# Learning Parameters For the Sequence Constraint From Solutions

Émilie Picard-Cantin[1], Mathieu Bouchard[2],
Claude-Guy Quimper[1], and Jason Sweeney[2]

[1] emilie.picard-cantin.1@ulaval.ca, claude-guy.quimper@ift.ulaval.ca
Université Laval, 2325, rue de l'Université, Québec, Canada, G1V 0A6
[2] mbouchard@petalmd.com, jsweeney@petalmd.com
PetalMD, 350 boulevard Charest Est, Québec, Canada, G1K 3H5

**Abstract.** This paper studies the problem of learning parameters for global constraints such as Sequence from a small set of positive examples. The proposed technique computes the probability of observing a given constraint in a random solution. This probability is used to select the more likely constraint in a list of candidates. The learning method can be applied to both soft and hard constraints.

**Keywords:** Constraint Acquisition, Timetabling, Machine Learning, CSP, Global Constraints, Solution Counting, Markov Chain, Soft Constraints

## 1 Introduction

Accurate mathematical modeling requires a specific and complex training process and a lot of scheduling experience as there are as many models as there are problems. This is why modeling automation has become a popular field of study.

In this paper, we propose a statistical approach that detects the parameters of multiple global constraints such as Among and Sequence, two common constraints used in timetabling. This approach, based on machine learning, analyzes given positive examples (the only inputs from the user) and determines which constraint better explains these examples.

The first contribution of this paper is a technique to compute the probability of observing a specific Sequence constraint. Constraint candidates (satisfied by all examples) are compared using their individual probability. The candidate with the lowest probability of being observed, if it is not part of the model, is chosen and added to the optimization model. The second contribution is an improvement on solution counting for the Regular constraint using a simplified automaton and a matrix representation. The last contribution is a machine learning tool that can be applied to both soft and hard global constraints.

Section 2 describes the problem that motivates our research. Section 3 lists major contributions to the constraint acquisition field. Section 4 details our machine learning approach to detect the most likely Sequence constraint for a set of positive examples. Section 5 summarizes our results on a timetabling problem supplied by PetalMD, a specialist in medical scheduling.

## 2 Problem Description

Our research is motivated by a medical timetabling problem. A schedule is a table where rows are associated to employees and columns are associated to days. The cell $(e, t)$ contains the task assigned to employee $e$ on day $t$. The optimization model has two objectives : assigning the maximum number of tasks and minimizing the deviations from employees' workload targets. Each employee can be assigned at most one task at a time. Let $x_{et}$ be the task assigned to employee $e$ at time $t$. Let $R$ be the total number of employees and $T$ be the set of all task types. At least $\underline{c}_i$ and at most $\bar{c}_i$ employees must work on task $i$ every day. We use a global cardinality constraint [16], $\text{GCC}([x_{1t}, \ldots, x_{Rt}], \underline{c}, \bar{c})$ for each day $t$ to ensure these requirements are met.

Let $d$ be the total number of days in the schedule. An employee can be assigned at least $l$ and at most $u$ tasks taken from a set $V \subseteq T$ within a period of $k$ consecutive days. This limit is imposed by $\textsc{Sequence}(l, u, k, [x_{e1}, \ldots, x_{ed}], V)$ for every employee $e$, i.e. on each row of the schedule. Both global constraints Among and Sequence were introduced by Beldiceanu and Contejean [1]. The constraint $\textsc{Among}(l, u, [x_{ij}, \ldots, x_{i(j+k-1)}], V)$ ensures that $x_{ij}, \ldots, x_{i(j+k-1)}$ are assigned to values in $V$ at least $l$ times and at most $u$ times. The constraint $\textsc{Sequence}(l, u, k, [x_{i1}, \ldots, x_{id}], V)$ ensures that $\textsc{Among}(l, u, [x_{ij}, \ldots, x_{i(j+k-1)}], V)$ holds for every subset of $k$ consecutive variables in $\{x_{i1}, \ldots, x_{id}\}$.

This paper addresses the problem of learning parameters $l$, $u$, $k$ and $V$ of a Sequence constraint from a small set of positive solutions. In particular, we apply our method on manually completed schedules provided by PetalMD. Sequence is one of the most common constraint, yet the parameters are hard to extract from clients. The automated learning of this constraint will save PetalMD time and money. Typically, clients express their constraints informally and model creation is a long interactive process where scheduling experts present new schedules and clients tell them what is wrong with the new schedule until all constraints and parameters are determined.

## 3 Background

As explained in Section 1, automatic modeling is a popular field of study. In the present section, we list important concepts related to our research.

