Boosting Combinatorial Problem Modeling with Machine Learning

Michele Lombardi and **Michela Milano**
DISI-University of Bologna
michele.lombardi@unibo.it, michela.milano@unibo.it

Abstract

In the past few years, the area of Machine Learning (ML) has witnessed tremendous advancements, becoming a pervasive technology in a wide range of applications. One area that can significantly benefit from the use of ML is Combinatorial Optimization. The three pillars of constraint satisfaction and optimization problem solving, i.e., modeling, search, and optimization, can exploit ML techniques to boost their accuracy, efficiency and effectiveness. In this survey we focus on the modeling component, whose effectiveness is crucial for solving the problem. The modeling activity has been traditionally shaped by optimization and domain experts, interacting to provide realistic results. Machine Learning techniques can tremendously ease the process, and exploit the available data to either create models or refine expert-designed ones. In this survey we cover approaches that have been recently proposed to enhance the modeling process by learning either single constraints, objective functions, or the whole model. We highlight common themes to multiple approaches and draw connections with related fields of research.

1 Introduction

Inferring models from observations and studying their properties is what science is about. Models can be descriptive or interpretative, thus enabling the understanding of a system/process behaviour. Models can be predictive, thus describing the future evolution of the system/process dynamic. Models can be prescriptive, thus providing decision support and assessing the effects of decisions on the system/process evolution in the medium or long term. All models are just approximations of the system/process they represent, but some of them are useful as they provide insights to describe, forecast and decide on the system. Their accuracy is essential for the task they have been built for.

In this paper *we focus on the specific case of combinatorial optimization models*, exhibiting discrete decision variables, a combinatorial structure defined by constraints and an objective function showing the direction for improving the solution quality. These models have been traditionally designed via a close interaction between optimization and domain experts, the former encoding the knowledge of the latter in mathematical or symbolic expressions. Combinatorial model design is indeed an iterative process, where subsequent model versions are evaluated by domain experts; these experts assess the model effectiveness and efficiency, and possibly provide feedback to the optimization expert for further refinement. Uncertainty can be taken into account or not, eventually resulting in either stochastic or deterministic models.

Nowadays, in many application domains we have access to data of unprecedented scale and accuracy about the system/process we want to model. Machine Learning (ML) provides techniques to exploit available data and extract value and useful information, which can be employed for modeling and solving combinatorial problems. The field has been widely explored in the last decade, as witnessed by the increasing number of researchers working in using ML in optimization. A good survey can be found in the edited collection [Bessiere *et al.*, 2016] on the intersection of Constraint Programming with Machine Learning and Data Mining.

In this paper *we provide a survey on methods for boosting the combinatorial modeling activity through ML methods and algorithms*. ML techniques can be used to learn either constraints that define the combinatorial structure of the problem, or objective functions describing optimization criteria. In this way, not only we learn part of the model from data, but we also have information about the model accuracy, which is otherwise inaccessible if the model is defined through the interaction with domain experts. We review the recent literature on the use of ML to improve the model or learn it when a declarative form is hard to shape even by domain experts.

The paper is organized as follows: Section 2 contains the bulk of the content and deals with approaches that employ ML to obtain (part of) a combinatorial model. In Section 3 we present approaches relying on an internal approximate models and active learning to directly solve a problem. In Section 4 we draw connections with some related areas, most notably black-box optimization (based on surrogate models) and system identification in Control Theory. In Section 5 we describe some horizontal topics common to the discussed approaches that represent important open research directions.

Finally, we open up a bit the scope of the paper by showing in section 6 a picture of the many directions for the integration of ML in combinatorial optimization: this includes search and

optimization, portfolio selection and algorithm configuration. Due to space limitations, we do not discuss the (many and interesting) techniques that explore the other direction of integration (i.e. optimization techniques to improve ML algorithms). Concluding remarks are provided in Section 7.

2 Learning Model Components

In this section we consider approaches that use implicit information from a set of examples to obtain part of a combinatorial model, in particular a constraint or an objective function. There are two main approaches to achieve this result: the first (covered in Section 2.1) aims at extracting information using the *native constraint language* of the solver; the second (in Section 2.2) consists in *embedding a fully-fledged Machine Learning model* in a combinatorial approach. As an extreme case, in Section 2.3 we consider approaches where the ML model makes up (almost) all of the combinatorial model.

