This article has been accepted for publication in IEEE Intelligent Systems but has not yet been fully edited.
Some content may change prior to final publication.

The Inductive Constraint Programming Loop

Christian Bessiere[1], Luc De Raedt[2], Tias Guns[2], Lars Kotthoff[3], Mirco Nanni[4], Siegfried Nijssen[2], 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

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, which we call the *inductive constraint programming 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.

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

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

[EF01, XHHL08, DGN08, BHO09, KBC10, CJSS12], but so far a truly general, integrated and cross-disciplinary approach has been 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 statically specified. However, in reality this model often 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. But 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 [Wal02, BH04] 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 constrained-based planning, the *conditional temporal* problem approach [TVP03] extends standard temporal constraint satisfaction by adding observation nodes and attaching labels to all nodes to indicate the situation(s) in which each will be executed. This extension permits the construction of conditional plans that are guaranteed to satisfy complex temporal constraints. This makes it possible to dynamically adapt the plan in response to the observations made during execution. However, it does not allow learning from experience from data, such as unsuccessful plans. Even further from CP technology, a *truth maintenance system* [Doy79] is a knowledge representation approach to recording and maintaining the reasons for program beliefs. The name truth maintenance is due to the ability of these systems to maintain consistency between old and current beliefs through a revision mechanism. In order to choose their actions, reasoning programs make assumptions and subsequently revise their beliefs when discoveries contradict these assumptions. Truth maintenance systems do not contain any constraint optimization or learning capabilities.

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 conse-

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

quence, the data that is being gathered today, in order to monitor the quality of the produced solutions and to help evaluate 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 paper, we propose and formalize the new framework of inductive constraint programming. This framework is based on what we call the *inductive constraint programming 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 whose 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.

2 Background

In this section we introduce the basic concepts used later in the paper. We briefly define and explain what is a constraint problem and a learning problem.

2.1 Constraint problem

The central notion in constraint programming is the *constraint*. A constraint is a Boolean function whose scope is a set of (integer) variables. Depending on whether the function returns true or false for a given input assignment of its variables, the constraint accepts or rejects the assignment. For instance, the constraint $X_1 + X_2 = X_3$ specifies that any combination of values for variables X_1, X_2 and X_3 has to be such that the sum of X_1 and X_2 equals X_3. Based on the notion of constraint, we define the concept of a constraint network and a solver.

A *constraint network* $N = (X, D, C, f)$ is composed of: a set X of variables taking values from 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* takes as input a constraint network and returns a solution/best solution or failure in case no solution satisfying all the constraints exists. If we take as an example the well-known Sudoku problem, a constraint network expressing it could be the following. The variables are the cells, namely $X = \{X_1 \ldots X_{81}\}$. Each X_i represents the digit in this cell, and as such, it takes a value from $D_i = \{1..9\}$. For every prefilled cell i, X_i is assigned the corresponding prefilled digit. For all the 810 pairs of cells i and j that belong to the same row, column

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

This article has been accepted for publication in IEEE Intelligent Systems but has not yet been fully edited.
Some content may change prior to final publication.

```
1  array[1..9,1..9] of 0..9: start; %% initial board 0 = empty
2  array[1..9,1..9] of var 1..9: puzzle;

3  % fill initial board
4  constraint forall(i,j in 1..9 where start[i,j] > 0)(
5    puzzle[i,j] = start[i,j] );

6  % All different in rows
7  constraint forall (i in 1..9) (
8    alldifferent( [ puzzle[i,j] | j in 1..9 ]) );

9  % All different in columns.
10 constraint forall (j in 1..9) (
11   alldifferent( [ puzzle[i,j] | i in 1..9 ]) );

12 % All different in sub-squares:
13 constraint forall (i, j in 1..3)(
14   let { int: a = (i-1)*3; int: b = (j-1)*3} in
15   alldifferent( [ puzzle[a+i1, b+j1] | i1,j1 in 1..3 ] ) );

16 solve satisfy;
```

Figure 1: Sudoku in pseudo-Minizinc.

or block, a constraint $X_i \neq X_j$ is put in C. Alternatively, C can be composed of 27 global constraints `alldifferent`(S), for the 27 sets S of 9 variables in the same row, column, or block.

There exist several languages/formats for specifying a constraint problem to be given to a solver for solving. Figure 1 expresses Sudoku as a constraint program using a pseudo-MiniZinc language [MS14]. Line `1` defines an input matrix `start` containing the prefilled cells of the Sudoku. Line `2` defines the matrix `puzzle` of variables that will contain the solution of the Sudoku. Lines `4-5` put equality constraints between the prefilled cells in the input matrix `start` and the matrix of variables `puzzle`. Lines `7-8` post an `alldifferent` constraint on every row of `puzzle`. `alldifferent(xi | i in 1..n)` is a global constraint that specifies that variables `x1..xn` must all take different values. Lines `10-11` do the same for the columns. The constraint specified in Lines `13-15` is a bit more tricky as it has to play with the indices of the subsquares to post the `alldifferent` constraints on the variables of every subsquare in `puzzle`. Finally, line `16` calls the solver on the instance.