### 3.1 Constraint Acquisition

Bessiere et al. in [6], [5], [8] and [10] propose an algorithm named *ConAcq* learning constraint networks from positive and negative solutions using version space learning. Version space learning defines the search space for the constraint network as a set of constraint network candidates (hypothesis). Each hypothesis not satisfied by all positive examples is removed. *ConAcq* encodes each example as a set of clauses where the atoms are taken from the constraint vocabulary of

the library of constraints. A solution to the corresponding satisfiability problem is therefore an admissible constraint network.

O'Connel et al. [13] propose an interactive version space algorithm, which creates a first version space from examples given by the user. From one of the hypotheses, the system builds an qualifying example. The user accepts or rejects it, and the version space is updated accordingly. The algorithm terminates when the version space contains a single hypothesis.

Bessiere et al. [4] propose an active learning system named *QuAcq*. *QuAcq* adds one constraint at a time in the network by presenting partial queries, which are classified by the user as positive or negative. *QuAcq* choose queries that satisfy the constraints in the current network and violate at least one constraint in the library until no such queries exist.

Beldiceanu and Simonis [3] propose a constraint acquisition tool they refer to as *Model Seeker*. This tool builds a satisfaction model from positive examples using constraints from the global constraint catalog. The *Model Seeker* creates a list of candidates, all global constraints satisfying the examples, by generating sequences and matching them against the global constraints using the *Constraint Seeker* [2]. The candidates are ordered according to their pertinence, which is computed using multiple criteria, such as solution density and constraint popularity. A dominance check is performed to remove redundant constraints.

### 3.2 Solution Counting

The constraint $\textsc{Regular}([X_i, \ldots, X_n], \mathcal{A})$ [14] forces the word $[X_i, \ldots, X_n]$ to belong to the regular language defined by the deterministic finite automaton $\mathcal{A}$. The automaton $\mathcal{A}$ is composed of a finite list of states $\mathcal{Q}$, an alphabet $\Sigma$, a transition set $\delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$, an initial state $q_0 \in \mathcal{Q}$, and a set of final states $\mathcal{F} \subset \mathcal{Q}$ which determine the end of all accepted words. A sequence $X_1, \ldots, X_n$ is accepted by $\mathcal{A}$ if and only if there exists a sequence of states $q_0, \ldots, q_n \in \mathcal{Q}$ such that $(q_{i-1}, X_i, q_i) \in \delta$ for all $i \in \{1, \ldots, n\}$ and $q_n \in \mathcal{F}$.

Zanarini and Pesant [19] use dynamic programming to count the solutions that satisfy $\textsc{Regular}([X_1, \ldots, X_n], \mathcal{A})$. Let $\bar{\mathcal{A}}$ be the unfolded version of $\mathcal{A}$ where layer $L_i$ contains states attainable with the subsequence $[X_1, \ldots, X_{i-1}]$, see Figure 1. $L_1$ contains the initial state and $L_{n+1}$ contains all final states.

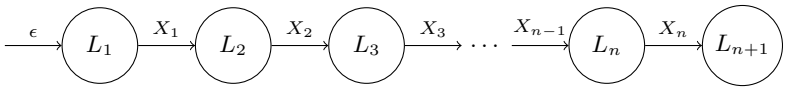


**Fig. 1.** Unfolded automaton, where $L_i$ contains states attainable with $[X_1, \ldots, X_{i-1}]$.

Let $v_{lq}$ be the state $q$ in layer $l$ and let $\#op(l, q)$ be the number of paths from $v_{lq}$ to a final state in layer $L_{n+1}$. Then, we have

$$\#op(n+1, q) = 1 \tag{1}$$

$$\#op(l,q) = \sum_{(v_{l,q},c,v_{l+1,q'})\in\delta} \#op(l+1,q'), \quad \forall q \in \mathcal{Q}, 1 \leq l \leq n\,. \tag{2}$$

The number of solutions that satisfy $\textsc{Regular}([X_1,\ldots,X_n],\mathcal{A})$ is $\#op(1,q_0)$.

Hoeve et al. [11] encode Sequence using Regular. Brand et al. [9] improve this encoding by simplifying $\textsc{Sequence}(l,u,k,[x_1,\ldots,x_d],V)$ to the constraint $\textsc{Sequence}(l,u,k,[y_1,\ldots,y_d],\{1\})$ where $\mathrm{dom}(y_i)=\{0,1\}$ and with the relation $y_i = 1 \iff x_i \in V$. Bessiere et al. [7] show how an automaton can encode the sliding of any constraint over a sequence of variables. Since $\textsc{Sequence}(l,u,k,[y_1,\ldots,y_d],\{1\})$ is defined as the sliding of the constraint Among on the sequence of variables $[y_1,\ldots,y_d]$, one can get an automaton in the following way. The states are labeled with sequences of zeros and ones of length at most $k-1$ and are partitioned into two sets. The states in $\mathcal{Q}^s$ are all possible sequences of length $s \leq k-2$ and are called *transitory states*. They are only visited at the beginning of the sequence. The states in $\mathcal{Q}^{k-1}$ are sequences of length $k-1$ that contains at least $l-1$ and at most $u$ occurrences of 1. The initial state is the empty sequence $\epsilon$ and the final states are $\mathcal{Q}^{k-1}$.

