# Constraint Learner for Scheduling Problems using Tensors

**Author1, Author2**

Declarative Languages and Artificial Intelligence, KU Leuven, Belgium

mohit.kumar@cs.kuleuven.be

## Abstract

When encountered with a constraint satisfaction problem, we usually assume the constraints to be given. However, there are many real world scenarios in which, we are given some examples and asked to reformulate the constraints, for instance when creating a schedule for nurses in a hospital, we might be given some old schedules and asked to create a new one. To address such problems, we propose an approach that learns constraints from the sample data. The method uses a tensor representation of the data, and applies tensor operations to find the constraints that are guaranteed to satisfy the given data. In this paper, we focus on scheduling problems like Nurse Schedule, Personal Schedule, Sports Schedule etc. We assess empirically several aspects of the proposed algorithm. Experiments demonstrate that the proposed algorithm is capable of producing human readable constraints that discover the underlying characteristics of the data and thus can be used to simulate it.

## 1 Introduction

In many fields, the generated data obeys some underlying constraints. For example, a hospital usually generates a weekly schedule for their nurses based on some rules, like maximum number of working days for a nurse or maximum number of nurses in a shift. If the hospital has a handful of nurses and some simple constraints are used, the schedule can be generated manually, but as the number of nurses and the complexity of constraints increases this can get really complicated and doing it every week makes it worse. Big organisations usually take help of domain experts to formulate constraints which can be used to generate the required data, which is expensive and time consuming.

An alternative approach is to use a constraint learner to induce the constraints from the past schedules, which then can be used to generate similar data. There has been a lot of work in the field of Constraint Learning ([**?**], [**?**], [**?**]). But none of these learners take into account the dimensionality of the data, by dimensions of the data we mean the variables which are present in the data. For example, in a nurse schedule the variables could be {Nurses, Days and Shifts}. These dimensions play an important part when learning constraints related to scheduling data as there are various constraints which are related to each dimension, we will see this in more detail in section 3.

To cater the demands of constraints for scheduling problems, we propose a novel approach of learning constraints using tensor representation, which is easily scalable to higher dimension, i.e. increasing number of variables in the data. The input to the algorithm is a working schedule and some additional background knowledge if applicable. Based on which the algorithm transforms the data to a tensor and then uses tensor operations to learn the constraints.

In section 2, we discuss some related works. The algorithm and mathematical formulation proposed in section 3 is the main contribution of this paper, section 4 discusses the experiments performed and their results. In section 5, we evaluate our algorithm on a real world data we picked from Nurse Rostering Competition page, to compare the accuracy of data generated using our proposed algorithm. We conclude with some final remarks in section 6.

## 2 Related Works

The type of constraints that we might encounter in data from different fields can vary dramatically. Recently there has been a lot of work to create constraint learners for different sets of data. For example, TaCLe can be used to learn relationship between rows and columns in a tabular data, which can be really helpful to learn the underlying rules followed by the rows and columns of a data and thus can also help us in compressing the data by replacing whole rows or columns by just a formula. ModelSeeker learns constraint models from positive examples, given a vector of values as input it groups them into regular patterns and then finds constraint patterns that apply to the group elements. Then there are ILP constraint learners like [**?**], where given a set of positive and negative examples, the method formulates a MILP problem and solves it to produce constraints which are guaranteed to separate feasible and infeasible regions and minimise the number of terms involved. In contrast, the approach presented in this paper takes into consideration the dimensions in the data, can easily represents constraints using mathematical operations on the data tensor and can be used to learn some important constraints used in the scheduling problems.