2.1 Learning via a Native Constraint Language

Traditionally, combinatorial optimization models are born from iterative interactions between a domain and an optimization expert. Intuitively, ML could support modeling activities by retaining the same process, but replacing the optimization expert with a *constraint acquisition algorithm*, and the domain expert with an example generator. Formally, the approaches in this section aim at learning a model in the form:

$$\begin{aligned} \min z = x_0 \qquad & (\mathbf{P1}) \\ \text{subject to: } \pi_i(\vec{x}) \qquad & \forall i \in I \\ \vec{x} \in D_{\vec{x}} & \end{aligned}$$

were $\vec{x}$ is the vector of problem variables and $D_{\vec{x}}$ their domain. The set I contains the indices of all problem constraints, represented here as predicates $\pi_i(\vec{x})$ that must hold in any feasible solution. The x_0 variable represents by convention the cost to be minimized, and is absent in pure constraint satisfaction problems. Crucially, *the predicates are defined using the building blocks from the hosting approach* (e.g. global constraints in Constraint Programming, linear equalities or inequalities in Mixed Integer Linear Programming): what changes is just the way they are discovered.

The example generator may be a human, a collection of data, or an existing automated system. In latter case, constraint acquisition can also be considered as a mean to explain in declarative terms the behavior of a procedural or subsymbolic decision support system. All approaches in this section focus on learning constraints, rather than objective functions: of course nothing prevents a constraint from representing the definition of the x_0 variable (as done in P1).

This is the idea behind systems such as CONACQ [Bessiere *et al.*, 2017a] (in its various versions), QUACQ [Bessiere *et al.*, 2013], and model seeker [Beldiceanu and Simonis, 2012], which build over the Constraint Programming paradigm, and behind the method in [Lallouet *et al.*, 2010], based on Inductive Logic Programming. Both CONACQ and QUACQ operate by picking constraints from a set of potential (instantiated) candidates (called a *bias*) and adding them to a target constraint network. Model seeker attempts to match (possibly) transformed subsets of variables in the training examples against a collection of (non-instantiated) global constraints; constraints that are compatible with all examples are added to the current model, and a series of simplification steps attempts to remove redundant relations. Since model seeker relies does not need to consider explicitly all possible instantiations in its candidate pool, it can usually deal with a large variety of constraints. The downside is that finding a matching becomes more complicated and requires the use of a heuristic step. The method from [Lallouet *et al.*, 2010] attempts instead to learn local rules that are partially independent on the specific values and variables appearing in the examples.

A first major design choice in all such approaches concerns the use of passive or active learning. Methods based on passive learning (e.g. Model seeker, the original CONACQ, and the one from [Lallouet *et al.*, 2010]) operate on a fixed collection of examples. Conversely, approaches based active learning (e.g. QUACQ and CONACQ.2) generate candidate examples themselves and query the generator (which in this case is instead a constraint checker) for their validity. Active learning enables convergence using a smaller number of examples, but is not applicable when a constraint checker is not available (e.g. when working on collections of historical data).

Among the mentioned approaches, only model seeker can work using just positive examples, which makes it well suited to deal with historical data. The system can also be used to obtain, based on a handful of examples, a candidate list of global constraints for modeling the problem. CONACQ and QUACQ employ both positive and negative examples (which may be a disadvantage), but are capable of using negative examples to quickly rule out large sets of constraints from the bias (which is a considerable advantage). QUACQ has the peculiarity of relying on *partial* examples, where only some of the problem variables are instantiated. This allows to speed up convergence both from a theoretical and practical perspective, giving the algorithm its namesake (QUick ACQuisition).

Finally, the method from [Lallouet *et al.*, 2010] learns a model using an intermediate representation; this is loosely inspired by modeling languages such as AMPL, OPL or MiniZinc, which make a clear distinction between the problem structure and its parameters. Thanks to this design choice, the approach is able to learn parameter-free models, and to generalize results obtained on smaller instances to larger ones. The price to pay for this impressive feat is a more complex formalism and a somewhat reduced expressivity.

