Learning Non-Linear Constraints using Tensors

Mohit Kumar[1], **Stefano Teso**[1], **Luc De Raedt**[1]

[1] KU Leuven

{mohit.kumar,stefano.teso,luc.deraedt}@cs.kuleuven.be

Abstract

Many problems in operations research require that constraints be specified in the model. Determining the right constraints is a hard and laborsome task. We propose an approach to automate this process using artificial intelligence and machine learning principles. So far there has been only little work on learning constraints within the operations research community. We focus on personnel rostering and scheduling problems in which there are often past schedules available and show that it is possible to automatically learn constraints from such examples. To realize this, we adapted some techniques from the constraint programming community and we have extended them in order to cope with multidimensional examples. The method uses a tensor representation of the example, which helps in capturing the dimensionality as well as the structure of the example, and applies tensor operations to find the constraints that are satisfied by the example. To evaluate the proposed algorithm, we used constraints from the Nurse Rostering Competition and generated solutions that satisfy these constraints; these solutions were then used as examples to learn constraints. Experiments demonstrate that the proposed algorithm is capable of producing human readable constraints that capture the underlying characteristics of the examples.

1 Introduction

Constraints are pervasive in practical scheduling and rostering problems. For example, hospitals usually generate a weekly schedule for their nurses based on constraints like the maximum number of working days for a nurse. As the number of nurses and the complexity of the constraints increases, however, generating the schedule manually becomes impossible. Organizations may hire domain experts to manually model the constraints, but this is expensive and time consuming. A tempting alternative is to employ constraint learning [?] to automatically induce the constraints from examples of past schedules.

Unfortunately, existing constraint learners are not tailored for this setting. Classical approaches like Conacq [?] and Inductive Logic Programming tools [?] focus on logical variables only, while scheduling constraints often include numerical terms. Very few approaches can handle this case. TaCLe [?] focuses on 2-D tabular data (Excel spreadsheets), while schedules are inherently multi-dimensional. To see what we mean, consider the nurse schedule shown in Table **??**. For each combination of nurse, day and shift the value of 1 represents that the nurse worked in that particular shift of that day, while a 0 means the nurse didn't work. It is easy to see that nurses, days, and shifts are independent of each and behave like different dimensions. ModelSeeker [?] is the only method that can handle such multi-dimensional structures, but it is restricted to global constraints only.

To address this issue, we propose COnstraint UsiNg TensORs (NL-COUNTOR), a novel constraint learning approach that leverages tensors for capturing the inherent structure and dimensionality of the schedules. In order to learn the constraints, NL-COUNTOR extracts and enumerates all (meaningful) slices of the input schedule(s), aggregates them through tensor operations, and then computes bounds for the aggregates to generate candidate numerical constraints. Some simple filtering strategies are applied to prune irrelevant and trivially satisfied candidates. When increasing the number of dimensions in the example, the rank of the tensor representing the example will increase accordingly but the proposed method NL-COUNTOR remains unchanged, so NL-COUNTOR can easily scale to large and complex schedules. The number of candidate sub-tensors however increases exponentially with the number of dimensions of $\mathbf{X}$.

We make the following key contributions: (**1**) A tensor representation of schedules and constraints appropriate for real-world personnel rostering problems. (**2**) A novel constraint learning algorithm, NL-COUNTOR, which uses tensor extraction and aggregation operations to learn the constraints hidden in the input schedules. (**3**) An empirical evaluation on real-world nurse rostering problems.

The paper is structured as follows. We present the method in Section 2, followed by evaluation on example instances in Section **??**. We conclude with some final remarks in Section **??**.

2 Method

We looked into most of the benchmark scheduling problems prevailing in the operations research community, some examples are Nurse Rostering, Course Timetabling, Home Care Scheduling and Project Scheduling. Most of the constraints observed in these problems are non-linear in nature but share some kind of structure, so we did some deep dive and came up with a mathematical structure to represent most of these constraints.