$$\mathcal{Q}^s = \{0,1\}^s, \forall\, s \in \{0,\ldots,k-2\}\,, \tag{3}$$

$$\mathcal{Q}^{k-1} = \{w \in \{0,1\}^{k-1} \mid l-1 \leq \sum_{i=1}^{k-1} w_i \leq u\}\,, \tag{4}$$

$$\mathcal{Q} = \bigcup_{i=0}^{k-1} \mathcal{Q}^i\,. \tag{5}$$

A state $w \in \mathcal{Q}^s$ for $s < k-1$ leads to the state $wc \in \mathcal{Q}^{s+1}$ upon reading the character $c \in \{0,1\}$, where $wc$ is the concatenation of the sequence $w$ with the character $c$. Finally, let $a$ and $b$ be two characters and $w$ a sequence of length $k-2$. Reading the character $b$ from state $aw \in \mathcal{Q}^{k-1}$ leads to state $wb \in \mathcal{Q}^{k-1}$ only if there are at least $l$ and at most $u$ occurrences of ones in the sequence $wb$.

$$\begin{aligned}\delta =& \{\langle w,c,wc\rangle \mid w \in \mathcal{Q}^s, wc \in \mathcal{Q}^{s+1}\} \\ &\cup \{\langle aw,b,wb\rangle \mid aw,wb \in \mathcal{Q}^{k-1}, l \leq a + \sum_{i=1}^{k-2} w_i + b \leq u\}\,.\end{aligned} \tag{6}$$

Figure 2 shows the automaton for $\textsc{Sequence}(1,2,3,[y_1,\ldots,y_d],\{1\})$. Note that states in $\mathcal{Q}^{k-1}$ are accepting because the transitions only lead to acceptable sequences and the initial state is $\epsilon$, the empty sequence. From the definition of the transitory states, certain states might be isolated in some automatons. Figure 3 illustrates a small example where a state labeled 0 is created but never used. We could reduce the automaton by removing those isolated states, but we keep them to simplify notation.

The number of solutions to Sequence can therefore be computed by counting the number of solutions to Regular, when used with the automaton that

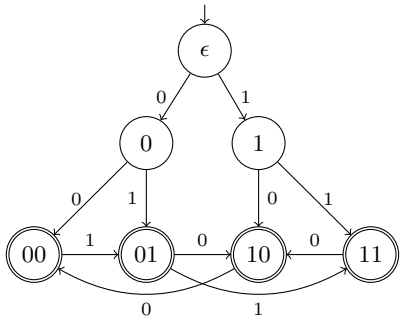


**Fig. 2.** Automaton corresponding to SEQUENCE$(1, 2, 3, [y_1, \ldots, y_d], \{1\})$

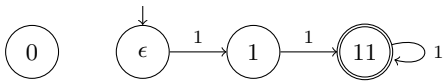


**Fig. 3.** Automaton corresponding to SEQUENCE$(3, 3, 3, [y_1, \ldots, y_d], \{1\})$

encodes SEQUENCE. As the automaton encodes $v \in V$ with the value 1 and $v \notin V$ with 0, we need a slightly modified version of the solution counting algorithm of Pesant [15] to take into account that there are $|V|$ ways to produce the value 1 and $|T \setminus V|$ ways to produce a 0. We replace equation (2) by the following.

$$\begin{aligned} \#op(l, q) = & \sum_{(v_{l,q}, 0, v_{l+1,q'}) \in \delta} |T \setminus V| \#op(l+1, q') \\ & + \sum_{(v_{l,q}, 1, v_{l+1,q'}) \in \delta} |V| \#op(l+1, q'), \quad \forall q \in \mathcal{Q}, 1 \leq l \leq n\,. \end{aligned} \tag{7}$$

For the constraint SEQUENCE$(l, u, k, [y_1, \ldots, y_n])$, since the automaton has $O(2^k)$ states, computing the number of solution is achieved in $O(n2^k)$.