2.2 Learning problem

In machine learning, the goal is to learn a hypothesis that explains the observed data, and thus is able to predict future data. The data typically consists of a set of training examples E, which are assumed to be independent and identically

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

distributed. Different learning methods differ largely in the type of examples to learn from, and the type of hypothesis they want to learn. The most popular learning setting is supervised learning, where each example in E is accompanied by a label that should be predicted.

Considering the sudoku example again, one might wish to learn how long it takes to solve a sudoku for a typical user, based on features of the sudoku such as the number of empty cells, how many cells contain the same number, the average number of choices left in the cells, etc. The training examples would then consist of these features for a certain sudoku, with as label the time it took a particular user to solve it. One can then search, for example, for a linear function over the features that best predicts the labels, or for a *decision tree* that does so.

More formally, we define the learning problem as follows. 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 $h \in H$ with respect to dataset E and the target hypothesis t. The goal is to find a hypothesis that minimizes the loss. This is a very general definition that encompasses both supervised and unsupervised settings, including clustering classification and regression.

For example, for linear regression, the data would be real-valued data $E \subset \mathbb{R}^d$ with real-valued labels identified by target function t, where $\forall \mathbf{e} \in E : t(\mathbf{e}) \in \mathbb{R}$. The goal is then to learn a linear function $h_\mathbf{c} : E \rightarrow \mathbb{R}$ 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.

More concretely for the sudoku solving time example, the target function would return the running time to solve the sudoku. The data would be three dimensional when using the number of empty cells, the average frequency of the numbers and the average number of choices left in each cell. The learned function could give, for example, high weight to the number of empty cells (more empty is longer to solve) and negative weight to the average frequency the numbers appear (higher frequency is easier).

Moreover, a range of machine learning methods such as (linear) regression and support vector machines can be expressed as standard optimization problems (often unconstrained), where the goal is to find an assignment to function parameters such that the loss is minimized.

For the linear regression with a sum of squared error (least squares regression), the optimization problem is defined as follows:

$$\underset{\mathbf{c}}{\text{minimize}} \quad \sum_{\mathbf{e} \in E} |\mathbf{e} \cdot \mathbf{c} - t(\mathbf{e})|^2$$

In case of *ridge* regression, the loss function includes a regularization component, to avoid fitting the given examples too exactly, by restricting the capacity of

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

This article has been accepted for publication in IEEE Intelligent Systems but has not yet been fully edited.
Some content may change prior to final publication.

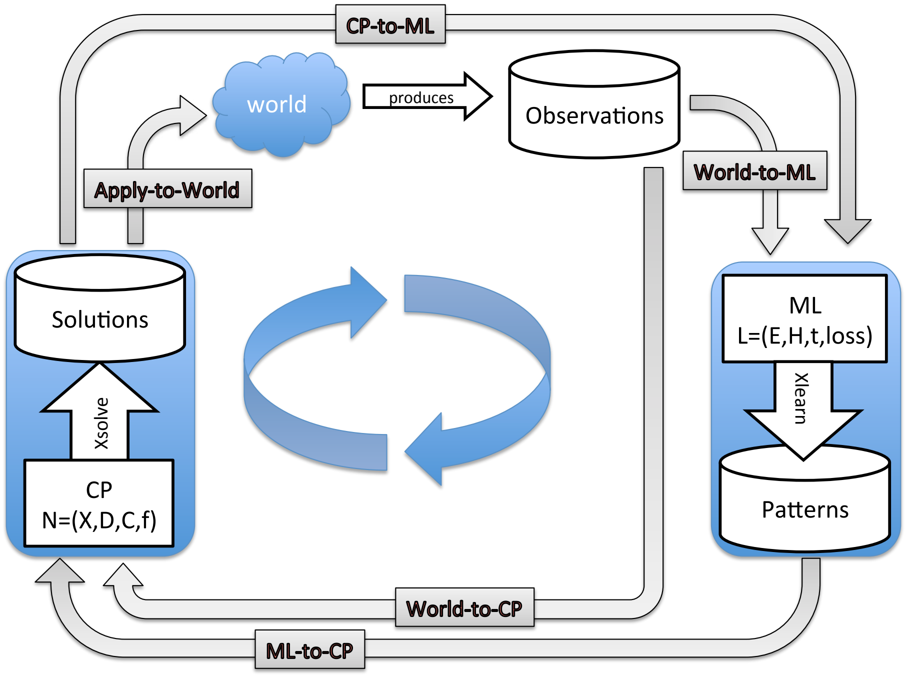

Figure 2: The inductive constraint programming loop

the weights:

$$\begin{aligned} \underset{\mathbf{c}}{\text{minimize}} \quad & \sum_{\mathbf{e} \in E} |\mathbf{e} \cdot \mathbf{c} - t(\mathbf{e})|^2 \\ \text{subject to} \quad & \sum_i |\mathbf{c}_i|^2 \leq 1 \end{aligned}$$

Similarly but more complex, support vector machine learning can also be seen as solving a convex optimization problem [STS11].

