Towards Hybrid Constraint Solving with Reinforcement Learning and Constraint-Based Local Search

Helge Spieker and **Arnaud Gotlieb**
Simula Research Laboratory
Fornebu, Norway
{helge,arnaud}@simula.no

Abstract

This paper proposes a framework for solving constraint problems with neural networks trained with reinforcement learning (RL) and constraint-based local search (CBLS). We approach constraint solving as a declarative machine learning problem, where for a variable-length input sequence an output sequence has to be predicted. Using generated instances from the constraint model and feedback from the CBLS solver, a problem-specific RL agent is trained to solve the problem. The predicted solution candidate of the RL agent is verified and repaired by CBLS to ensure feasible solutions, that satisfy the constraint model. We introduce the framework and its components and discuss results and future applications of this method.

1 Learning and Constraint Solving

In this work, we present a constraint solving framework combining data-driven reinforcement learning (RL) and constraint solvers. The goal of this approach is to build problem-specific machine learning models, that are able to support solving problems as they occur in constraint programming (CP). Constraint solvers are highly optimized, still, searching for good or even optimal solutions often is a time-consuming task due to large search spaces that have to be traversed. So far, constraint solvers mostly ignore the fact, that in real-world applications, the same problem has to be solved repeatedly with different inputs. Solving a problem repeatedly provides data and insights into the problem structure and from earlier experiences, it is possible to learn an optimized behavior for future instances.

The advantage at hand for machine learning in this scenario are the shorter inference times compared to the solving times of constraint solvers. Still, a constraint solver can guarantee to find a solution, which satisfies all constraints, but for data-driven machine learning (ML), this is not possible with current methods. To overcome this challenge, the presented method combines the ML/RL with a constraint-based local search (CBLS) [Hentenryck and Michel, 2009] method which can verify and repair an initial solution candidate assignment, such that it satisfies a constraint model.

Whereas many machine learning models are trained in a supervised way by providing both inputs and expected outputs during training, it is challenging to generate a sufficiently large training corpus for constraint problems. Besides generating the problem instance, it also requires solving the problem to generate the optimal labels. This is a computationally expensive process and it also leads the machine learning model to imitate exactly the solutions found by the solver used to build the training dataset and does not generalize [Bello *et al.*, 2017]. Especially if an instance has multiple optimal solutions, using one specific solution limits the ability to generalize to an individual problem-solving strategy. In RL, it is not necessary to have solved all instances of the training set, because only scalar reward, formed by the number of constraint violations and the objective value of the solution candidate, is necessary as feedback, which is easier to compute than an optimal or high-quality solution. Learning from rewards allows to find individual solving strategies and to abstract from the direct input-output mapping of supervised learning. Linking a machine learning model and its training to the abstract constraint model allows encapsulation of the domain knowledge and does not require additional expertise in machine learning to use the framework.

In constraint modeling, a domain expert explicitly models a problem in terms of constraints, variables, their domains, and, in case of optimization, an objective function. A constraint model is thereby a distillation of domain and expert knowledge, necessary to find feasible solutions to all instances of the problem. There are several abstract constraint modeling languages that support this modeling step in a solver-independent manner, e.g. MiniZinc, Essence or XCSP. Such an abstract constraint model can be described as a function, that takes a set of instance parameters as an input, and, with the help of a constraint solver, solves the instance for which then the solution is returned as a set of output variables, i.e. the assigned variables. For example, in the model in Figure 2, the inputs are n and $limits$, and the output is x. The size of both $limits$ and x depend on n, which makes the in-/output to be of variable length. This formulation hides the notion of constraints, domains, and relations and is a simplified view of a constraint problem. However, when building a problem-specific ML model, the hidden components are constants and part of the function to be approximated, which allows easier representation of the problem in ML methods.

2 Related Work

The proposed integration, consisting of interacting CP and ML components, can be embedded in terms of the inductive constraint programming loop (ICP) [Bessiere *et al.*, 2017]. This recently proposed framework formulates the combination of a CP component, an ML component, and their interaction with each other and an external environment. Both the CP and ML component stimulate and receive feedback from each other. ICP is defined in a general way without being focused on one specific task but provides a general background for the close integration of CP and ML as independent components. In terms of ICP, our framework mainly relies on the links *ML-to-CP*, to verify and repair solution candidates, and *CP-to-ML*, to give feedback to the RL agent, but also all other links in the ICP model are relevant to some degree.