Table 1: Data Format

| | Mon | | | Tue | | | Wed | | | Thu | | | Fri | | | Sat | | | Sun | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S1 | S2 | S3 | S1 | S2 | S3 | S1 | S2 | S3 | S1 | S2 | S3 | S1 | S2 | S3 | S1 | S2 | S3 | S1 | S2 | S3 |
| Nurse1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| Nurse3 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Nurse4 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

# 3 Formalisation

Given a schedule, our goal is to learn constraints governing that schedule. Here we will give a formal shape to our problem. First, we will introduce a running example and define how we transform the input data into a tensor, then we will use some mathematical operations on the built tensor to learn the constraints related to scheduling problem, and then we will discuss some algorithms to filter out some unnecessary constraints learnt in the previous step, and then finally we will discuss an algorithm to learn some more specific constraints by considering the background knowledge about the variables.

## 3.1 Input Format

Consider a weekly schedule for nurses in a hospital, where each day has 3 different shifts:

| | Start Time | End Time |
|---|---|---|
| S1 | 5:00 | 13:00 |
| S2 | 13:00 | 21:00 |
| S3 | 21:00 | 5:00 |

We assume the data will be given in the format shown in Table 1. The columns represent different shifts on different days, while the rows represent different nurses. A value of 1 in the data means the nurse worked in that shift on that particular day, while a 0 means the nurse did not work in that shift.

Given this data as an input, first we figure out the variables in the data. In the example above we have 3 variables, $V = \{Nurses, Days, Slots\}$. The distinct values of a variable $V_i$ is represented as $Val(V_i)$ and $|Val(V_i)|$ is the number of elements in $Val(V_i)$. In the running example, $Val(Nurses) = \{Nurse1, Nurse2, Nurse3, Nurse4\}$ and $|Val(Nurses)| = 4$

If the number of variables in the data is $n$, we represent the data using a tensor $\mathbf{X}$ of rank n, by rank we mean the dimensionality of the tensor, or equivalently, the number of indices needed to label a component of the tensor. Once the dimensionality of the tensor is known, we need to know the shape, which can be represented as a tuple telling us the size for each dimension of the tensor. So, in our case, the shape of the tensor will depend on the number of distinct values for each variable. For our running example, the data has 3 dimensions as mentioned above where the first dimension corresponds to the number of nurses, second to the number of days and third to the number of shifts. So, we would represent the data using a tensor of rank 3 with shape $[4, 7, 3]$ where each value in the tuple represents the distinct number of values for the corresponding dimension. We would represent any value of the tensor by using the indices for the dimensions. So, if we want to see whether Nurse 2 worked or not on Monday in Shift 3, we can access the value from our data tensor as $\mathbf{X}_{1,0,2}$.

## 3.2 Constraint Representation

To represent the constraints mathematically, we need some definitions:

- Indicator function,
$$I(condition) = \begin{cases} 1 & if\ condition \\ 0 & otherwise \end{cases}$$
- If $v_i \in V_i$, We use $\mathbf{X}_{v_i}$ to represent a lower dimensional tensor by taking value $v_i$ for the ith dimension in $\mathbf{X}$. For example, $\mathbf{X}_{Mon}$ below shows us the working schedule of different Nurses for different slots on Monday
$$\mathbf{X}_{Mon} = \begin{array}{c} \\ \text{Nurse1} \\ \text{Nurse2} \\ \text{Nurse3} \\ \text{Nurse4} \end{array} \begin{array}{c} \begin{array}{ccc} \text{S1} & \text{S2} & \text{S3} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$
Similarly we can increase the number of fixed dimensions. For example,
$$\mathbf{X}_{Nurse1,Mon} = \begin{array}{c} \begin{array}{ccc} \text{S1} & \text{S2} & \text{S3} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \end{array}$$
- Cartesian Product,
$V_1 \otimes V_2 \otimes ... \otimes V_m = \{(v_1, v_2, ..., v_m) \mid v_i \in V_i\}$
- For any subset of variables, $V' \subseteq V$:
  - $CartesianProduct(V')$ = Cartesian product of elements of $V'$

Using the definitions and notations above, we now define a couple of functions which take the data tensor $\mathbf{X}$ and a subset of the variables, $V' \subseteq V$ as the input and output a tensor $\mathbf{X}'$. When applied in succession these functions can represent some important constraints which are frequently observed in scheduling problems. For example, number of employees each day, number of working days for each employee etc. Once we get these numbers we can apply max/min operation

to find the upper/lower bounds and thus the constraints governing the data.

