank matrices U and V that can approximate R . The approximation of R is $R' = U \times V$ (see Tables 7 and 8).

Finally, the approximation of R which is R' is depicted in Table 9. Given a new user ($u_{current}$ in Table 9) who has already specified a set of attribute values, a nearest neighbor (or a set of nearest neighbors) NN can be identified in R' (in our example, $NN = \{u_1\}$) and the corresponding table entries (matrix factorization predictions) can be used for specifying the value orderings for determining a solution for the current user (see Table 9). Note that this approach does not directly predict values of relevance for a user but is used to derive a *variable domain ordering* than can be used by a constraint solver to derive a preferred solution.

5 Research issues

The integration of high-level reasoning in terms of different symbolic AI based approaches with sub-symbolic reasoning is not new but, however, an open challenge in AI-related research areas (Manhaeve et al., 2018). In the following, we summarize major issues for future research specifically related to the topic of integrating machine learning with constraint solving.

Table 7 The $user \times feature$ matrix U

user	f_1	f_2
u_1	0.88	0.0
u_2	0.0	1.15
u_3	0.66	0.30
u_4	0.60	0.36

Features are hidden aspects describing the relationship between user interests and the corresponding variable values. The ‘hidden’ denotes the fact that standard matrix factorization does not associate any semantics with the used features

Table 8 The *feature* $\times$ *attribute value matrix* V describes the dependencies between individual variable values and the hidden features

user	$x_1.v_1$	$x_1.v_2$	$x_2.v_1$	$x_2.v_2$	$x_3.v_1$	$x_3.v_2$	$x_3.v_3$
f_1	0.12	0.98	1.33	0.0	0.92	0.20	0.0
f_2	0.95	0.0	0.07	0.73	0.01	0.88	0.0

Algorithm Selection In constraint solving scenarios, machine learning has the potential to significantly improve the overall performance of solvers, for example, in terms of *lower runtimes*, but also in terms of an *increased prediction quality*. Important to mention is that machine learning techniques helped to change the focus from optimizing individual solvers (algorithms) to the development of *algorithm selection architectures* that flexibly select the best fitting algorithm (or algorithm bundle) – this approach also helps to improve flexibility and applicability in the context of new problem settings. Quite in line with this, constraint solving benchmarks and competitions⁶ help to foster new developments on the level of individual constraint solvers as well as on the level algorithm selection and parametrization. A research issue in the context of algorithm selection is to take into account criteria beyond solver runtime performance, for example, solvers should also be able to *predict user preferences* – well-established techniques that can serve as a starting point can be found in recommender systems related research (Falkner et al., 2011).

Evaluation Metrics for Constraint-based Systems Predominant evaluation criteria used in the constraint solving literature are *performance-related*, for example, the time needed to find a solution, and *accuracy-related*, for example, classifying the satisfiability of a given SAT problem. Only a few approaches focus on aspects that go beyond basic performance analyses and classification accuracy (Erdeniz et al., 2019). Especially in the context interactive constraint-based applications (e.g., configurators), prediction quality has also to be measured in terms of the fit of a proposed solution with the wishes and needs of a user. Such evaluations are typically conducted on the basis of datasets that provide a so-called *ground truth* or *gold standard*. A major issue for future work is to define evaluation metrics that cover a broad range of different scenarios and thus form the basis of a more detailed analysis of the quality of constraint-based systems. Compared to algorithm evaluation techniques existent, for example, in recommender systems (Gunawardana & Shani, 2015), evaluation metrics for constraint-based systems still need to be developed or at least adapted and/or extended. For example, prediction quality can be measured in terms of the share of correct item predictions. In constraint-based scenarios, there are different types of prediction tasks that need to be covered by evaluation metrics. For example, constraint solvers can be used to predict/propose values for single variables but also values for groups of variables and in the “extreme” case propose completions, i.e., value settings for variables that have not been instantiated up to now. Diagnosis scenarios experience a similar situation in terms of being in the need of standard evaluation metrics that help to increase comparability. For example, diagnosis in constraint-based applications is often used in interactive settings to proactively support users in finding out a way from the *no solution could be found* dilemma (Felfernig et al., 2012). Evaluation metrics needed in such contexts go beyond standard metrics and should enable, for example, the measurement of diagnosis minimality.