Another recent approach is empirical decision model learning [Lombardi *et al.*, 2017]. Here, a learning method is trained and then embedded into the constraint model. ML inference can thereby be used within the constraint solving process and be integral to the decisions made during search. Once an RL agent is sufficiently trained, empirical decision model learning can be a way to deploy a trained agent into the constraint model.

Solving combinatorial optimization problems with deep learning, and especially reinforcement learning, is an active research area. Recent works [Vinyals *et al.*, 2015; Bello *et al.*, 2017; Dai *et al.*, 2017; Selsam *et al.*, 2018] propose different approaches to learn how to solve combinatorial optimization problems and report successes. However, the problem sizes for which these approaches are effective are limited and not competitive to constraint solvers.

Other integrations of ML and CP have been evaluated in a large body of work. For an overview, and besides the given references, we refer to [Freuder, 2018].

3 Proposed Method

3.1 Framework and Process

The hybrid constraint solver consists of three main components: a) a constraint model, on which the hybrid solver is trained b) a feedback-enabled constraint solver c) an RL-based ML component. In Figure 1, the overall scheme and process of the hybrid solving method is shown.

At first, an instance is an input to the hybrid solver. We distinguish two kinds of instance sources, although they do not make a difference for the solver or the process. One type are generated problem instances, which are especially used for the initial training phase of the solver. The other type of instances are real-world instances, that is, those instances that stem from other sources and which are not only solved to generate a training feedback.

Within the solving component, the RL agent calculates a solution for the problem instance. Calculating the solution candidate is done via a neural network (NN), that receives an encoding of the problem instance as an input and predicts the solution candidate as an output. The network architecture is a separate module in the RL agent and further research is required to evaluate which is most effective.

The solution candidate can, and after a sufficient training period should be a feasible or close to a feasible solution for any given instance. However, due to the nature of both machine learning and constraint optimization, guaranteeing feasible solutions is a challenging task. The CBLS component receives the solution candidate, as well as the constraint model, as inputs with the task to both verify the feasibility of the solution candidate and, if necessary, repair it to be feasible. In case of an unsatisfiable instance, this is detected by the CBLS component, too. Also, it would be possible to give the solver additional time to find an improved solution.

3.2 Instance Generator

The constraint model is defined in an abstract constraint modeling language by an expert with domain knowledge. It is itself applicable to solve a problem instance together with a solver. The constraint model can be transformed into an instance generator by declaring the instance parameters as output variables, which are then included in the search. Additionally, in case of an optimization problem, the model is declared as a satisfaction problem. The resulting generator model is then solved with random variable and value selection to find random instances for the original problem. This method to transform a constraint model was presented in [Gent *et al.*, 2014] to generate discriminating instances for constraint model selection.

Nevertheless, transforming the optimization model to a generator model is one way to use the encoded domain knowledge and reuse for training the machine learning model.

3.3 Reinforcement Learning Agent

In RL, an agent interacts with its environment, such that it receives a state information as an input and selects an action as a result. Afterwards, it receives feedback from the environment, called a reward. From this reward, the agent adjusts its experience and behavior. Predicting a solution for a constraint problem has the challenge, that the action space, that is possible output variable assignment, has a huge size and it will not be possible to explore all possible actions. It is therefore necessary to introduce advanced RL methods.

We propose to train the ML model by an advantage actor-critic RL method (A2C) [Mnih *et al.*, 2016] for sequence-to-sequence problems [Nguyen *et al.*, 2017]. An A2C agent consists of two neural networks, an actor and a critic. The actor-network is responsible for action selection based on the input, whereas the critic-network estimates the expected reward for an action. That is, the critic-network serves as a surrogate model for the reward and approximates the objective function of the constraint model, which stabilizes learning.

There are alternative ways to represent a problem instance, the network input, and the solution candidate, the output. As constraint problems often handle integer values, it is both feasible to directly input these values into the neural network. An alternative would be to use an embedding, as it is done for text processing, which represents each possible integer value with a vector of float values. Using embeddings is a dense representation and allows to learn relations between values, which are not directly given by their ordinal integer relation.

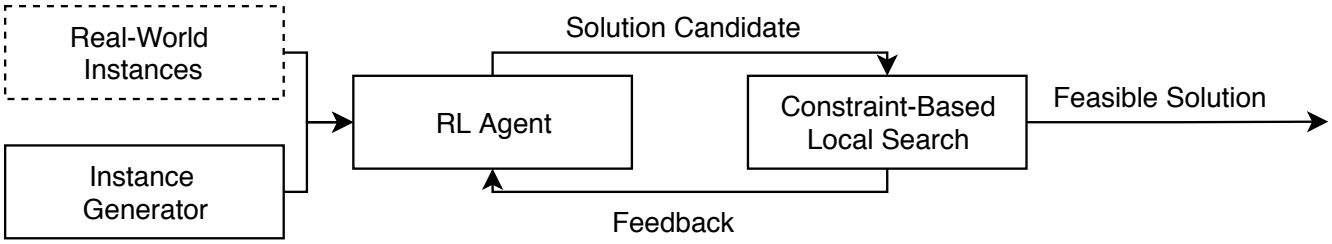

Figure 1: Process overview for hybrid constraint solving loop. Constraint-based local search both ensures a feasible solution and feedback for the learning component. During training, a generator provides instances.