2.2 Incorporating Machine Learning Models

Unlike the approaches described in Section 2.1, the methods considered here attempt to incorporate a fully-fledged Machine Learning model within a combinatorial optimization model. Formally, these works deal with problems in the form:

$$\begin{aligned} \min z = x_0 \qquad & (\mathbf{P2}) \\ \text{subject to: } \pi_i(\vec{x}) \qquad & \forall i \in I \\ \nu_m(\vec{x}_{m,in}, \vec{x}_{m,out}) \qquad & \forall m \in M \\ \vec{x} \in D_{\vec{x}} & \end{aligned}$$

where the $\pi_i(\vec{x})$ predicates represent constraints obtained in a traditional fashion, while each $\nu_m(\vec{x}_{m,in}, \vec{x}_{m,out})$ is a predicate that: 1) corresponds to a Machine Learning model m

from a set M; and 2) is satisfied iff the value of the input and output variables match the evaluation of the ML model, i.e.:

$$\nu_m(\vec{x}_{m,in}, \vec{x}_{m,out}) \Leftrightarrow \vec{x}_{m,out} = m(\vec{x}_{m,in})$$

The ML components (defining either constraints or the objective function) are integrated with the rest of the optimization model in a seamless fashion. The emphasis is not on how the ML models are obtained, but on methods for embedding them efficiently and effectively into a combinatorial model. This is the key idea in Empirical Model Learning [Lombardi *et al.*, 2017] and can be achieved by either expanding or exploiting the constraint language. We will rely on this distinction to group approaches in this section.

Embedding a ML model by expanding the language is a natural solution in the Constraint Programming domain: it requires to introduce a new modeling block (e.g. a new global constraint) and to define an operational semantic (e.g. a propagator). For example, [Lombardi *et al.*, 2017] embed a pre-trained Neural Network in CP by associating a "Neuron Constraint" to each network unit, and using interval-based reasoning to prune the input/output variables. The approach is extended in [Lombardi and Gualandi, 2016] to two-layer networks via a Lagrangian relaxation. A similar approach is taken in [Lallouet and Legtchenko, 2007] and related references by the same authors. In these works, however, the starting point is a collection of examples that implicitly (and approximately) define a constraint. Then, a set of ML classifiers (either Neural Networks or Decision Trees) are learned for checking the consistency of each variable-value pair. Interval based reasoning is then used to generalize the classifiers so that they can work with unbound variables.

The second main method to embed a ML component in a combinatorial model consists in *encoding the ML model using the native language* offered by the optimization technology (e.g. linear constraints and integer variables in Mixed Integer Linear Programming, or linear constraints and boolean predicates in Satisfiability Modulo Linear Real Arithmetic). This encoding is formally a decomposition of the $\nu_m(\vec{x}_{m,in}, \vec{x}_{m,out})$ predicates and should be not only correct, but also effective at supporting the solver at search time.

A simple encoding for Decision (and Regression) Trees in CP and Satisfiability Modulo Theories (LRA in particular) is proposed in [Lombardi *et al.*, 2017], in the context of a thermal-aware workload assignment problem: the encoding is based on modeling each root-to-leaf path as an implication, with the addition of a few redundant constraints. A wider range of CP encodings, based on Multi-Valued Decision Diagrams/TABLE constraints plus discretized numeric attributes, is instead considered in [Bonfietti *et al.*, 2015], and benchmarked on the same target problem: these encodings are more complex and computationally expensive, but they also enforce a *much* stronger level of consistency.

A MILP encoding for Decision/Regression Trees (based on associating a binary variable to each root-to-leaf path in the tree) is described in [Verwer *et al.*, 2017], and employed within an auction optimization problem. The encoding uses a small number of integer variable, which is effective at limiting branching, but relies on big-Ms for the linearization of disjunctions, weakening the Linear Programming relaxation.

CP encodings for Random Forests are briefly considered in [Bonfietti *et al.*, 2015], although with limited effectiveness. A multi-step HVAC control problem which employs a Deep Neural Network to model the state transition function is considered in [Say *et al.*, 2017]: the authors focus on networks based on REctifier Linear Units (ReLUs). For these they provide a MILP encoding strengthened 1) by simple redundant constraints, and 2) by a pre-processing step that sparsifies the network to boost the efficiency of the underlying solver.

2.3 ML Models as Problem Models

As an extreme case, we consider works where a ML model (almost) entirely replaces a classical combinatorial optimization model. This happens chiefly when optimization is used for generating counterexamples or for safety verification, and makes such approaches closer to the idea of using optimization to support ML tasks. Formally, approaches in this group deal with problems in the form:

$$\begin{aligned} &\min z = x_0 && (\mathbf{P3}) \\ &\text{subject to: } \nu_m(\vec{x}_{m,in}, \vec{x}_{m,out}) && \forall m \in M \\ &\vec{x} \in D_{\vec{x}} \end{aligned}$$

In practice, simple additional constraints on $\vec{x}_{m,in}$ and $\vec{x}_{m,out}$ are usually supported. The main point, however, is that the focus on a specific model structure (e.g. a Neural Network) allows for tailored optimization techniques.