⁶See, for example, www.minizinc.org.

Table 9 The resulting matrix $R' = U \times V$ that approximates the original matrix R

user	$x_1.v_1$	$x_1.v_2$	$x_2.v_1$	$x_2.v_2$	$x_3.v_1$	$x_3.v_2$	$x_3.v_3$
u_1	0.11	0.86	1.17	0.0	0.81	0.17	0.0
u_2	1.10	0.0	0.08	0.83	0.01	1.02	0.0
u_3	0.37	0.65	0.90	0.22	0.61	0.40	0.0
u_4	0.41	0.59	0.83	0.26	0.56	0.44	0.0
$u_{current}$	0	1	1	0	?:1	?:2	?:3

The current user ($u_{current}$) who currently interacts with the recommender has already selected values for the variables x_1 and x_2 . The most similar user to $u_{current}$ in R' is u_1 . The predicted interests for $u_1 \times x_3$, i.e. [0.83, 0.17, 0.0] can be used to derive a variable domain ordering for the solver ($v_1 : 1, v_2 : 2, v_3 : 3$), i.e., v_1 will be the first value selected by the solver

Learning Configuration Spaces Learning configuration/solution spaces is an important aspect to be taken into account in the context of developing constraint-based systems (Alves Pereira et al., 2019). For example, in the context of software engineering scenarios, variability aspects are often represented in terms of feature models which specify functional aspects in terms of how individual components can be combined with each other (e.g., operating system packages or source components in the compilation phase). In the sampling phase of such systems, solutions have to be identified that (1) satisfy the constraints specified in the feature model but (2) also take into account non-functional criteria (often not covered by the feature model) such as runtime and stability of the generated solutions. Since it is not possible to test the whole configuration space (specifically for reasons of time limitations), a major research challenge is to optimize individual sampling strategies in order to provide a stable dataset that allows machine learning to build a generalized model. In this context, research questions such as *what is the optimal sample size?* and *how can we measure the overall quality of a given sample?* have to be answered. For an in-depth overview of applications and open research issues in *configuration space learning* we refer to Alves Pereira et al. (2019).

Testing and Debugging Constraint-based Systems Similar to Software Engineering scenarios, testing and debugging also plays a major role in the development of constraint-based systems (Felfernig et al., 2004). For example, before deploying a configuration system in a productive environment, knowledge engineers have to assure the correctness of the underlying configuration knowledge base. Faulty configurations contradicting the product domain knowledge can lead to situations where even faulty products are delivered to a customer. In order to avoid such situations, intelligent testing and debugging mechanisms are needed to help to automatically identify faulty parts in a knowledge base. Existing approaches to the automated testing and debugging of constraint-based systems focus on a consistency-based approach where predefined test cases derived, for example, from already completed configuration sessions, are used for inducing conflicts in a knowledge base. Corresponding diagnoses indicate potential sources of the faulty behavior in the knowledge base (Felfernig et al., 2004). A research issue in this context is how to automatically generate test cases. Machine learning concepts can support the automated generation of test cases by learning to predict faulty regions in a knowledge base on the basis of insights from already existing development and maintenance iterations. In a similar fashion, such concepts should help to indicate potentially faulty regions of a knowledge base that should be tested more intensively compared to other parts of the knowledge base. Features of relevance for machine

learning in this context can come from different sources such as *development process information* (e.g., who worked how long on the adaptation of which constraints and which constraints have not been tested for quite a long time) and *basic complexity metrics* for knowledge bases (Felfernig, 2004).