For these reasons, we use an embedding of the possible integer values as inputs and outputs. The agent returns all output variables of the constraint model. Only the objective variable is excluded, because it can be calculated from the other variables and the often large domain of the objective variable unnecessarily increases the embedding size and thereby the model complexity.

3.4 Solver Feedback and Rewards

The reward for a solution candidate is formed by the number of violated constraints in the solution candidate and its objective value. If the solution candidate is a feasible solution and satisfies all constraints, the reward is equal to the objective value of the solution (in case of maximization):

$$r = \begin{cases} -\text{violations} & \text{if } violations > 0 \\ \text{objective} & \text{else} \end{cases} \quad (1)$$

This reward function puts emphasis on the feasibility of the solution candidate by giving low rewards for unfeasible candidates without considering the objective.

An alternative formulation of a reward function is used in CBLS methods, where violations are acceptable during search. Here, the number of violations and the objective of a solution are combined to quantify the value of the current solution: $r = \alpha * \text{violations} + \beta * \text{objective}$ with α and β being (adaptive) configuration parameters [Björdal *et al.*, 2015]. This reward function emphasizes finding solution candidates which are close too high-quality solutions but are not necessarily feasible. Due to the additional complexity introduced by configuration α and β, we do not consider this second reward function further for the experiments in this paper.

4 Experiments

4.1 Constraint Optimization Problem

The optimization problem, *maximum alldifferent with limits* (MAWL), is a maximization problem with the goal to find the maximum sum of a list under given limits and with an alldifferent global constraint which enforces all list values to be unique. The length of the list n and the maximum values per position, $limits$, are given as instance parameters, i.e. input variables. The output is x, the list of assigned values. A MiniZinc [Nethercote *et al.*, 2007] formulation of MAWL is shown in Figure 2.

The training corpus consists of 50,000 instances and their optimal solution for an initial supervised pre-training phase

```
include "alldifferent.mzn";
include "increasing.mzn";
int: n; % Input
array[1..n] of 0..1000: limits; % Input
array[1..n] of var int: x; % Output
constraint alldifferent(x);
constraint increasing(x);
constraint forall(i in 1..n)(x[i] < limits[i]);
var int: objective = sum(x);
solve maximize objective;
```

Figure 2: Maximum alldifferent with limits (MAWL)

and 100,000 instances for the main training phase. The validation test set contains 1000 instances. All instances have been sampled from a transformed model (as described in Section 3.2) with $n \in [2, 250]$ and duplicates have been removed.

4.2 Training Settings

The actor and the critic are modeled as encoder-decoder networks with global attention [Luong *et al.*, 2015] as described in [Nguyen *et al.*, 2017]. In an encoder-decoder network, the encoder transforms a variable-length input sequence into a fixed size hidden representation. The decoder then transforms this representation into the output sequence. An attention mechanism allows selectively highlighting parts of the input sequence during encoding and decoding. Both the encoder and decoder are single-layer LSTMs with an embedding and hidden size of 500 cells. The dictionary size is derived from the constraint model and contains 1005 elements, consisting of the numbers $[0, 1000]$ and four helper elements to mark beginning and end of a sequence, unknown elements, and for padding.

Rewards are calculated by a modified version of Choco 4 [Prud'homme *et al.*, 2017], a constraint solver with local search functionality.

4.3 Results and Analysis

Figure 3 shows the number of feasible solutions for the validation set and the total rewards received. After 10 epochs of supervised pre-training, the training method is switched to reinforcement learning, which improves the performance on the validation test set. Whereas the supervised training predicts 32% feasible solution candidates, after one epoch of RL, this value increases to 88% and after some epochs to 92%. Both the number of feasible solutions and the total reward converges after 20 epochs.