In this context, [Fischetti and Jo, 2018] model a ReLU based Deep Neural Network using MILP and bound tightening. Starting from a given (correctly classified) example, the approach searches for a minimally-distant input perturbation that invalidates the classification. The method is general, but applied to image classification as a case study. The use of bound tightening proved crucial for the method effectiveness.

In the SMT domain, [Huang *et al.*, 2017] introduce an approach for generating counterexamples for image classification tasks. The authors improve scalability and obtain more meaningful results by replacing basic decisions (i.e. choosing the color of each pixel) with image manipulation operators, applied to an original (correctly classified) example. Conversely, [Katz *et al.*, 2017] follow a more fundamental, and low-level, approach by explicitly writing a theory solver for Linear Real Arithmetic augmented with ReLUs. The solver attempts to deal with each ReLU in the form $y = \max(0, x)$ by updating x or y to satisfy the relation, and then proceeding with the normal Simplex algorithm. Actual disjunctions are introduced only when a maximum number of updates has been performed on a given ReLU. The method exhibits very good scalability and is successfully employed to verify properties of an unmanned aircraft control system.

3 Learning while Solving Problems

In this section we consider approaches that obtain an approximate model via active learning *while searching for an optimal solution*. Promising candidate solutions are identified and evaluated; then, based on the feedback, the internal model is updated and the whole process repeated. Active learning is also employed in constraint acquisition (e.g. QUACQ and CONACQ.2): however, for the approaches in this section, *the*

main outcome of the process is a solution rather than the learnt model. Moreover, these approaches focus on learning an explicit *objective function*, rather than an arbitrary model component. Formally, we deal with problems in the form:

$$\begin{aligned} \min z = f(\vec{x}; \vec{w}) \qquad & (\mathbf{P4}) \\ \text{subject to: } \pi_i(\vec{x}) \qquad & \forall i \in I \\ \vec{x} \in D_{\vec{x}} \qquad & \end{aligned}$$

where f is the objective function, represented using an approximate model with a parameter vector $\vec{w}$ that is learnt at search time. These approaches are employed (e.g.) when dealing with user preferences. In this context, requesting user feedback is referred to as *preference elicitation*, and can involve more than a simple evaluation (e.g. it may require comparisons or explicit changes by the user).

Rather than on preference elicitation in itself (an extensively researched topic), *here we are interested in approaches that use such technique to learn the objective function of a combinatorial problem*. Typically, this is achieved by using preferences to learn a linear combination of "features", i.e.:

$$f(\vec{x}; \vec{w}) = \sum_{j=1}^{m} w_j \phi_j(\vec{x}) \tag{1}$$

where each $\phi_i(\vec{x})$ is a *feature function*. In principle, embedding Equation (1) in combinatorial optimization is easy, as long as the solver can deal with the $\phi_i(\vec{x})$ functions. In practice, identifying a candidate solution requires to estimate both the solution quality *and the degree of uncertainty of the model*: this ensures that promising candidates are not disregarded due to overestimation errors. The need to take into account both these aspects frequently results in restricting assumptions on the supported feature functions.

A method based on SAT Modulo Theories is provided in [Campigotto *et al.*, 2011]. The original approach was cast in [Dragone *et al.*, 2018] as an instance of a more general framework, grounded also using MILP and two more preference elicitation methods by other authors. The paper assumes discrete or linear numerical features. Other approaches may be viable in principle, but actual application examples to non-trivial combinatorial problems are scarce.

The need to speed-up optimization with expensive objective functions (e.g. defined via numerical simulation) has also motivated the use of *surrogate models in heuristic methods*. In this case an internal, approximate model is employed to reduce the number of needed function evaluations. As an example, [Gilan and Dilkina, 2015] use Gaussian process regression to boost the efficiency of a Genetic Algorithm for sustainable building design.

4 Related Areas

In this section we review methods that do not strictly fall within the paper focus (i.e. the use of ML to boost modeling in combinatorial optimization), but are nevertheless closely related and likely of interest for the reader.