Constraint-aware Machine Learning Typical machine learning approaches focus on building a prediction model on the basis of a given set of training data. Given a new user, for example, the user of a recommender system, predictions are based on the model derived from a dataset. Especially in the context of constraint solving, the defined constraints must be taken into account and – as a consequence – machine learning must be combined with the constraint solver in one way or another. More generally, there are different ways to take into account constraints in scenarios in which we want to combine constraint solving with machine learning techniques. First, *constraint solving can be regarded as its own layer* accepting the input of an upstream machine learning phase. An example thereof is that a neural network determines context-specific search heuristics and the solver applies these heuristics in a problem-dependent fashion. Second, *constraint violations can be used as a penalty function* in machine learning processes. For example, if we want to predict attribute settings that will be of relevance for a user, we could do this even without activating the constraint solver. Including constraint violations as penalties in machine learning optimization functions helps to build prediction models that implicitly take into account (explicitly mentioned) constraints, however, there is no absolute guarantee that with such an approach predictions are always correct. Third, *constraint enforcing approaches* go one step further and encode domain constraints directly into the machine learning model. In neural networks, dependencies (constraints) can be defined between individual node activations. For example, the constraint $a \leftrightarrow b$ can be directly translated into constraints referring to node activations $highactivation(a) \leftrightarrow highactivation(b)$. Fourth, *data sampling and augmentation* can be used to increase the “quality” of a dataset and thus help to increase the prediction quality of the learned model (Alves Pereira et al., 2019). In this context, important issues for future research include efficient ways to encode constraints into machine learning models in such a way that constraint solving can be (partially) replaced by machine learning (and vice-versa). A related research question is to which extent reasoning on top of a constraint solver, for example, in the form of a model-based diagnosis task, can be directly encoded into a corresponding machine learning model.

Explanations Combining constraint solving with machine learning makes both areas profiting from each other. On the one hand, constraint solving has to deal with situations where different explanations are possible and it is important to figure out the most relevant ones for a user (Felfernig et al., 2009). For example, in the context of interactive constraint-based systems such as configurators it is an important issue to provide mechanisms supporting users to get out from *no solution could be found* dilemmas. Typically, there are many explanation candidates and machine learning from past user interactions can help to significantly improve the prediction quality of explanations. On the other hand, machine learning approaches such as neural networks lack explanation capabilities (Darwiche & Hirth, 2020), i.e., are not able to provide deep explanations that can help to understand the proposed solutions. Combining deep learning with knowledge-based approaches can help to make machine learning based solutions more accessible for users. Consequently, explanation-related research in both areas can profit from each other in terms of (1) exploiting machine learning for predicting relevant explanations and (2) exploiting symbolic AI based approaches to support semantic reasoning about explanations.

6 Conclusions

In this article, we have analyzed and summarized research contributions related to the integration of machine learning with constraint solving including (1) the *basic approaches* of constraint satisfaction, SAT solving, and answer set programming, and (2) *applications* thereof including configuration, diagnosis, and constraint-based recommendation. In this context, we have summarized research contributions that solve basic tasks such as *algorithm selection*, *parametrization*, *selection and parametrization of search heuristics*, *performance prediction*, and the *prediction of satisfiability*. In the majority of the cases, supervised machine learning approaches have been applied for problem solving whereas unsupervised and reinforcement learning approaches are in the minority and often applied in combination with corresponding supervised approaches (Bengio et al., 2021). Topics related to the integration of machine learning with constraint solving can be regarded as a thriving research area which requires research focusing on the combination of symbolic and sub-symbolic AI. In order to stimulate further related research, we have discussed relevant research issues comprising the topics of algorithm selection, evaluation metrics for constraint-based systems, learning configuration (solution) spaces, machine learning aspects in the testing and debugging of constraint-based systems, constraint-aware machine learning, and explanations.

Acknowledgements The presented work has been developed within the research project ParXCell (Machine Learning and Parallelization for Scalable Constraint Solving) which is funded by the Austrian Research Promotion Agency (FFG) under the project number 880657.