Let's see this running example before we define this structure. Consider a Nurse rostering model with the following variables.

$$\mathbf{X}_{n,s,d} = \begin{cases} 1, & \text{if nurse } n \text{ is allocated shift } s \text{ on day } d \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathbf{H}_{n} = \begin{cases} 1, & \text{if nurse } n \text{ is high skilled.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathbf{M}_{n} = \begin{cases} 1, & \text{if nurse } n \text{ is medium skilled.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathbf{L}_{n} = \begin{cases} 1, & \text{if nurse } n \text{ is low skilled.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathbf{R}_{s,d} = \text{Minimum number of high or medium skilled nurses required in shift } s \text{ on day } d$$

Considering these variables, a constraint like "the minimum number of high or medium skilled nurses working in a shift" can be represented as:

$$\sum_{n} \mathbf{X}_{n,s,d} \times \mathbf{H}_n + \sum_{n} \mathbf{X}_{n,s,d} \times \mathbf{M}_n \geq \mathbf{R}_{s,d} \quad \forall s, d \tag{1}$$

A generic version of the constraint above is represented below (Eq 2). Let $\mathbb{X}$ represents the set of all tensors $\mathbf{P}(\mathbb{X})$ represents the power set of $\mathbb{X}$ and $\mathbf{E}, \mathbf{F} \subseteq \mathbf{P}(\mathbb{X})$.

$$\sum_{\mathbf{e}\in\mathbf{E}}(\sum_{S_e}\prod_{\mathbf{Y}\in\mathbf{e}}\mathbf{Y}_{M_{e,y}}) - \sum_{\mathbf{e'}\in\mathbf{E'}}(\sum_{S_{e'}}\prod_{\mathbf{Y}\in\mathbf{e'}}\mathbf{Y}_{M_{e',y}}) \leq \mathbf{Z}_M \quad \forall \bigcup_{l\in E\cup E'}(\bigcup_{\mathbf{Y}\in\mathbf{l}} M_{l,y}/S_l)\bigcup M \tag{2}$$

Each constraint in Eq 2 can be uniquely identified by the following variables:

- E, E', Z
- $M_{l,y} \quad \forall l \in E \bigcup F \quad and \quad y \in l$
- $S_l \quad \forall l \in E \bigcup F$
- M

In other words, by enumerating through the various possible values of these variables, we can enumerate through all the possible constraints. For example, choosing the following assignments will give us the constraint represented by Eq 1

$$E = \emptyset$$
$$F = \{(X, H), (X, M)\}$$
$$Z = R$$
$$M_{(X,H),X} = M_{(X,M),X} = \{n, s, d\}$$
$$M_{(X,H),H} = M_{(X,M),M} = \{n\}$$
$$S_{(X,H)} = S_{(X,M)} = \{n\}$$
$$M = \{s, d\}$$

Enumerating through all the possible combinations is not a feasible solution as the number of combinations for this structure is huge and enumerating through all the possibilities will not scale for the real world problems. In this paper we define a two step solution to tackle this problem:

2.1 Signatures

Signatures are a pro active measure to refine the set of constraints to be enumerated. We define the following conditions which must be satisfied by a constraint in order for it to be a valid constraint. Let N_y represents the dimension set of $\mathbf{Y}$

- $M_{l,y} \subseteq N_y \quad \forall l \in E \bigcup F$
- $S_l \subseteq \bigcup_{\mathbf{Y}\in l} M_{l,y}$ for all $l \in E \bigcup F$
- $\bigcup_{\mathbf{Y}\in l} M_{l,y}/S_l = \bigcup_{\mathbf{Y}\in l'} M_{l,y}/S_{l'} \quad \forall l, l' \in E \bigcup F$
- $\bigcup_{\mathbf{Y}\in l} M_{l,y}/S_l \subseteq M$ or $\bigcup_{\mathbf{Y}\in l} M_{l,y}/S_l \supseteq M \quad \forall l \in E \bigcup F$
- $\mathbf{Y} \neq \mathbf{Y}'$ for any $\mathbf{Y}, \mathbf{Y}' \in l \quad \forall l \in E \bigcup F$
- $|E| + |F| \leq 2$
- $|l| \leq 2 \quad \forall l \in E \bigcup F$

If we apply just the last two signatures to Eq 2, the structure reduces to:

$$\sum_{S} \mathbf{X}_{M_1} * \mathbf{Y}_{M_2} + \sum_{S'} \mathbf{X}'_{M'_1} * \mathbf{Y}'_{M'_2} \leq \mathbf{Z}_M \tag{3}$$

$$\sum_{S} \mathbf{X}_{M_1} * \mathbf{Y}_{M_2} + \sum_{S'} \mathbf{X}'_{M'_1} * \mathbf{Y}'_{M'_2} \geq \mathbf{Z}_M \tag{4}$$

$$\sum_{S} \mathbf{X}_{M_1} * \mathbf{Y}_{M_2} - \sum_{S'} \mathbf{X}'_{M'_1} * \mathbf{Y}'_{M'_2} \leq \mathbf{Z}_M \tag{5}$$

2.2 Intelligent Enumeration

For enumerating intelligently we make a couple of assumptions, at the end we will also discuss how the algorithm will change if these assumptions don't hold.

