CP for ILP

AAAI 2024 Bridge on Constraint Programming and Machine Learning
Pitch or Discussion Track

Andrew Cropper and Céline Hocquette
University of Oxford
andrew.cropper@cs.ox.ac.uk, celine.hocquette@cs.ox.ac.uk

1 Inductive logic programming

Inductive logic programming (ILP) [11, 4] is a form of supervised machine learning (ML). As with other forms of ML, the goal is to induce a hypothesis that generalises training examples. However, whereas most forms of ML use tables to represent data, ILP uses logic programs (sets of logical rules). Moreover, whereas most forms of ML learn functions, ILP learns relations.

ILP has several attractive features [5]. For instance, it is data efficient and can induce hypotheses from small numbers of examples. Moreover, because of logic's similarity to natural language, logic programs can be easily read by humans, which is crucial for explainable AI.

2 CP for ILP

Classical ILP approaches search by revising a hypothesis through the addition and removal of literals [13, 12, 1, 14]. These approaches use heuristics, such as information gain, to guide the search through the hypothesis space. An emerging trend in ILP is to formulate the ILP problem as a *constraint problem* (CP). For instance, ASPAL [2] encodes an ILP problem as an answer set programming (ASP) problem. Specifically, ASPAL constructs an ASP program that describes every training example and every possible rule in the hypothesis space. ASPAL then asks an ASP solver to find a subset of the rules that covers all the positive but none of the negative examples.

The main advantages of these recent ILP-as-CP approaches is that they can more easily learn recursive programs and use state-of-the-art CP solvers. However, most approaches encode the ILP problem as a single one-shot CP instance. These approaches, therefore, struggle to scale to non-trivial problems. For instance, because ASPAL precomputes every possible rule it struggles to learn rules with more than a few literals.

3 Popper

Popper [8, 3, 6, 7] encodes an ILP problem as a CP problem, where each solution to the CP problem represents a hypothesis. However, rather than precompute every possible rule,

the key idea of Popper is to incrementally discover constraints from smaller hypotheses to rule out larger hypotheses. Specifically, Popper uses a *generate*, *test*, and *constrain* loop to generate hypotheses and test them on the examples. In the generate stage, Popper asks a CP solver to find a solution to the CP instance, where the solution corresponds to a hypothesis. In the test stage, Popper tests the hypothesis on the training examples. If a hypothesis is too general (entails too many negative examples), Popper builds constraints to prune logically more general programs from the hypothesis space, as they too will be too general. Likewise, if a hypothesis is too specific (entails too few positive examples), Popper builds constraints to prune logically more specific hypotheses, as they too will be too specific. Popper adds these constraints to the CP problem to prune solutions and thus prune hypotheses. Popper repeats this loop until it finds a good hypothesis or there are no more hypotheses.

4 Popper at CP-ML bridge

Popper combines various forms of CP for ML, including ASP, SMT, SAT, and MaxSAT. For instance, Popper uses ASP to generate hypotheses. It uses MaxSAT solvers to find combinations of hypotheses with minimal description length [10]. It also uses SMT solvers to search for hypotheses with numerical values [9]. We, therefore, think that Popper would interest a broad audience at the CP-ML bridge program.

We want to be at the bridge because we want to collaborate with CP researchers. In particular, we want to improve the scalability and efficiency of Popper by using CP tools better. For instance, we encode our *generate stage* as an ASP problem, which is often the bottleneck of Popper. However, we only use ASP because we have a logic programming background knowledge. We do not need answer set semantics so it is plausible that another CP paradigm is more suitable for this task. Moreover, there might be alternative ways to encode our CP problems.

5 Pitch/discussion presentation

We propose to give a brief (<20 minutes) introduction to ILP and Popper. The emphasis of the presentation will be on the use of CP for ILP and, in particular, the current limitations of Popper due to our current CP approach. The goal is to foster collaboration between CP and ILP researchers. We expect for both authors of this proposal to attend the bridge program.

References

[1] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, (1-2):285–297, 1998.

[2] D. Corapi, A. Russo, and E. Lupu. Inductive logic programming in answer set programming. In *Inductive Logic Programming - 21st International Conference*, pages 91–97, 2011.

[3] A. Cropper. Learning logic programs though divide, constrain, and conquer. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*, pages 6446–6453. AAAI Press, 2022.

[4] A. Cropper and S. Dumancic. Inductive logic programming at 30: A new introduction. *J. Artif. Intell. Res.*, 74:765–850, 2022.

[5] A. Cropper, S. Dumancic, R. Evans, and S. H. Muggleton. Inductive logic programming at 30. *Mach. Learn.*, 111(1):147–172, 2022.

[6] A. Cropper and C. Hocquette. Learning logic programs by combining programs. In *ECAI 2023 - 26th European Conference on Artificial Intelligence*, volume 372, pages 501–508. IOS Press, 2023.

[7] A. Cropper and C. Hocquette. Learning logic programs by discovering where not to search. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI*, pages 6289–6296. AAAI Press, 2023.

[8] A. Cropper and R. Morel. Learning programs by learning from failures. *Mach. Learn.*, (4):801–856, 2021.

[9] C. Hocquette and A. Cropper. Relational program synthesis with numerical reasoning. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, pages 6425–6433. AAAI Press, 2023.

[10] C. Hocquette, A. Niskanen, M. Järvisalo, and A. Cropper. Learning mdl logic programs from noisy data, 2023.

[11] S. Muggleton. Inductive logic programming. *New Generation Computing*, (4):295–318, 1991.

[12] S. Muggleton. Inverse entailment and progol. *New Generation Comput.*, (3&4):245–286, 1995.

[13] J. R. Quinlan. Learning logical definitions from relations. *Mach. Learn.*, pages 239–266, 1990.

[14] A. Srinivasan. The ALEPH manual. *Machine Learning at the Computing Laboratory, Oxford University*, 2001.