Funding Open access funding provided by Graz University of Technology.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Acher, M., Temple, P., Jézéquel, J.-M., Galindo, J., Martinez, J., & Ziadi, T. (2018). VaryLATEX: learning paper variants that meet constraints. In *12th International Workshop on Variability Modelling of Software-Intensive Systems* (pp. 83–88).
- Adorf, H.-M., & Johnston, M. (1990). A discrete stochastic neural network algorithm for constraint satisfaction problems. In *International Joint Conference on Neural Networks* (pp. 917–924).
- Alves Pereira, J., Martin, H., Acher, M., Jézéquel, J.-M., Botterweck, G., & Ventresque, A. (2019). Learning software configuration spaces: a systematic literature review research report 1-44, Univ Rennes, Inria, CNRS, IRISA.
- Ansótegui, C., Sellmann, M., & Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In I. P. Gent (Ed.) *Principles and Practice of Constraint Programming - CP 2009* (pp. 142–157). Berlin: Springer.
- Apt, K. (2003). *Principles of constraint programming*. Cambridge: Cambridge University Press.
- Arbelaez, A., Hamadi, Y., & Sebag, M. (2010). Continuous search in constraint programming. *Autonomous Search*, 1, 53–60.

- Beck, J. C., Prosser, P., & Wallace, R.J. (2004). Variable Ordering Heuristics Show Promise. In M. Wallace (Ed.) *Principles and Practice of Constraint Programming – CP 2004* (pp. 711–715). Berlin: Springer.
- Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning.
- Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2), 405–421.
- In C. Bessiere, L. De Raedt, L. Kotthoff, S. Nijssen, B. O'Sullivan, & D. Pedreschi (Eds.) (2016). *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary approach, volume 10101 of Lecture Notes in Artificial Intelligence*. Berlin: Springer.
- Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Frechette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., & Vanschoren, J. (2015). ASLib: A Benchmark Library for Algorithm Selection. *Artificial Intelligence*, 237, 41–58.
- Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Berlin: Springer.
- Bonfietti, A., Lombardi, M., & Milano, M. (2015). Embedding decision trees and random forests in constraint programming. In L. Michel (Ed.) *Integration of AI and OR Techniques in Constraint Programming* (pp. 74–90). Cham: Springer International Publishing.
- Brailsford, S., Potts, C., & Smith, B. (1999). Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3), 557–581.
- Brewka, G., Eiter, T., & Truszczynski, M. (2011). Answer set programming at a glance. *Communications of the ACM*, 54, 92–103.
- Bünz, B., & Lamm, M. (2017). Graph neural networks and boolean satisfiability. arXiv:1702.03592.
- Burkard, J. (2009). Virginia Tech, Advanced Research Computing, Interdisciplinary Center for Applied Mathematics.