The iterative refinement loop described in Section 3 is at the core of *black box optimization approaches based on surrogate models*, recently surveyed in [Vu *et al.*, 2017]. Such methods typically tackle problems where the objective function is expensive to compute (e.g. it is defined via a numerical simulator), and use an internal approximate model to reduce the number of evaluations. The models of choice are traditionally (second degree) multivariate polynomials, kriging, or Radial Basis Functions. The solution methods have roots in Mathematical Programming and emphasize dealing with non-linear functions rather than with complex constraints and discrete variables. Surrogate based optimization is *one of the research areas where active learning for optimization has been better investigated*. We have left these approaches out of Section 3 because their application to combinatorial problems has so far been limited (but the picture is changing rapidly).

The ALAMO system from [Wilson and Sahinidis, 2017] is designed to automate the construction of algebraic models of functions that are expensive to evaluate. The approach relies on active learning, and generates new sampling points by maximizing the estimated error of the current model. It can be considered an extreme form of surrogate based optimization where minimizing the error is the only objective. It was originally designed to learn objective/constraint functions for numeric (rather than combinatorial) optimization problems, and for this reason has been left out of Section 2.1.

OptQuest [Laguna, 2011] interleaves simulation for evaluating the problem objective and optimization (based on scatter search). OptQuest uses predictive models within the search engine to establish promising research directions. These predictors can be either based on multivatiate linear regression or neural networks that are trained during search. The approach has not been considered in the Section 3 because the ML component is not strictly used for modeling.

In Control Theory, similar concepts have been deeply studied in the context of Model Predictive Control [Christofides *et al.*, 2013] (MPC). MPC chooses control actions by repeatedly solving an online constrained optimization problem, which aims at minimizing a performance index over a finite horizon based on predictions obtained by a system model. The model is obtained through a system identification methodology that is capable to accurately predict the system dynamics. System identification starts from a data set, a set of candidate models (the model structure) and a rule by which candidate models can be assessed against data (e.g. the least square selection rule). The selected model is then validated and refined. Despite some similarities between system identification and ML and between MPC and optimization exist, these methods are inherently different in the decisions they take. Control models act on-line by taking and applying decisions to the system. Combinatorial optimization is in general concerned with more strategic decisions that have a longer time horizon and are not applied at optimization time.

5 Common Themes and Shared Issues

In this section we discuss some themes and issues that tend to be shared by all approaches attempting to modeling activities via ML. Some of them have been recognized and tackled (at least in certain subfields), while others have been generally neglected or only recently discovered. Some themes have been briefly discussed in the previous sections, but it is worth

to treat them here in a more systematic fashion.

Dealing with the Model (In)accuracy: When ML makes up part of the problem model, we explicitly acknowledge that the solutions may suffer from approximation errors. This is exacerbated by the fact that *the solutions (or more accurately the ML input configurations) visited at search time may be considerably different from those considered at training time*. There are two main ways to deal with this situation.

First, for approaches that include a passive learning stage, the problem can be mitigated by ensuring that the training set provides a sufficiently uniform coverage of the search space. This can be done by choosing the training data according to Design of Experiments principles (e.g. via Latin hypercube sampling). This approach has been well investigated in surrogate-based optimization, because the availability of a simulator (or the chance to perform physical experiments) provides the opportunity to design an ad-hoc training set.

The second (more rigorous) approach consists in using active learning. The simple approach of evaluating a solution and updating the ML model is sufficient to guarantee convergence, provided that the updates improve the model accuracy in a roughly monotonic fashion. However, the simulators frequently employed in active learning may introduce additional approximation. Finally, both the first and the second method are not easily applicable to collections of historical data.

Optimizer Bias: In the case of optimization (say, minimization) problems, the solver will naturally be attracted by solutions with a low objective value and will actively avoid solutions with large objective value. In practice, *such values may have more to do with approximation errors than with the actual quality of the solution*. In case of an underestimated objective, active learning is sufficient to ensure convergence to high quality solutions in most cases. Dealing with overestimation, however, additionally requires access to information about the model accuracy.

Some techniques (e.g. Gaussian processes, Support Vector Machines) provide this information in a rigorous fashion under specific conditions (e.g. smoothness). For this reason they have been thoroughly investigated in the context of black-box optimization and preference elicitation. For example, assuming that the ML model provides a confidence interval for the problem objective, the solver may be instructed to focus on the lower-end of such interval rather than on the actual prediction. More in general, a solution may be accepted as long as it is *sufficiently likely* to improve over the best known value. Finding effective methods to combine uncertainty and solution quality information is however far from trivial, and the details of this integration are the area where multiple black-box and preference elicitation methods tend to differ the most.