- First, we define a function $Reduce$, which takes the data tensor $\mathbf{X}$ and a subset of variables $V'$, and reduces it to a tensor $\mathbf{X}'$ of rank $|Val(V')|$, by applying indicator function $I$ on each value $\mathbf{X}_e$, with the condition of $\mathbf{X}_e$ being non-zero, i.e. $I(\mathbf{X}_e \neq \mathbf{0})$, where $e \in CrossProduct(V')$.

  For example, if $V' = \{Nurses, Days\}$ in our example, then

  $$\mathbf{X}'[i,j] = I(\mathbf{X}_{Nurse(i),Day(j)} \neq \mathbf{0})$$

  Here, $\mathbf{X}'[i,j]$ tells us whether the ith Nurse worked in any slot on the jth Day. The complete tensor would be:

  $$\mathbf{X}' = Reduce(\mathbf{X}, V') = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (1)$$

- Now we define the second function, $Sum$, which again takes the data tensor $\mathbf{X}$ and a subset of variables $V'$, and outputs a tensor $\mathbf{X}'$ of rank $|Val(V')|$, but instead of applying indicator function, it sums all the values in $X_e$, where $e \in CrossProduct(V')$. For example, if $V' = \{Nurses, Days\}$ in our example, then

  $$\mathbf{X}'[i,j] = \text{Summation of values in } \mathbf{X}_{Nurse(i),Day(j)}$$

  Here, $\mathbf{X}'[i,j]$ is the number of slots worked by the ith Nurse on the jth Day. The complete tensor would be:

  $$X' = Sum(X, V') = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (2)$$

- Now, consider two disjoint non empty subsets $M$ and $S$ of $V$, and define:

  $$Count(X, M, S) = Sum(Reduce(X, M \cup S), S) \quad (3)$$

Let us understand what $Count$ represents using our example. If we take $M = \{Nurses\}$ and $S = \{Days\}$ then

$$Count(X, M, S) = \begin{bmatrix} 4 & 4 & 4 & 4 & 4 & 3 & 3 \end{bmatrix}$$

$Count(X, M, S)$ is the number of distinct employees working on different days of the week, if we take the max and min value from this tensor we can find out lower and upper bounds respectively. So, this example represents two constraints, the lower and upper bound on the number of working days for a Nurse.

By using Equation(3) and iterating through all the possible values of M and S we can represent a lot of constraints. If we consider both max and min of these counts as two separate constraints, we can cover exactly $2 * (3^n + 1 - 2^{(n+1)})$ constraints, where n is the number of variables in the data, by using a single mathematical structure defined in Equation(3).

In Table 2 we have shown the constraints covered by this expression in case of our 3 dimensional data for each value of M and S.

Table 2: Constraint Definitions

| M | S | $Count(X, M, S)$ |
|---|---|---|
| {Nurses} | {Days} | # of Working Days |
| {Nurses} | {Slots} | # of different slots worked by a Nurse |
| {Nurses} | {Days, Slots} | # of working slots |
| {Days} | {Nurses} | # of distinct employees / day |
| {Days} | {Slots} | # of slots with at least one employee on at least one day |
| {Days} | {Nurses, Slots} | # of employees each day |
| {Slots} | {Nurses} | # of distinct people working in a slot over a week |
| {Slots} | {Days} | # of working days for a slot |
| {Slots} | {Days, Nurses} | # of employees in a slot over a week |
| {Nurses, Days} | {Slots} | # of working slots in a day |
| {Nurses, Slots} | {Days} | # of working days in the same slot |
| {Days, Slots} | {Nurses} | # of employees / slot |

We can also notice that to represent each constraint, we are using three different functions. First, we reduce the data tensor, then we apply sum and finally we take max or min to learn the bounds. The constraints that we learn using these functions are still very limited, so we tried replacing some functions to learn some other important constraints we encounter in scheduling problems.