- Numbers in the tensors are either all positive or all negative. In other words, a tensor can't have both positive and negative numbers
- Mod of the numbers in a tensor is all greater than or equal to 1

With the first assumption in place and given the structure of the constraints that we enumerate using Eq 2, we can safely assume that all the tensors have only positive numbers. This is because even if we have any negative data tensor **T**, we can transform it to **-T** for our algorithm without loosing any information as our equation considers both **T** and **-T** when enumerating through constraints.

With these assumptions we can now define the enumeration process. We use the following three properties to define an intelligent way to enumerate:

$$t_1 \not\leq t \implies t_1 + t_2 \not\leq t \quad \forall t_2 \geq 0 \tag{6}$$

$$t_1 \not\leq t \implies t_1 * t_2 \not\leq t \quad \forall t_2, s.t. |t_2| \geq 1 \tag{7}$$

$$t_1 \not\geq t \implies t_1 * t_2 \not\geq t \quad \forall t_2, s.t. |t_2| \leq 1 \tag{8}$$

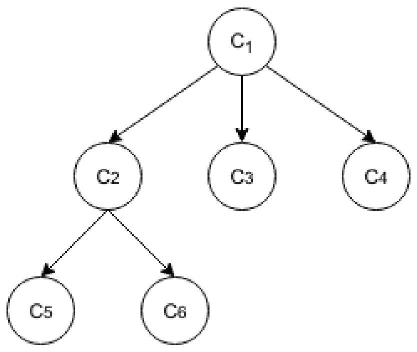

Figure 1: Constraint Dependency Tree: The root represents the most general constraint and as we keep going down the constraints become more specific.

We use this idea to define a constraint dependency tree as shown in Fig 1. A directed edge between two nodes represents entailment. For example, in Fig 1, C_2 entails C_1 which means whenever C_1 is satisfied C_2 will also be satisfied, which also means if C_2 is not satisfied then C_1 is not satisfied either.
There can be two approaches for enumeration: (1) Top Down or (2) Bottom Up.

Top Down: In the top down approach, we start with the most general constraint, if it's satisfied then we don't need to check any of it's descendents. For example, in Fig 1, we start with checking the validity of C_1, if it is satisfied we don't need to check any other constraints in the tree as they would be satisfied as well, and if it's not then we check each of it's children one by one and repeat the process.

Bottom Up: Bottom up works on the similar idea as the top down approach, here we start with the most specific constraint, in our case it would be C_5 or C_6, if it is not satisfied we don't need to check any ancestors of that constraint as it won't be satisfied either, but if it's satisfied we check each parent one by one and repeat the process.

We are going to discuss bottom up approach in the algorithm, but we can easily transform the algorithm to consider top down approach. So, to use the bottom up approach we always start with the most specific constraint and if it's not satisfied we can safely discard all the constraints which are ancestors of that constraint. We can easily represent any constraint using the set of variables defined earlier ($\{E, F, Z, M_{l,y}, S_l, M\}$). So let's see how does the most specific constraint look like based on our assumptions and signatures.

- $|F| = 2$ and $E = \phi$
- $|l| = 2 \quad \forall l \in E \bigcup F$
- $S_l = \bigcup_{\mathbf{Y} \in l} M_{l,y} \quad \forall l \in E \bigcup F$

Given two nodes $\{E, F, Z, M_{l,y}, S_l, M\}$ and $\{E', F', Z', M'_{l,y}, S'_l, M'\}$. The second one will pe a parent of the first one if and only if at least one of the following properties is satisfied keeping everything else unchanged.

- $|F'| < |F|$
- $|E'| > |E|$
- $\exists l' \in E \bigcup F \quad s.t. \quad |l'| < |l|$
- $\exists S'_l \in E \bigcup F \quad s.t. \quad S'_l \subset S_l$