- Burke, R. (2000). Knowledge-based recommender systems. In *Encyclopedia of library and information systems* (p. 2000). Marcel dekker.
- Cameron, C., Chen, R., Hartford, J., & Leyton-Brown, K. (2020). Predicting propositional satisfiability via end-to-end learning. *AAAI Conference on Artificial Intelligence*, 34, 3324–3331.
- Cappart, Q., Moisan, T., Rousseau, L.-M., Prémont-Schwarz, I., & Cire, A. (2020). Combining reinforcement learning and constraint programming for combinatorial optimization.
- Carbonell, J. G., Michalski, R. S., & Mitchell, T.M. (1983). Machine learning: a historical and methodological analysis. *AI Magazine*, 4(3), 69.
- Da Col, G.iacomo., & Teppan, E.C. (2017). Learning constraint satisfaction heuristics for configuration problems. In *19Th international configuration workshop* (pp. 8–11).
- Darwiche, A., & Hirth, A. (2020). On the reasons behind decisions. In *24Th european conference on artificial intelligence (ECAI 2020)* (pp. 712–720).
- Epstein, S., & Freuder, E. (2001). Collaborative learning for constraint solving. In *Intl. Conference on principles and practice of constraint programming, volume 2239 of LNCS* (pp. 46–60).
- Erdeniz, S. P., & Felfernig, A. (2018). Cluster and learn: Cluster-specific heuristics for graph coloring. In *Intl. Conference on the practice and theory of automated timetabling* (pp. 401–404). Elsevier.
- Erdeniz, S. P., Felfernig, A., & Atas, M. (2018). Learndiag: A Direct Diagnosis Algorithm Based On Learned Heuristics. In *Joint german/austrian conference on artificial intelligence (künstliche intelligenz)* (pp. 190–197). Springer.
- Erdeniz, S. P., Felfernig, A., & Atas, M. (2019). Matrix factorization based heuristics for direct diagnosis. In *Intl. Conference on industrial, engineering and other applications of applied intelligent systems (IEA/AIE'19)*. Springer.
- Erdeniz, S. P., Felfernig, A., Atas, M., Tran, T. N. T., Jeran, M., & Stettinger, M. (2017). Cluster-Specific Heuristics for constraint solving. In *Intl. Conference on industrial, engineering and other applications of applied intelligent systems* (pp. 21–30). Springer.
- Erdeniz, S. P., Felfernig, A., Samer, R., & Atas, M. (2019). Matrix Factorization based Heuristics for Constraint-based Recommenders. In *34Th ACM/SIGAPP symposium on applied computing (SAC'19)* (pp. 1655–1662). ACM.
- Falkner, A., Felfernig, A., & Haag, A. (2011). Recommendation technologies for configurable products. *AI Magazine*, 32(3), 99–108.
- Felfernig, A. (2004). Effort estimation for knowledge-based configuration systems. In *16Th intl. Conference on software engineering and knowledge engineering (SEKE2004)* (pp. 148–155). Canada: Banff.
- Felfernig, A., & Burke, S. R. (2008). Constraint-based recommender systems: technologies and research issues. In *10th Intl. Conference on Electronic commerce, ICEC '08, New York, NY USA*. Association for Computing Machinery.