Information about the model uncertainty is in fact available for many ML methods: (Deep) Neural Network classifiers have softmax scores, Decision Trees report misclassified examples on their leaves, Regression Trees can do the same with output variance, and Random Forests provide vote counts. To the best of our knowledge, however, this kind of information has never been thoroughly exploited in optimization.

Passive vs Active Learning: In general, *active learning can provide significant advantages over passive learning* in terms of solution accuracy and convergence. *However, the technique is not without disadvantages*. In first place, active learning requires the ability to evaluate solutions, which is not available for example when working with historical data. Second, training at search time may be prohibitively expensive, depending on the response times that are needed for the considered optimization application. The cost may stem from the time for a function evaluation and from the number of examples needed to obtain a reasonably accurate model.

The last point makes it particularly difficult to combine active learning with data hungry ML models, such as Deep Neural Networks. Due to the effectiveness of DNNs and the practical benefits of active learning, identifying techniques to combine the two is a promising research topic. Intuitively, it would require finding methods for making significant updates to a pre-trained DNN using only a small number of examples.

Deterministic vs Non-Deterministic Systems: Machine Learning is frequently used to model systems that are not deterministic, e.g. that are stochastic or simply exhibit an inherently degree of inconsistency (e.g. user evaluations). In practice, *in both cases the same input example may lead to different results, with multiple evaluations following some kind of probability distribution*.

Many of the methods described in this work (e.g. those in Section 2 and classical black-box optimization) tend to overlook this behavior. In the case of passive learning with historical data, the training test implicitly encodes information about the probability distribution, which ends up mitigating the problem. When the training set is crafted via Design of Experiment, or when active learning is used, extra care should be put when the underlying system is non-deterministic: for example, one could resort to multiple evaluations of the same (or similar) examples, or make use of ML models that are robust w.r.t. inconsistent evaluations. The latter approach is typically employed in preference elicitation.

A more subtle issue is that *a model trained for maximum likelihood over a stochastic system may lead to sub-par results when used in combinatorial optimization*. This fact has been recently recognized in [Donti *et al.*, 2017], and tackled via so called end-to-end task based learning, i.e. by explicitly taking into account the optimizer during passive training. Currently, the approach is viable only for a specific class of (convex) optimization problems, and a straightforward generalization may be very computationally expensive.

More in general, accuracy issues are not the only reason why updating the model may become necessary. The problem specification itself may be incomplete, or change over time. This is explicitly recognized in [Bessiere *et al.*, 2017b], which proposes the Inductive Constraint Programming Loop as a general framework do deal with model updates.

Accuracy vs Optimization: In our opinion, however, the most relevant and less investigated issues relate to recognizing the approximate nature of the model. In first place, *when dealing with an approximate model, the practical value of*

finding global optima appears questionable. This does not hold for approaches that attempt to verify properties of ML models, and holds to a lesser degree for problems where an accurate model is available, but expensive to compute. In all other cases, reaching global optimality may even be detrimental, if the corresponding solution happens to be poor in terms of real world robustness. Intuitively, it should be possible to exploit uncertainty information to stop the search process earlier, depending on the level of accuracy of the model. However, no such attempt has been considered in the literature to the best of our knowledge.

Finally, when the ML model and the combinatorial model need to be co-designed, there exists an interesting *trade-off between model accuracy and suitability for optimization*. Namely, a complex model (e.g. a deeper network) may be more accurate, but more difficult to optimize, possibly leading to worse results in practice. To the best of our knowledge, this kind of trade-off has never been thoroughly investigated.

6 The Use of ML beyond Modelling

Beside the modeling activity, Machine Learning has been exploited also in the solution of the problem by improving search. For instance in Large Neighborhood search an initial solution is gradually improved by alternately destroying and repairing the solution. The destroying part is performed by selecting and fixing some fragment of the solution and the repairing part is based on search. Machine Learning, and in particular, reinforcement learning has been used in [Mairy, 2011] to tune the search limit, the fragment size and the fragment selection procedure.