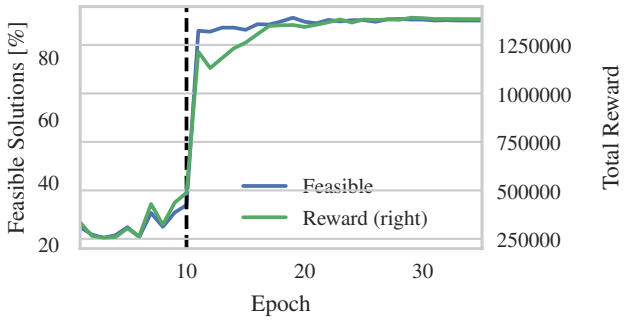

Figure 3: Feasible solutions for validation instances. Supervised pre-training until epoch 10, reinforcement learning afterwards.

To evaluate the influence of both pre-training and RL, we train once with each technique only. Running the supervised training for 20 epochs increases the performance to 33% feasible solutions and stagnates afterward. Running only RL, the agent initially does not produce valid outputs to form solution candidates and has to learn the structure of a solution first. Here, training did not produce an usable agent.

It is not fair to compare the outcomes of both trainings, due to the different training corpus sizes. However, comparing the results to the combination of pre-training and RL shows a clear difference in performance and the advantage of RL beyond training set size.

We also evaluate the minimum reward on the validation set to gain insights on the worst performance of the RL agent. At the end of the pre-training phase, the minimal reward is -88, i.e. the worst solution candidate has 88 violations. After convergence, the worst solution candidate has 10 violations, i.e. a reward of -10.

The results on MAWL show the ability to learn solution candidate prediction and underline the motivation to use reinforcement learning for training. Besides the reduced effort in training data generation, without having to solve the training instances to an optimal solution, giving a scalar reward instead of a feasible solution shows better generalization.

5 Conclusion

In this work, we present a hybrid framework for constraint solving, consisting of a machine learning model to predict a solution and a CBLS solver to verify or repair this solution. The machine learning model is trained via reinforcement learning using feedback from the CBLS solver. Combining RL and CBLS allows to exploit historical information and learn from the constraint model. Providing an RL agent shifts computational load from solving time to training time.

The two main components of the framework, the RL agent and the CBLS solver, are modular. One open task for further research is to evaluate the different network architectures and ways to represent problem instances and solutions as inputs and outputs to the network.

A preliminary evaluation shows general applicability on a constraint optimization problem. The agent learns to produce solution candidates, which are feasible to a large degree and receives high rewards.

References

[Bello *et al.*, 2017] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization. *ICLR*, 2017.

[Bessiere *et al.*, 2017] Christian Bessiere, Luc De Raedt, Tias Guns, Lars Kotthoff, Mirco Nanni, Siegfried Nijssen, Barry OSullivan, Anastasia Paparrizou, Dino Pedreschi, and Helmut Simonis. The Inductive Constraint Programming Loop. *IEEE Intelligent Systems*, 2017.

[Björdal *et al.*, 2015] Gustav Björdal, Jean-Noël Monette, Pierre Flener, and Justin Pearson. A constraint-based local search backend for MiniZinc. *Constraints*, 2015.

[Dai *et al.*, 2017] Hanjun Dai, Elias Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning Combinatorial Optimization Algorithms over Graphs. In *NIPS*, 2017.

[Freuder, 2018] Eugene C Freuder. Progress towards the Holy Grail. *Constraints*, 2018.

[Gent *et al.*, 2014] Ian P Gent, Bilal Syed Hussain, Christopher Jefferson, Lars Kotthoff, Ian Miguel, Glenna F Nightingale, and Peter Nightingale. Discriminating instance generation for automated constraint model selection. In *CP*, 2014.

[Hentenryck and Michel, 2009] Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. The MIT Press, 2009.

[Lombardi *et al.*, 2017] Michele Lombardi, Michela Milano, and Andrea Bartolini. Empirical decision model learning. *Artificial Intelligence*, 244:343–367, 2017.

[Luong *et al.*, 2015] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. In *EMNLP*, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *ICML*, 2016.

[Nethercote *et al.*, 2007] Nicholas Nethercote, Peter Stuckey, Ralph Becket, Sebastian Brand, Gregory Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *CP*, 2007.

[Nguyen *et al.*, 2017] Khanh Nguyen, Hal Daumé III, and Jordan Boyd-Graber. Reinforcement Learning for Bandit Neural Machine Translation with Simulated Human Feedback. In *EMNLP*, 2017.

[Prud'homme *et al.*, 2017] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. Choco Documentation, 2017.

[Selsam *et al.*, 2018] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT Solver from Single-Bit Supervision. *arXiv:1802.03685*, 2018.

[Vinyals *et al.*, 2015] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In *NIPS*, 2015.