We will discuss some more functions that when replaces Sum in Equation(3) can learn some really important constraints related to the consecutive values in the data. For example, number of consecutive working days / slots for an employee. So, we define the function $MaxConsOne(\mathbf{X}, V')$, which takes the data tensor $\mathbf{X}$, and a subset of variables $V'$, and outputs a tensor $\mathbf{X}'$ of rank $|Val(V')|$ by taking maximum number of consecutive ones in $X_e$, where $e \in CrossProduct(V')$. When $X_e$ is a tensor of rank greater than 1, talking about consecutive ones doesn't make much sense, so for this function we will only consider the cases where $|V/V'| = 1$. For example, if M={Days} and

S={Nurses}, using $MaxConsOne$ instead of $Sum$ in Equation(3) would give us the number of max consecutive working days for each Nurse, taking a max would give us the upper bound for this constraint. Similarly for the lower bound we can define $MinConsOne(\mathbf{X}, V')$ which finds the minimum number of consecutive ones and then further taking a min will give us the lower bound for the constraint.

We used a couple of more functions in our experiments, $MaxConsZero$ and $MinConsZero$, which are very similar to $MaxConsOne$ and $MinConsOne$, it only differs in the fact that instead of taking Max/Min consecutive ones it takes Max/Min consecutive zeros, so these functions can be used to learn constraints like Max/Min number of consecutive holidays for an employee.

Below we give definition of a function very similar to $Count(X, M, S)$, the only difference is that it uses MaxConsOne instead of sum in Equation(3), similar functions can be defined for the other three functions discussed above.

$$\begin{gathered} ConsCount(X, M, S) \\ = \\ MaxConsOne(Reduce(X, M \cup S), S) \end{gathered} \tag{4}$$

In Table(3), we have compiled a list of constraints covered by $ConsCount(X, M, S)$ for different values of $M$ and $S$ and as mentioned earlier, we only use values of M such that $|M|$=1. A complete list of constraints covered by the other three functions is given in the appendix.

Table 3: Constraint Definitions

| S | M | $ConsCount(X, M, S)$ |
|---|---|---|
| {Nurses} | {Days} | # of Cons. Working Days |
| {Nurses, Days} | {Slots} | # of Cons. working slots in a day |
| {Nurses, Slots} | {Days} | # of Cons. working days in the same slot |
| {Slots} | {Nurses} | Doesn't make sense |
| {Days} | {Nurses} | Doesn't make sense |
| {Days, Slots} | {Nurses} | Doesn't make sense |
| {Nurses} | {Slots} | Doesn't make sense |
| {Days} | {Slots} | Doesn't make sense |
| {Slots} | {Days} | Doesn't make sense |

We can notice that in Table (2) and Table (3) above, there are a lot of combinations of $M$ and $S$ which doesn't make any sense, especially in Table (3). So, we need a method that could filter out the constraints which are not important or doesn't make any sense. We discuss such algorithm in the next section.

### 3.3 Filter Constraints

In our algorithm the type of constraints we are dealing with are bound constraints, which means we will find bounds for each case we consider, but some times the induced constraints might be redundant. For example, learning that minimum number of working days for a nurse is 0 every week doesn't give any extra information. So, it's important to devise some ways to filter out such redundant constraints. The following two filtering methods can be used to discard some obvious constraints.

- Discard the constraint if

  $$Max(Count(X, M, S)) = |CrossProduct(S)|$$

  For example, when M={Nurses} and S={Days, Slots}, and if we learn that $Max(Count(X, M, S)) = |CrossProduct(S)| = 21$, it translates to maximum number of working slots being 21 in a week using equation(3), which is an obvious bound, so we will drop this constraint. The same rule holds for $ConsCount$.