In more traditional tree search, Machine Learning has been used either to select the best heuristics [Liberto *et al.*, 2016] from a portfolio, or to guide search by estimating the best variable-value selections and/or a bound on the objective: the estimate may come from a model trained off-line over available solutions [Hottung *et al.*, 2017; Galassi *et al.*, 2018], or from sampling (e.g. [Loth *et al.*, 2013]). Both methods can be effective, but sampling may incur significant overhead at solution time, while training off-line may lead to a lack of generality: if the Machine Learning model is trained on a given problem dimension, then it can be used to guide the search in instances of lower or equal dimension, but it does not scale to larger dimension problems. This difficulty is partially overcome in [Bello *et al.*, 2016] by using pointer networks and reinforcement learning, but the technique requires to engineer the network structure for the considered problem, limiting in part its flexibility. In general, the way to generalise these approaches and improve their efficiency is a very intriguing direction for future research.

Machine Learning has also been applied successfully in automatic algorithm portfolio selection. This line of research has received more attention w.r.t. the more recent topics described above. An algorithm portfolio consists of a set of algorithms, and portfolio selection is the problem of choosing the best one for an input instance. A highly successful approach in satisfiability (SAT) is SATzilla [Xu *et al.*, 2008], other known systems are Hydra, CP-Hydra, and SMAC. It is based on the identification of features that characterize problem instances and are related to instance hardness. In general this cannot be done automatically, but rather by capturing the knowledge of a domain expert. Among all approaches developed for this purpose, SMAC stands out for its use of active learning and an approximate model trained at search time (see [Hutter *et al.*, 2011]). Other works have focused instead on using ML to choose the best technique to solve a sub-problem, and specifically to choose the best propagator for a given constraint in CP (e.g. [Balafrej *et al.*, 2015]).

ML techniques are also applicable in other contexts like performance predictions (see e.g. [Hutter *et al.*, 2014], [Malitsky and Sellmann, 2012]) to devise a schedule with time allocations for each algorithm in the portfolio, which can then be applied sequentially or in parallel.

7 Conclusions

The hybridization of ML and optimization is a broad research field that has been gaining considerable popularity in recent years. This topic has been so far tackled in relative isolation in multiple domains, somehow hindering the research progress. In this work, we have provided a first cross-disciplinary survey focused on *the use ML to boost the modeling activity of combinatorial problems*. We have attempted a classification and highlighted closely related fields. More importantly, we have identified a number of common themes and issues that have been addressed in different context, which may serve as a guide to promote cross-fertilization efforts.

References

[Balafrej *et al.*, 2015] Amine Balafrej, Christian Bessière, and Anastasia Paparrizou. Multi-armed bandits for adaptive constraint propagation. In *Proc. of IJCAI*, pages 290–296, 2015.

[Beldiceanu and Simonis, 2012] Nicolas Beldiceanu and Helmut Simonis. A model seeker: Extracting global constraint models from positive examples. In *Proc. of CP*, pages 141–157, 2012.

[Bello *et al.*, 2016] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.

[Bessiere *et al.*, 2013] Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Constraint acquisition via partial queries. In *Proc. of IJCAI*, pages 475–481, 2013.

[Bessiere *et al.*, 2016] Christian Bessiere, Luc De Raedt, Lars Kotthoff, Siegfried Nijssen, Barry O'Sullivan, and Dino Pedreschi, editors. *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary approach*. Springer, 2016.

[Bessiere *et al.*, 2017a] Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. *Artifcial Intelligence*, 244:315–342, 2017.

[Bessiere *et al.*, 2017b] Christian Bessiere, Luc De Raedt, Tias Guns, Lars Kotthoff, Mirco Nanni, Siegfried Nijssen,

Barry O'Sullivan, Anastasia Paparrizou, Dino Pedreschi, and Helmut Simonis. The inductive constraint programming loop. *IEEE Intelligent Systems*, 32(5):44–52, 2017.

[Bonfietti *et al.*, 2015] Alessio Bonfietti, Michele Lombardi, and Michela Milano. Embedding decision trees and random forests in constraint programming. In *Proc. of CPAIOR*, pages 74–90, 2015.

[Campigotto *et al.*, 2011] Paolo Campigotto, Andrea Passerini, and Roberto Battiti. *Active Learning of Combinatorial Features for Interactive Optimization*, pages 336–350. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Christofides *et al.*, 2013] Panagiotis D. Christofides, Riccardo Scattolini, David Muñoz de la Peña, and Jinfeng Liu. Distributed model predictive control: A tutorial review and future research directions. *Computers & Chemical Engineering*, 51:21–41, 2013.

