The Inductive Constraint Programming Loop

Christian Bessiere, Luc de Raedt, Tias Guns, Lars Kotthoff, Mirco Nanni, Siegfried Nijssen, Barry O'Sullivan, Anastasia Paparrizou, Dino Pedreschi, Helmut Simonis

Extended Abstract: The Inductive Constraint Programming Loop

Christian Bessiere[1], Luc De Raedt[2], Tias Guns[2], Lars Kotthoff[3], Mirco Nanni[4], Siegfried Nijssen[2,5], Barry O'Sullivan[3], Anastasia Paparrizou[1], Dino Pedreschi[4], and Helmut Simonis[3]

[1]CNRS, University of Montpellier, France
[2]DTAI, KU Leuven, Belgium
[3]Insight, University College Cork, Ireland
[4]University of Pisa, Italy
[5]Universiteit Leiden, The Netherlands

Abstract

Constraint programming is used for a variety of real-world optimization problems, such as planning, scheduling and resource allocation problems. At the same time, one continuously gathers vast amounts of data about these problems. Current constraint programming software does not exploit such data to update schedules, resources and plans. We propose a new framework, that we call the *Inductive Constraint Programming (ICON) loop*. In this approach data is gathered and analyzed systematically in order to dynamically revise and adapt constraints and optimization criteria. Inductive Constraint Programming aims at bridging the gap between the areas of data mining and machine learning on the one hand, and constraint programming on the other hand.

This chapter is an extended abstract of

> Christian Bessiere, Luc De Raedt, Tias Guns, Lars Kotthoff, Mirco Nanni, Siegfried Nijssen, Barry O'Sullivan, Anastasia Paparrizou, Dino Pedreschi, Helmut Simonis. *The Inductive Constraint Programming Loop*. CoRR, abs/1510.03317, 2015.
> `http://arxiv.org/abs/1510.03317`

1 Introduction

Machine Learning/Data Mining (ML/DM) and Constraint Programming (CP) are central to many application problems. ML is concerned with learning functions/patterns characterizing some training data whereas CP is concerned with

finding solutions to problems subject to constraints and possibly an optimization function.

The problem with current technology is that the problems of data analysis and constraint satisfaction/optimization have almost always been studied independently and in isolation. Indeed, there exist a wide variety of successful approaches to analysing data in the field of ML, DM and statistics, and at the same time, advanced techniques for addressing constraint satisfaction and optimization problems have been developed in the CP community. Over the past decade a limited number of isolated studies on specific cases has indicated that significant benefits can be obtained by connecting these two fields [EF01, XHHL08, DGN08, BHO09, KBC10, CJSS12], but so far a truly general, integrated and cross-disciplinary approach is missing.

CP technology is used to solve many types of constraint satisfaction and optimization problems, such as in power companies generating and distributing electricity, in hospitals planning their surgeries, and in public transportation companies scheduling buses. Despite the availability of effective and scalable solvers, current approaches are still unsatisfactory. The reason is that when using CP technology to solve these applications, the constraints and criteria, that is, the *model*, must be specified statically. However, in reality often this model needs to be revised over time. The revision can be needed to reflect changes in the environment due to external events that impact the problem. The revision can also be needed because the execution of the solution generated by the model has modified the characteristics of the problem. Finally the revision can be needed simply because the original model did not capture correctly the problem. Observing the impact of the solution allows us to correct or improve the model. Therefore, there is an urgent need for improving and revising a model over time *based on data* that is continuously gathered about the performance of the solutions and the environment they are used in. The CP community has extended the basic constraint satisfaction and optimization problems to better tackle changing environments. The *dynamic* constraint satisfaction approach ([DD88]) allows the addition/retraction of constraints from the initial model. This approach does not predict the changes from data, but rather the addition/retraction of constraints is performed by the user. The *online/stochastic* constraint programming approach ([BH04, Wal02]) offers a framework to deal with unknown future events, such as customer requests. It builds a finite set of *future scenarios*, e.g. using sampling from a known distribution, and the optimization problem is then defined over each of the scenarios. The framework does not capture ways of using data, other than for the prediction of possible scenarios of events.

In general, exploiting gathered data to modify and adjust any aspect of a model is difficult and labor intensive with state-of-the-art solvers. As a consequence, the data that is being gathered today in order to monitor the quality of the produced solutions and to help evaluating the effect of possible adjustments to the constraints or optimization criteria, is not fully exploited when changes in a schedule or plan are needed. Hence, schedules and plans that are produced are often suboptimal. This, in turn, leads to a waste of resources. Instead of using

data passively, data should be actively analysed in order to discover and update the underlying regularities, constraints and criteria that govern the data.