- Discard the constraint if

  $$Min(Count(X, M, S)) = 0$$

  Using the same logic as the previous example, we can neglect these constraints because they state an obvious constraint. Again, the same rule holds for $ConsCount$.

The last filtering method is specifically for constraints learned using $ConsCount$ in Equation(4). We can see in Table(3) that a lot of constraints don't make any sense, but there is a reason for that. We know that ordering of Nurses in the data is not important, so finding consecutive values of ones across Nurses doesn't make sense.

- For a constraint, if $X_i \in M$ such that ordering of values in $X_i$ is not important, then we neglect the constraints learned using Equation(4).

Using these filtering rules, we can discard a lot of constraints that are not important or do not make sense.

### 3.4 Adding Dimension

In the previous sections, we have seen a mathematical representation of constraints using tensor operations, and then we discussed some methods to filter out redundant or unimportant constraints. In this section, first we will discuss the scalability of the proposed algorithm, i.e., what happens when we increase the number of dimensions in the data. Once we discuss the scalability, we will introduce a method to include some background knowledge of variables to learn more specific constraints. For example, if categorise days into weekdays and weekends, we can use this data to learn constraints like, maximum number of nurses required each day on the weekends.

**Scalability:** When increasing the number of dimensions in the data, the rank of the tensor representing the data will increase accordingly. But, the constraint representation used in Equation(3) and Equation(4) remains the same. It's just that the number of ways we can split the set of variables into two non-empty disjoint subsets will increase, and hence

the number of constraints that could be learnt will increase as well which is quite intuitive. For example, if we have a data set like this:

| | M | Tu | W | Th | F | Sa | Su |
|---|---|---|---|---|---|---|---|
| Nurse1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| Nurse2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| Nurse3 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

we can only learn constraints related to Days and Nurses and their combinations, but when we add a dimension to the data:

| | Weekday | | | | | Weekend | |
|---|---|---|---|---|---|---|---|
| | M | Tu | W | Th | F | Sa | Su |
| Nurse1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| Nurse2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| Nurse3 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

We can learn all the constraints we had in the previous dataset as well as some more specific constraints related to the shift variable, like Max/Min Number of Nurses in a Shift. So, irrespective of the dimensions in the data, the proposed constraint learning algorithm remains the same. We will see how adding dimensions impact the time taken to run the algorithm in the experiments section.

**Background Knowledge:** Another way which is widely used to add information to the data is by adding some background knowledge about particular dimensions. For example, categorising nurses on the basis of their skills can induce more constraints like Maximum number of high skilled nurses required in each shift. To learn such constraints, first we transform the background knowledge to multiple 2-d tensors, where each tensor represents one particular value for the dimension. For example, if we have the data regarding skills of Nurses as in Table(4), we would create 2 tensors, one for each type of Nurses, i.e., High Skilled and Low Skilled. Now, these tensors are constructed in such a way that multiplying it with the data tensor, filters out the data unrelated to the value that the tensor represents. For example, the tensor corresponding to the High Skilled Nurses would be such that, when we multiply data tensor with it, it filters out the data related to the Low Skilled Nurses. Once we get the specific data, we can apply our constraint learner to learn the specific constraints related to the specific value of the variable, in this case the High Skilled Nurses.

Now, let us understand the construction of these variable's value specific 2-d tensor through an example. Consider the data given below in addition to the data given in Table(1).

Table 4: Background Knowledge

| | Type |
|---|---|
| Nurse1 | High Skilled |
| Nurse2 | Low Skilled |
| Nurse3 | Low Skilled |
| Nurse4 | High Skilled |

First we identify the number of distinct types of nurses, which in this case are 2, so we would need two 2-d tensors, one for the type High Skilled and the other for the Low skilled. Let, $H = \{Nurse1, Nurse4\}$, be the list of High Skilled Nurses and $L = \{Nurse2, Nurse3\}$, be the list of Low Skilled Nurses. If $|Val(Nurses)| = n$ and $|H| = m$ then the shape of the tensor corresponding to High Skill will be $m \times n$, where for each row $i$, all values will be zero except at the $jth$ column such that $H[i] = Nurses[j]$. So, the given background knowledge will be split into two tensors shown in figure 1.