[Donti *et al.*, 2017] Priya Donti, J. Zico Kolter, and Brandon Amos. Task-based end-to-end model learning in stochastic optimization. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5490–5500. Curran Associates, Inc., 2017.

[Dragone *et al.*, 2018] Paolo Dragone, Stefano Teso, and Andrea Passerini. Constructive preference elicitation. *Front. Robotics and AI*, 2018, 2018.

[Fischetti and Jo, 2018] Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. In *Proc. of CPAIOR*, 2018.

[Galassi *et al.*, 2018] Andrea Galassi, Michele Lombardi, Paola Mello, and Michela Milano. Model agnostic solution of csps via deep learning: a preliminary study. In *Proc. of CPAIOR*, 2018.

[Gilan and Dilkina, 2015] Siamak Safarzadegan Gilan and Bistra Dilkina. Sustainable building design: A challenge at the intersection of machine learning and design optimization. In *Proc. of AAAI Workshop on Computational Sustainability*, 2015.

[Hottung *et al.*, 2017] Andre Hottung, Shunji Tanaka, and Kevin Tierney. Deep learning assisted heuristic tree search for the container pre-marshalling problem. In *arXiv:1709.09972v1*, 2017.

[Huang *et al.*, 2017] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Proc. of CAV*, pages 3–29, 2017.

[Hutter *et al.*, 2011] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION*, pages 507–523, 2011.

[Hutter *et al.*, 2014] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[Katz *et al.*, 2017] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proc. of CAV*, pages 97–117, 2017.

[Laguna, 2011] Manuel Laguna. Optquest optimization of complex systems. In *OptQuest White Papers*, 2011.

[Lallouet and Legtchenko, 2007] Arnaud Lallouet and Andrei Legtchenko. Building consistencies for partially defined constraints with decision trees and neural networks. *International Journal on Artificial Intelligence Tools*, 16(4):683–706, 2007.

[Lallouet *et al.*, 2010] Arnaud Lallouet, Matthieu Lopez, Lionel Martin, and Christel Vrain. On learning constraint problems. In *Proc. of IJCAI*, pages 45–52, 2010.

[Liberto *et al.*, 2016] G. Liberto, S. Kadioglu, K. Leo, and Y. Malitshy. Dash: Dynamic approach for switching heuristics. *European Journal of Operational Research*, 2016.

[Lombardi and Gualandi, 2016] Michele Lombardi and Stefano Gualandi. A lagrangian propagator for artificial neural networks in constraint programming. *Constraints*, 21(4):435–462, 2016.

[Lombardi *et al.*, 2017] Michele Lombardi, Michela Milano, and Andrea Bartolini. Empirical decision model learning. *Artificial Intelligence*, 244(Supplement C):343 – 367, 2017. Combining Constraint Solving with Mining and Learning.

[Loth *et al.*, 2013] Manuel Loth, Michèle Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-based search for constraint programming. In *Proc. of CP*, pages 464–480, 2013.

[Mairy, 2011] Jean-Baptiste Mairy. Reinforced adaptive large neighborhood search. In *Proc. of CP*, 2011.

[Malitsky and Sellmann, 2012] Yuri Malitsky and Meinolf Sellmann. Instance-specific algorithm configuration as a method for non-model-based portfolio generation. In *Proc. of CPAIOR*, pages 244–259, 2012.

[Say *et al.*, 2017] Buser Say, Ga Wu, Yu Qing Zhou, and Scott Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *Proc. of IJCAI*, pages 750–756, 2017.

[Verwer *et al.*, 2017] Sicco Verwer, Yingqian Zhang, and Qing Chuan Ye. Auction optimization using regression trees and linear models as integer programs. *Artificial Intelligence*, 244:368–395, 2017.

[Vu *et al.*, 2017] Ky Khac Vu, Claudia D'Ambrosio, Youssef Hamadi, and Leo Liberti. Surrogate-based methods for black-box optimization. *ITOR*, 24(3):393–424, 2017.

[Wilson and Sahinidis, 2017] Zachary T. Wilson and Nikolaos V. Sahinidis. The ALAMO approach to machine learning. *Computers & Chemical Engineering*, 106:785–795, 2017.

[Xu *et al.*, 2008] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.