In this chapter, we propose and formalize the new framework of inductive constraint programming. This framework is based on what we call the *Inductive Constraint Programming (ICON) loop*, which is an interaction between a machine learning component (ML) and a constraint programming component (CP). The ML component observes the world and extracts patterns. The CP component solves a constraint satisfaction or optimization problem using these patterns; its solution is applied to the world. We assume the world changes over time, possibly due to the impact of applying our solution. This process is repeated in a loop. Inductive constraint programming will serve the long-term vision of easier-to-use and more effective tools for resource optimization and task scheduling.

An introduction to Constraint Programming and Data Mining was already given earlier in this book; the focus of this chapter is on introducing the formalism behind the loop. Extensive examples of the loop can subsequently be found in the subsequent chapters.

2 Inductive Constraint Programming Loop

The inductive constraint programming loop will cope with changes in the world by iteratively solving a learning problem and a constraint problem. The loop is composed of several components that interact with each other through writing and reading operations. A visualization of the loop is given in Figure 1. We introduce each of the elements in the loop in turn.

CP Component. An important element of the CP component is the constraint network. A *constraint network* $N = (X, D, C, f)$ is composed of: a set X of variables taking values in domain D. These variables are subject to constraints in the set C. The optional evaluation function f takes as input an assignment on X and returns a cost for it. A solution (optionally *best* solution) of N is an assignment in D^X satisfying all the constraints in C (optionally minimizing f).

A *solver* Xsolve takes as input a constraint network and returns a solution/best solution or failure in case no solution satisfying all the constraints exists.

The CP component is composed of the constraint network $N = (X, D, C, f)$, the constraint solver Xsolve, and a Solutions repository. Xsolve generates solutions of N, or good/best solutions of N according to f, that it writes in the Solutions repository. In case Xsolve is not able to produce any solution to be applied to the world, the CP component notifies the ML component by sending information about the failure.

More details about CP can be found in the first chapter of this book.

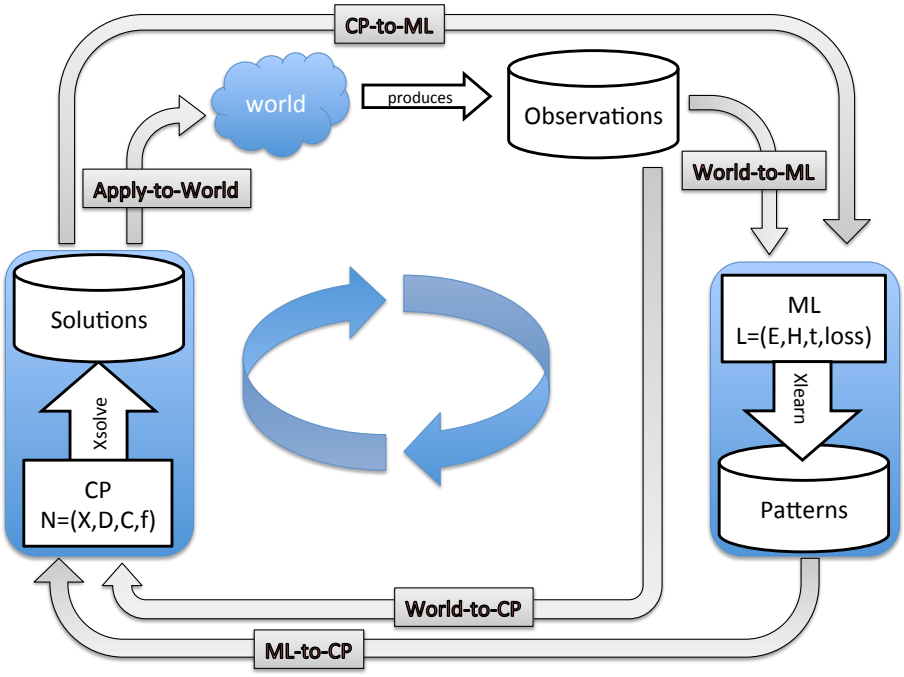

Figure 1: The Inductive Constraint Programming loop

ML Component. A *learning problem* $L = (E, H, t, loss)$ is composed of a set E of examples, a hypothesis space H, the target function t that one wants to learn, and a loss function $loss(E, h, t)$ that measures the quality of a hypothesis $h \in H$ w.r.t. dataset E and the target hypothesis t. The goal is to find a hypothesis that minimizes the loss.