First we convert the information into multiple rank-2 matrices, the number of required matrices would be same as the number of distinct types in the given information. For our example, we would need two matrices, one corresponding to High Skilled Nurses and the other Corresponding to Low Skilled Nurses as shown in Figure 1. These matrices will have 2 rows each, for each nurses of the corresponding type and four columns each representing the index of that nurse in the complete list. Once we convert the data into these tensors, we

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

High Skilled Nurses Low Skilled Nurses

Figure 1: Converting Background Knowledge to a Tensor

apply the concept of tensor multiplication to filter out the data that can give us more detailed constraints. To understand this, first let us see how tensor multiplication works, we all have seen how we can multiply two rank-2 tensors, which is simple matrix multiplication. But what if we have two tensors of different dimensions, let's say we want to multiply a rank-2 tensor A of shape [I, J] with a rank-3 tensor B of shape [J, K, L], we will get a rank-3 tensor C of shape [I, K , L] and the values in this new tensor is calculated based on the following formula:

$$C_{i,k,l} = \sum_{j} A_{i,j} \cdot B_{j,k,l}$$

Let us see what we get when we apply this tensor product on our example. So, let's take the first tensor from Figure 1, which represents the high skilled Nurses and let's multiply this to a small part of the data we have in Table1:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times$$

| | Mon | | |
|---|---|---|---|
| | S1 | S2 | S3 |
| Nurse1 | 1 | 0 | 0 |
| Nurse2 | 0 | 1 | 0 |
| Nurse3 | 0 | 0 | 1 |
| Nurse4 | 0 | 0 | 1 |

$$\Downarrow$$

| | Mon | | |
|---|---|---|---|
| | S1 | S2 | S3 |
| Nurse1 | 1 | 0 | 0 |
| Nurse4 | 0 | 0 | 1 |

So, when multiplied with the tensor representation of the high skilled nurses, we get a subset of the data with all the information for just the high skilled Nurses, now if we apply our constraint learner on this data we can learn all the constraints we learned earlier, but just specific to High Skilled Nurses. We can do this for each type of nurses to learn specific constraints.

So, given some background knowledge about each specific variable, we have come up with a transformation to a tensor representation, such that when we multiply this tensor with the data tensor, we can filter the data to contain information which can help us learn specific constraints related to a particular value of a variable. For example, if we consider the table with skill level of nurses, we can filter our data tensor to only contain information about high skilled nurses to learn corresponding constraints like, maximum number of high skilled nurses in a slot, or maximum number of working days for high skilled nurses. If we are given multiple tables, we can learn even more specific constraints for example, if we have another table categorising weekdays and weekends we can learn constraints like, maximum number of high skilled nurses working on a weekday. The number of such specific constraints would be exponential in number of tables given.

## 4 Experiments

We generated some schedules using the constraints we defined in the tables above, then we used our algorithm to learn the constraints using just the generated schedules. We also synthesised the constraints using the benchmark algorithm and compared the synthesised model across three parameters:

- Percentage of constraints correctly learned.
- How well the feasible region of the synthesised model match the actual feasible region.
- Accuracy of the simulated data using the synthesised model

We also picked some models and a number of corresponding schedules for each model from (to be discussed). Then, we synthesised the constraints using 80% of the schedules, and then checked whether the generated constraints satisfy the other 20% of the data.

## 5 Results

We used the CP solver provided by Google OR tools to generate solutions for Nurse Scheduling Problem. We then used this data to generate constraints using our constraint learner. Below is a table showing the constraints used to generate the data, constraints learnt from the generated data and number of constraints filtered.

The graph below shows the time taken in learning constraints versus the number of constraints learnt.