In practice, one does not typically use generic optimization techniques for such problems, but more specialized and scalable solving methods that exploit specific properties of the optimization problem.

3 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 2. We introduce each of the elements in the loop in turn.

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

The CP component is composed of a constraint network $N = (X, D, C, f)$ (f is optional), a 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.

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

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 from 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:

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

This article has been accepted for publication in IEEE Intelligent Systems but has not yet been fully edited.
Some content may change prior to final publication.

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

---

function CYCLE(Observations, *optional* Solutions)
  repeat
    $L_o \leftarrow$ World-to-ML(Observations)
    $L_p \leftarrow$ CP-to-ML(Solutions)
    construct L from L_o and L_p
    Patterns $\leftarrow$ applyXlearn(L)

    $N_o \leftarrow$ World-to-CP(Observations)
    $N_p \leftarrow$ ML-to-CP(Patterns)
    construct N from N_o and N_p
    Solutions $\leftarrow$ applyXsolve(N)
  until Apply-to-World(Solutions)
end function

Initially, World-to-ML is used to gather training data for 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. This amounts to extracting observations or training instances that the learning method will use as input. 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 constraint network is constructed using the output of World-to-CP and ML-to-CP. In many cases, a base constraint network representing the problem already exists in the CP component. The output of World-to-CP and ML-to-CP determines some of the parameters (e.g., weights) of the constraints in the base network, or some additional constraints to be added on top of the base network. In other cases, all the information to build the constraint network comes from the output of World-to-CP and ML-to-CP. Then, 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 particular 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.

A second observation we can make is that at each execution of the loop, XLearn (resp. Xsolve) is called on a learning problem (resp. constraint network)

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

This article has been accepted for publication in IEEE Intelligent Systems but has not yet been fully edited.
Some content may change prior to final publication.

potentially very similar to the one of the previous execution. It could be useful to use incremental learning and constraint solving algorithms, which would start from the previous solution to build the new one. However, incrementally solving combinatorial problems is far from being simple. Theory tells us that two very close problems can have totally different solving complexities. We thus do not address this issue in this paper, despite it can have an impact in practice.

4 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, physiotherapy 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. Therefore, important parameters of this scheduling problem 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.

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 cur-

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

rently 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 constraint 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. Figure 3 shows MiniZinc code for the task scheduling problem. In this model, the parameters of the problem setting are reflected as follows:

- the `dur` array represents the durations of all the tasks, as predicted by the ML component (line `3`);
- the `prev` array indicates for each task which task needs to be executed before this task; note that we assume that there is a dummy first task that precedes all tasks (line `4`);
- the `cap` array represents the capacity of the resources available (line `7`);
- the `use` array represents how many resources of each type need to be used to execute a certain task (line `8`).

The variables that need to be found are the `start` variables (line `11`), which indicate at which times the tasks need to be executed. The constant `max_time` represents the latest time at which a task may still start, this could be specified for each task separately as well.

The constraints are twofold:

- the constraint on line `14` is a `cumulative` constraint; for a given resource, it ensures for each time point that the use of the resource is within the capacity bound of that resource. Note that the `cumulative` constraint is a built-in constraint available in the MiniZinc language. Constraints that can involve any number of variables are called *global* constraints. They can capture complex structural constraints of the problem. Global constraints are an essential part of the efficiency of CP models.

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

This article has been accepted for publication in IEEE Intelligent Systems but has not yet been fully edited.
Some content may change prior to final publication.

```
1  % Tasks: duration and precedence
2  int: nbTasks; set of int: Tasks = 1..nbTasks;
3  array[Tasks] of int: dur;
4  array[Tasks] of int: prev;

5  % Resources: capacity and use
6  int: nbRes; set of int: Res = 1..nbRes;
7  array[Res] of int: cap;
8  array[Res,Task] of int: use;

9  % Variables: start times
10 int: max_time;
11 array[Tasks] of var 0..max_time: start;

12 % Resource capacities
13 constraint forall(r in Res) (
14   cumulative(start, dur, use[r], cap[r]) );

15 % Precedence between tasks
16 constraint forall(t in Tasks) (
17   start[t] > (start[prev[t]] + dur[prev[t]]) ));

18 % Minimize the amount of time
19 solve minimize max(t in Tasks) (start[t]+dur[t]);
```

Figure 3: The hospital scheduling problem in pseudo-Minizinc.