- Felfernig, A., Friedrich, G., Jannach, D., & Stumptner, M. (2004). Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2), 213–234.
- Felfernig, A., Friedrich, G., Jannach, D., & Zanker, M. (2006). An integrated environment for the development of knowledge-based recommender applications. *International Journal of Electronic Commerce (IJEC)*, 11(2), 11–34.
- Felfernig, A., Friedrich, G., Schubert, M., Mandl, M., Mairitsch, M., & Teppan, E. (2009). Plausible repairs for inconsistent requirements. In *21st international joint conference on artificial intelligence (IJCAI'09)* (pp. 791–796). California: Pasadena.
- Felfernig, A., Hotz, L., Bagley, C., & Tihihonen, J. (2014). *Knowledge-Based Configuration: From Research to Business Cases*. Burlington: Morgan Kaufmann.
- Felfernig, A., Le, V., Popescu, A., Uta, M., Tran, T., & Atas, M. (2021). An overview of recommender systems and machine learning in feature modeling and configuration, (pp. 1–8). Austria: ACM.
- Felfernig, A., Schubert, M., & Zehentner, C. (2012). An efficient diagnosis algorithm for inconsistent constraint sets. *Artificial intelligence for engineering design. Analysis, and Manufacturing (AIEDAM)*, 26(1), 53–62.
- Felfernig, A., Walter, R., Galindo, J. A., Benavides, D., Erdeniz, S. P., Atas, M., & Reiterer, S. (2018). Anytime diagnosis for reconfiguration. *Journal of Intelligent Information Systems*, 51(1), 161–182.
- Fleischanderl, G., Friedrich, G., Haselböck, A., Schreiner, H., & Stumptner, M. (1998). Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems*, 13(4), 59–68.
- Freuder, E. (1997). In pursuit of the holy grail. *Constraints*, 2, 57–61.
- Fritsch, R., Fritsch, R., Fritsch, G., & Fritsch, G. (1998). *Four-Color Theorem*. Berlin: Springer.
- Galassi, A., Lombardi, M., Mello, P., & Milano, M. (2018). Model Agnostic Solution of CSPs via Deep Learning: A Preliminary Study. In W.-J. van Hoeve (Ed.) *Integration of constraint programming, artificial intelligence, and operations research* (pp. 254–262). Springer.
- Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M. T., & Ziller, S. (2011). A portfolio solver for answer set programming: Preliminary report. In J. P. Delgrande, & W. Faber (Eds.) *Logic Programming and Nonmonotonic Reasoning* (pp. 352–357). Berlin: Springer.
- Gent, I., Jefferson, C., Kotthoff, L., Miguel, I., Moore, N., Nightingale, P., & Petrie, K. (2010). Learning when to use lazy learning in constraint solving. *Frontiers in Artificial Intelligence and Applications*, 215, 873–878.
- Gent, I. P., Kotthoff, L., Miguel, I., & Nightingale, P. (2010). Machine learning for constraint solver design – a case study for the alldifferent constraint. In *3rd workshop on techniques for implementing constraint programming systems (TRICS)* (pp. 13–25).
- Grasch, P., Felfernig, A., & Reinfrank, F. (2013). Recommend: Towards Critiquing-based Recommendation with Speech Interaction. In *7th ACM conference on Recommender systems (RecSys '13)* (pp. 157–164). China: ACM.
- Gu, J., Purdom, P. W., Franco, J., & Wah, B.W. (1996). Algorithms for the satisfiability (SAT) problem: a survey. In *DIMACS Series in discrete mathematics and theoretical computer science* (pp. 19–152). American Mathematical Society.
- Guerri, A., & Milano, M. (2004). Learning techniques for automatic algorithm portfolio selection. In *16th European Conference on Artificial Intelligence, ECAI'04* (pp. 475–479). NLD: IOS Press.
- Gunawardana, A., & Shani, G. (2015). Evaluating Recommender Systems. In F. Ricci, L. Rokach, & B. Shapira (Eds.) *Recommender Systems Handbook* (pp. 265–308). Springer.
- Haim, S., & Walsh, T. (2009). Restart strategy selection using machine learning techniques. In O. Kullmann (Ed.) *Theory and Applications of Satisfiability Testing - SAT 2009* (pp. 312–325). Berlin: Springer.
- Hutter, F., Hamadi, Y., Hoos, H., & Leyton-Brown, K. (2006). Performance prediction and automated tuning of randomized and parametric algorithms. In *Intl. Conference on principles and practice of constraint programming (CP 2006)*, (Vol. 4204 pp. 213–228).
- Hutter, F., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206, 79–111.
- Jannach, D. (2013). Toward Automatically Learned Search Heuristics for CSP-encoded Configuration Problems - Results from an Initial Experimental Analysis. In *15th international configuration workshop* (pp. 9–13).
- Johnston, M., & Minton, S. (1994). Analyzing a heuristic strategy for constraint satisfaction and scheduling. *Intelligent Scheduling*, 257–289.
- Junker, U. (2004). QuickXPlain: preferred explanations and relaxations for over-constrained problems. In *AAAI 2004* (pp. 167–172). AAAI.
- Konstan, J., Miller, B., Herlocker, J., Gordon, L., & Riedl, J. (1997). GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, 40(3), 77–87.

- Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8), 30–37.
- Korte, B., & Vygen, J. (2000). *Combinatorial Optimization – Theory and Algorithms*. Berlin: Springer.
- Kotthoff, L. (2014). Algorithm Selection for Combinatorial Search Problems: A Survey. *AI Magazine*, 35(3), 48–60.
- Kumar, V. (1992). Algorithms for Sonstraint-Satisfaction Problems: A Survey. *AI Magazine*, 13(1), 32–44.
- Kurin, V., Godil, S., Whiteson, S., & Catanzaro, B. (2019). Improving SAT solver heuristics with graph networks and reinforcement learning. arXiv:1909.11830.
- Lagoudakis, M., & Littman, M. (2001). Learning to select branching rules in the DPLL procedure for satisfiability. *Electronic Notes in Discrete Mathematics*, 9, 344–359.
- Lallouet, A., & Legtchenko, A. (2005). Two contributions of constraint programming to machine learning. In *European conference on machine learning (ECML)*, volume 3720 of LNCS (pp. 617–624).
- Lederman, G., Rabe, M., Seshia, S., & Lee, E.A. (2020). Learning heuristics for quantified boolean formulas through reinforcement learning. In *Intl. Conference on learning representations*.
- Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6), 489–506.
- Liang, J., Ganesh, V., Poupard, P., & Czarnecki, K. (2016). Learning rate based branching heuristic for SAT solvers. In *Intl. Conference on theory and applications of satisfiability testing (SAT 2016)* (pp. 123–140).
- Lombardi, M., & Milano, M. (2018). Boosting combinatorial problem modeling with machine learning. In *27th intl Joint conf on artificial intelligence (IJCAI 2018)* (pp. 5472–5478).
- Loreggia, A., Malitsky, Y., Samulowitz, H., & Saraswat, V. (2016). Deep learning for algorithm portfolios. In *AAAI Conference on artificial intelligence, AAAI'16* (pp. 1280–1286). AAAI Press.
- Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., & Raedt, L.D. (2018). Deepproblog: neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.) *Advances in Neural Information Processing Systems* (pp. 3749–3759).
- Maratea, M., Pulina, L., & Ricca, F. (2012). Applying machine learning techniques to ASP solving. In *28th intl. Conference on logic programming (ICLP 2012)*, (Vol. 17 pp. 37–48).
- Mouhoub, M., & Jafari, B. (2011). Heuristic Techniques for Variable and Value Ordering in CSPs. In *13th conference on genetic and evolutionary computation* (pp. 457–464). ACM.
- Myllylä, V., Tihoonen, J., Raatikainen, M., & Felfernig, A. (2014). Using answer set programming for feature model representation and configuration. In *International workshop on configuration, volume 1220 of CEUR workshop proceedings* (pp. 1–8).
- Nareyek, A. (2004). *Choosing search heuristics by Non-Stationary reinforcement Learning*, (pp. 523–544). USA: Kluwer Academic Publishers.
- O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., & O'Sullivan, B. (2013). Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving? In *Irish Conf. on Artificial Intelligence and Cognitive Science* (pp. 210–216).
- Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. USA: Addison-Wesley Longman Publishing Co. Inc.
- Raedt, L. D., Nijssen, S., O'Sullivan, B., & Hentenryck, P.V. (2011). *Constraint Programming meets Machine Learning and Data Mining* Vol. 1. Germany: Dagstuhl Reports.
- Raedt, L. D., Passerini, A., & Teso, S. (2018). Learning constraints from examples. In *32nd AAAI conference on artificial intelligence (AAAI 2018)* (pp. 2–7).
- Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, 23(1), 57–95.
- Rossi, F., Beek, P. V., & Walsh, v. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. USA: Elsevier Science Inc.
- Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5), 285–286.
- Sadeh, N., & Fox, M. S. (1996). Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *AI Journal*, 86(1), 1–41.
- Samulowitz, H., & Memisevic, R. (2007). Learning to solve QBF. In *AAAI Conference on Artificial Intelligence* (pp. 255–260).
- Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., & Dill, D.L. (2019). Learning a SAT solver from Single-Bit supervision.
- Simons, P., Niemelä, I., & Sooinen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138, 181–234.
- Spieker, H., & Gotlieb, A. (2018). Towards Hybrid Constraint Solving with Reinforcement Learning and Constraint-Based Local Search Data Science meets Optimization Workshop at Federated Artificial Intelligence Meeting.