For example, given real-valued data $E \subset \mathbb{R}^d$ and real-valued labels identified by target function t, where $\forall \mathbf{e} \in E : t(\mathbf{e}) \in \mathbb{R}$, the goal of linear regression is to learn a linear function $h_\mathbf{c} : E \to \mathbb{R}$ with coefficients $\mathbf{c}$ that minimizes the sum of squared errors between the predicted value and the observed value: $loss(E, h_\mathbf{c}, t) = \sum_{\mathbf{e} \in E} |h_\mathbf{c}(\mathbf{e}) - t(\mathbf{e})|^2 = \sum_{\mathbf{e} \in E} |\mathbf{e} \cdot \mathbf{c} - t(\mathbf{e})|^2$. Many other loss functions and hypothesis spaces have been defined in the literature.

The ML component is composed of the learning problem $L = (E, H, t, loss)$, the learner XLearn, and a Patterns repository. XLearn learns hypotheses t (typically one) and writes them in the Patterns repository.

More information about data mining and machine learning can be found in the second chapter of this book.

World. The World component is composed of a world W, an evaluation function *eval_world*, and a Observations repository. The world W can have its own

independent behavior, dynamically changing under the effect of time and the effect of applying solutions of the Solutions repository. The solutions are evaluated by the *eval_world* function and this feedback is stored in the Observations repository.

Now that we have defined the basis of the inductive constraint programming loop, we need to define the way the CP component, the ML component and the world interact with each other. They interact through a set of reading/writing functions.

An *inductive constraint programming loop* is composed of a world $(W, eval\_world)$, a CP component (N, Xsolve), and an ML component (L, XLearn). The loop uses the following channels of communication:

- function World-to-ML reads data and evaluations from the Observations repository and updates the learning problem L, that will be used by XLearn to learn a hypothesis h;
- function CP-to-ML is used to send feedback from the previous iteration of the CP component to the ML component, e.g. when Xsolve cannot find any satisfactory solution to be applied to the world;
- function World-to-CP reads data from the Observations repository that can be used to directly update the constraint network N used by Xsolve;
- function ML-to-CP reads patterns from the Patterns repository and updates the constraint network N used by Xsolve to produce solutions;
- function Apply-to-World takes solutions in the Solutions repository and applies them to the world, if possible.

The following pseudo code demonstrates how these communication channels are used in the inductive constraint programming loop:

Algorithm 1 Pseudo code of a loop cycle using the components.

```
function CYCLE(Observations, optional Solutions)
    repeat
        L_o ← World-to-ML(Observations)
        L_p ← CP-to-ML(Solutions)
        L ← constructL(L_o, L_p)
        Patterns ← applyXlearn(L)
        N_o ← World-to-CP(Observations)
        N_p ← ML-to-CP(Patterns)
        N ← constructN(N_o, N_p)
        Solutions ← applyXsolve(N)
    until Apply-to-World(Solutions)
end function
```

Initially, World-to-ML is used to gather training data to the ML component. These data can be feedback from previous executions of solutions of the CP

component on the world. The solution of the previous cycle can also directly be used as well, through CP-to-ML. This is especially useful if the previous solution could not be applied to the world, for example because the learned patterns lead to an inconsistency. Using the output of World-to-ML and CP-to-ML, the learning problem L can then be constructed, specific to the learner at hand. Next, the learner is applied to L and patterns are obtained. These patterns can be weights of an objective function, constraints, or any other type of structural information that is part of the CP problem.

A similar process then happens for the CP component, the network is constructed using the output of World-to-CP and ML-to-CP, after which the solving method is used and solutions are obtained.

These solutions are then applied to the world using Apply-to-World. As mentioned before, it may be that the found solution (or non-solution) is not applicable to the world. In that case, a new iteration of the loop is started immediately which bypasses the world. Otherwise the solutions are applied to the world, after which a new cycle with new observations can be started.

We can observe that there is no direct link between the ML component and the world. Our framework is indeed devoted to solving combinatorial problems such as scheduling and routing, revising them based on feedback from the world; it does not aim to only classify or predict events in the world.

3 Illustrative Example

To illustrate the inductive constraint programming loop we will use a scheduling setting that occurs in hospitals. This setting includes an ML component, a CP component and a world component.

We will first describe the CP component. In this component we focus on a task scheduling problem. The treatment of a patient typically involves the execution of various tasks on this patient, such as executing scans, taking blood tests, operating on the patient, physiotherapeutic sessions, and so on. These tasks need to be executed in a well-defined order, and require the use of the resources of the hospital for a certain amount of time. The overall scheduling problem is how to schedule these tasks in the shortest amount of time possible, using the limited resources of the hospital.

Important parameters of this scheduling problem hence include the resources available in the hospital and the tasks that need to be executed. For each task, it is important which resources need to be used, how many such resources are needed, and for how long they need to be used.

Whereas for many patients it is clear which procedures need to be followed before the patient can be discharged from the hospital, this is not the case for the duration of these tasks: depending on parameters such as age or health conditions, a certain task may take much longer for one patient than for another patient.