- the constraint on line 17 ensures that a task only executes after the task that should precede it has finished.

The optimization criterion is to minimize the makespan, that is, to assign the `start` variables so that the total amount of time used by the schedule is minimum (line 19).

To predict the durations of the tasks in the hospital, a regression problem needs to be solved. Many different models can be made for this regression problem, each corresponding to learning a different type of regression model. Arguably the most simple regression model is the linear model, in which the task duration prediction is based on a linear combination of the characteristics of the patient on which the task is executed.

The problem of learning such a regression model is formalized in Figure 4. Variables `X` and `Y` represent the training data, where `X` contains the descriptive attributes of various tasks and `Y` the historical durations of these tasks; variable `W` represents the weights of the features that we are learning.

Based on these weights, we can calculate an error for the predictions; line 10 calculates a weighted linear combination for each training observation, using the weights `W`; this prediction is used in line 12 to calculate an error for each observation. Line 15 minimizes the error over all observations, where line 17

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

```
% Dimension of the input data.
int: N; % Number of observations
int: M; % Dimension of observations

% Input data: observed data (X) and target labels (Y)
array[1..N, 1..M] of float: X;
array[1..N] of float: Y;

% Weights to fit (W[M+1] is constant term)
array[1..M+1] of var float: W;

% Calculate predictions and errors
array[1..N] of var float: Est =
             [ sum(j in 1..M) (W[j]*X[i,j]) + W[M+1] | i in 1..N ];
array[1..N] of var float: Err =
             [ Est[i] - Y[i] | i in 1..N ];

% Minimize the squared error
solve minimize norm2(Err);

% Auxiliary functions for computing the 2-norm
function var float: norm2(array[int] of var float: W) =
        sum(j in index_set(W)) ( W[j]*W[j]);
```

Figure 4: The hospital learning problem in pseudo-Minizinc.

defines that the errors for the individual observations are combined by summing the squared errors. Whereas this problem is formulated as a generic constraint optimization problem, many machine learning toolkits exist that have highly optimized algorithms for this problem.

The scheduling model and the machine learning model together define both components of the inductive constraint programming loop. We have 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 components is an appealing prospect, it is not a requirement for the applicability of the inductive constraint programming loop.

5 Other Examples of Applications

There are numerous other kinds of problems that can be captured in the inductive constraint programming loop. In the long version of this paper we describe three other real problems (optimizing bus schedules, car pooling, and energy-aware data centers) that can be expressed in a neat and efficient way through the inductive constraint programming loop [BDG+15]. In that long version we also present two existing academic problems (constraint acquisition and algorithm selection in a portfolio) that can be seen with a new eye through the inductive

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

constraint programming loop.

In our approach, the machine learning component is first applied and then the outcome is used by the CP program. In some of the applications we mentioned above (e.g., energy-aware data centers), it would also be suitable to do the machine learning while taking the operational cost (the outcome of the CP problem with the learned weights) into account. This can be achieved by making the operational cost a part of the loss function of the ML problem. One can then repeatedly iterate between solving the ML and CP component, before applying the schedule in the world [TR13].

6 Conclusion

After a brief introduction to constraint programming and machine learning, we have introduced the framework of inductive constraint programming. The key idea in the inductive constraint programming loop is that the CP and ML components interact with each other and with the world in order to adapt solutions to changes in the world. This is an essential requirement 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 by applying a partial solution, e.g. in the case of constraint acquisition. We have presented multiple examples of the use of inductive constraint programming loop in real-world problem settings. Many other settings exist, and as the frequency with which learning methods are used to produce schedules and other operational plans, the need for a framework that can adapt to changes in the world will increase.

Acknowledgments

This work was funded by the European Commission under the project Inductive Constraint Programming, contract number FP7-284715. We thank the members of the consortium for the intense collaborations and the project reviewers for their insightful comments.

References

[BDG+15] 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. *CoRR*, abs/1510.03317, 2015.

[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.

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

[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.

[Doy79] J. Doyle. A truth maintenance system. *Artif. Intell.*, 12(3):231–272, 1979.

[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.

[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.

[STS11] J. Shawe-Taylor and S. Sun. A review of optimization methodologies in support vector machines. *Neurocomputing*, 74(17):3609 – 3618, 2011.

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

[TVP03] I. Tsamardinos, T. Vidal, and M. E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003.

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE

[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.

Digital Object Identifier 10.1109/MIS.2017.265115706 1541-1672/$26.00 2017 IEEE