- Temple, P., Acher, M., Jézéquel, J.-M., & Barais, O. (2017). Learning Contextual-Variability models. *IEEE Software*, 34, 64–70.
- Temple, P., Galindo Duarte, J., Acher, M., & Jézéquel, J.-M. (2016). Using machine learning to infer constraints for product lines. In *20Th international systems and software product line conference* (pp. 209–218).
- Tsang, E. P. K. (1993). *Foundations of Constraint Satisfaction. Computation In cognitive science*. Cambridge: Academic Press.
- Ulz, T., Schwarz, M., Felfernig, A., Haas, S., Shehadeh, A., Reiterer, S., & Stettinger, M. (2016). Human Computation for Constraint-based Recommenders. *Journal of Intelligent Information Systems (JIIS)*, 49, 37–57.
- Uta, M., & Felfernig, A. (2020). Towards machine learning based configuration. In C. Forza, L. Hvam, & A. Felfernig (Eds.) *22nd International Configuration Workshop* (pp. 25–28).
- Uta, M., Felfernig, A., Le, V.-M., Popescu, A., Tran, T., & Helic, D. (2021). Evaluating recommender systems in feature model configuration. In *25th ACM International Systems and Software Product Line Conference (SPLC '21)*. United Kingdom: ACM.
- Wang, C., & Tsang, E. (1991). Solving Constraint Satisfaction Problems Using Neural Networks. In *2nd Intl Conference on Artificial Neural Networks* (pp. 295–299).
- Wen, J., Khan, M., Che, M., Yan, Y., & Yang, G. (2020). Constraint solving with deep learning for symbolic execution.
- Wohlin, C. (2014). Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *18th Intl Conference on Evaluation and Assessment in Software Engineering (EASE'14)* (pp. 1–10).
- Xu, H., Koenig, S., & Kumar, T.K.S. (2018). Towards effective deep learning for constraint satisfaction problems. In J. N. Hooker (Ed.) *24th Intl. Conference on Principles and Practice of Constraint Programming (CP Lille, France, August 27-31, 2018, volume 11008 of Lecture Notes in Computer Science* (p. 2018). Springer.
- Xu, L., Hoos, H., & Leyton-Brown, K. (2012). Predicting satisfiability at the phase transition. In *AAAI Conference on artificial intelligence*, (Vol. 1 pp. 584–590).
- Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2008). SATZilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research*, 32, 565–606.
- Xu, Y., Stern, D., & Samulowitz, H. (2009). Learning adaptation to solve constraint satisfaction problems. In *LION 2009, Learning and intelligent optimization*.
- Yang, Z., Ishay, A., & Lee, J. (2020). NeurASP: embracing neural networks into answer set programming. In C. Bessiere (Ed.) *29th Intl. Joint Conf. on Artificial Intelligence, IJCAI-20* (pp. 1755–1762).
- Yolcu, E., & Pó, c.zos. B. (2019). Learning local search heuristics for boolean satisfiability. In H. M. Wallach, H. Larochelle, Beygelzimer A., F. D'Alché-Buc, E. B. Fox, & R. Garnett (Eds.) *Neural Information Processing Systems 2019 NeurIPS 2019* (pp. 7990–8001).
- Zanker, M. (2008). A Collaborative Constraint-based Meta-level Recommender. In *ACM Conference on recommender systems* (pp. 139–146). ACM.
- Zanker, M., Aschinger, M., & Jessenitschnig, M. (2007). Development of a collaborative and constraint-based web configuration system for personalized bundling of products and services. In *Intl. Conference on web information systems engineering* (pp. 273–284). Springer.
- Zhang, W., Sun, Z., Zhu, Q., Li, G., Cai, S., Xiong, Y., & Zhang, L. (2020). NLocalSAT: boosting local search with solution prediction. In C. Bessiere (Ed.) *Intl Joint Conference on Artificial Intelligence (IJCAI-20)* (pp. 1177–1183).

Affiliations

Andrei Popescu¹ · Seda Polat-Erdeniz¹ · Alexander Felfernig¹ · Mathias Uta² · Müslüm Atas¹ · Viet-Man Le¹ · Klaus Pils³ · Martin Enzelsberger³ · Thi Ngoc Trang Tran¹

Seda Polat-Erdeniz
spolater@ist.tugraz.at

Alexander Felfernig
afelfern@ist.tugraz.at

Mathias Uta
mathias.uta@siemens.com

Müslüm Atas
muatas@ist.tugraz.at

Viet-Man Le
vietman.le@ist.tugraz.at

Klaus Pils
klaus.pils@combeeneration.com

Martin Enzelsberger
martin.enzelsberger@combeeneration.com

Thi Ngoc Trang Tran
ttrang@ist.tugraz.at

¹ Institute of Software Technology (Graz University of Technology), Graz, Austria

² Siemens Energy Global GmbH, Erlangen, Germany

³ Combeeneration GmbH, Sarleinsbach, Austria