The goal of the ML component is to address this challenge: its role is to predict how long a task is estimated to take for a patient. This involves solving

a regression problem as identified earlier: for each given task for a patient, the properties of the task and the patient, together with similar historic data and the resulting durations, are used to predict the task duration, which is a real number.

The world component executes the schedules; it produces data about patients and observations concerning the true durations of tasks.

Clearly, as the tasks are executed in the hospital, the predicted durations may differ from the actual durations. Furthermore, new patients and hence new tasks arrive. This means that the hospital needs to schedule tasks on a regular basis. The patient data that is collected during each such iteration can here be used to improve the quality of the predicted task durations. This makes it a good example of the inductive constraint programming loop. Within this loop, we can distinguish the following components and functions:

- function World-to-ML reads historical patient data and historical task durations for these patients; furthermore, it reads the patients that are currently in the hospital and the tasks that need to be executed for these patients;
- the ML component predicts the durations for the tasks that need to be executed, using the historical data;
- function ML-to-CP reads the learned durations and updates the CP network accordingly;
- function World-to-CP reads the tasks that need to be executed from the world, as well as the resources available in the hospital;
- the CP component solves the updated scheduling problem;
- function Apply-to-World applies the resulting schedule in the world.

In this example, the function CP-to-ML is not used; it could be used, for instance, if there is a preference to schedule nurses and doctors in similar teams or with similar load or time-breaks from day-to-day.

Both components can be formalized using a CP language, such as the MiniZinc language mentioned earlier.

The scheduling model and the machine learning model together define both components of the inductive constraint programming loop. We here demonstrated how a declarative, unified language could be used to model both the learning problem and the solving problem. While a single language for both the learning and solving is an appealing prospect, it is not a requirement for the applicability of the inductive constraint programming loop.

4 Conclusion

The key idea in the inductive constraint programming (ICON) loop is that the CP and ML components interact with each other and with the world in order to

adapt the solutions to changes in the world. This is an essential need in problems that change under the effect of time, or problems that are influenced by the application of a previous solution. It is also very effective for problems that are only partially specified and where the ML component learns from observation of applying a partial solution, e.g. in the case of constraint acquisition.

The subsequent chapters will provide a number of examples of the use of the ICON loop.

References

[BBP15] A. Balafrej, C. Bessiere, and A. Paparrizou. Multi-armed bandits for adaptive constraint propagation. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 290–296, Buenos Aires, Argentina, 2015. AAAI Press.

[BCOP07] C. Bessiere, R. Coletta, B O'Sullivan, and M. Paulin. Query-driven constraint acquisition. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 44–49, Hyderabad, India, 2007.

[BH04] R. Bent and P. Van Hentenryck. Online stochastic and robust optimization. In *Proceedings of the 9th Asian Computing Science Conference (ASIAN 04)*, pages 286–300, Chiang Mai, Thailand, 2004. Springer.

[BHO09] C. Bessiere, E. Hebrard, and B. O'Sullivan. Minimising decision tree size as combinatorial optimisation. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP 2009)*, pages 173–187, Lisbon, Portugal, 2009. Springer.

[CJSS12] E. Coquery, S. Jabbour, L. Saïs, and Y. Salhi. A sat-based approach for discovering frequent, closed and maximal patterns in a sequence. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 258–263, Montpellier, France, 2012. IOS Press.

[DD88] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 37–42, St. Paul, MN, 1988. AAAI Press / The MIT Press.

[DGN08] L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 204–212, Las Vegas, Nevada, 2008. ACM.

[EF01] S.L. Epstein and E.C. Freuder. Collaborative learning for constraint solving. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP 2001)*, pages 46–60, Paphos, Cyprus, 2001. Springer.

[HKMO14] B. Hurley, L. Kotthoff, Y. Malitsky, and B. O'Sullivan. Proteus: A hierarchical portfolio of solvers and transformations. In *Proceedings of the 11th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2014)*, pages 301–317, Cork, Ireland, 2014. Springer.

[KBC10] M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (CP 2010)*, pages 552–567, St. Andrews, Scotland, 2010. Springer.

[MS14] K. Marriott and P.J. Stuckey. A minizinc tutorial. http://www.minizinc.org/downloads/doc-latest/minizinc-tute.pdf, 2014.

[TR13] T. Tulabandhula and C. Rudin. Machine learning with operational costs. *Journal of Machine Learning Research*, 14:1989–2028, 2013.

[Wal02] T. Walsh. Stochastic constraint programming. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence (ECAI'2002)*, pages 111–115, Lyon, France, 2002. IOS Press.

[XHHL08] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res. (JAIR)*, 32:565–606, 2008.