### 3.3 Markov Chains

Markov chains can be used to compute the number of solutions for REGULAR by encoding the automaton as a transition matrix. A *Markov chain* [17] is a stochastic process defined by a set of time steps $t \in \{0, 1, \ldots, n\}$ and a set of states $i \in \{0, 1, \ldots, m\}$. The variable $X_t$ represent the state of the process at time $t$. If $X_t = i$, then the process is considered to be in state $i$ at time $t$. The probability of transitioning from state $i$ to state $j$ is given by a fixed probability $P_{ij}$. $P_{ij}$ is independent of the states before $i$. We must have $\sum_{j=1}^{m} P_{ij} = 1$ for all states $i$. The transition probabilities are gathered in a matrix $P$, called the *matrix of one-step transition probabilities* [17]. The n-step transition probabilities $P_{ij}^n$ are the probabilities of being in state $j$ after $n$ transitions if starting in state

$i$. Let $\alpha_i$ be the initial probability of state $i$. The unconditional probability of ending at state $j$ after $n$ transitions is

$$P[X_n = j] = \sum_{i=0}^{m} \alpha_i P_{ij}^n \,. \tag{8}$$

## 4 Constraint Acquisition

In this paper, we propose a statistical learning algorithm, which we divide into three steps. The first step analyzes the given solution, positive example, and makes a list of all constraints satisfied by the solution that call candidates. This is done by verifying each possible constraint against the solution. The second step ranks the selected constraints by computing the individual prior probability of the candidates using Markov chains. The last step chooses the constraint that explains the most the positive example we are given.

### 4.1 Listing Candidates

The first step of the learning process lists the constraints satisfied by the given solution. We call these satisfied constraints *candidates*, meaning that the real constraint we want to learn is in this subset. To learn the parameters of the constraint $\textsc{Sequence}(l, u, k, [y_1, \ldots, y_d], V)$, we create all possible sets of parameters and validate them against the solution. The only known parameter is $d$, since the scope of the constraint is known.

*Example 1.* Let the number of days be $d = 6$. Let $[y_1, \ldots, y_6] = [1, 1, 0, 0, 1, 1]$ be a solution for which we want to list all $\textsc{Sequence}(l, u, k, [y_1, \ldots, y_6], \{1\})$ candidates. Since $V$ is known, we refer to each constraint with the tuple $(l, u, k)$.

We have the following candidates. Since each window of length 1 has either 0 or 1 assignments, we have the candidate $(0, 1, 1)$. We have between 0 and 2 assignments for a window of length 2. Therefore, we have the candidate $(0, 2, 2)$. For a window of length 3, we either have 1 assignment or 2, which gives us the candidate $(1, 2, 3)$. Note that we also observe the candidates $(0, 2, 3)$, $(0, 3, 3)$ and $(1, 3, 3)$ which are less restrictive than $(1, 2, 3)$. For windows of length 4, we have the candidates : $\{(l, u, 4) : 0 \leq l \leq 2 \wedge 2 \leq u \leq 4\}$. If we continue this process up to $k = d = 6$, we obtain the list

$$\begin{aligned} C = &\{(0,1,1),(0,2,2)\} \cup \{(l,u,3) : 0 \leq l \leq 1 \wedge 2 \leq u \leq 3\} \\ &\cup \{(l,u,4) : 0 \leq l \leq 2 \wedge 2 \leq u \leq 4\} \cup \{(l,u,5) : 0 \leq l \leq 3 \wedge 3 \leq u \leq 5\} \\ &\cup \{(l,u,6) : 0 \leq l \leq 4 \wedge 4 \leq u \leq 6\} \,. \end{aligned}$$

### 4.2 Prior Probabilities

To determine which candidate should be learned, we compare them according to the probability that a random solution validates the candidate constraint.

The best choice is the constraint with the lowest probability because it is highly improbable that we observe this constraint by chance in the given solution.

Let $\text{dom}(x_j) = \{0, 1, \ldots, m\} = T$ for all $j \in \{1, \ldots, d\}$ and let $p_i = P[x_j = i]$. Consider $\textsc{Sequence}(l, u, k, [x_1, \ldots, x_d], V)$ with $V \subset T$ and its simplification $c = \textsc{Sequence}(l, u, k, [y_1, \ldots, y_d], \{1\})$. An *acceptable solution according to c* is a solution that validates $c$. Let $E_c$ be the random event of observing an acceptable solution for $c$. Let $X_c$ be the set of all solutions $x$ satisfying $c$ and let $P[x]$ be the probability of the specific solution $x$. Then, we have $P[E_c] = \sum_{x \in X_c} P[x]$.

Inspired from the automaton presented in Section 3, which accepts sequences satisfying a Sequence constraint, we define a Markov chain $\mathcal{A}$' that computes the probability that a random assignment satisfies the sequence constraint. For the constraint $\textsc{Sequence}(l, u, k, [y_1, \ldots, y_d], \{1\})$, let the states of the Markov chain be the set $\mathcal{Q}' = \mathcal{Q}^{k-1} \cup \{\sigma\}$, the sequences of length $k-1$ described in Section 3 and the sink state $\sigma$. The states of $\mathcal{Q}^s$ for $s \in \{0, \ldots, k-2\}$ are not required since a Markov chain does not need an initial state. Let $a, b \in \{0, 1\}$ and $w \in \{0, 1\}^{k-2}$. Then, $aw, wb \in \mathcal{Q}^{k-1}$. There is a transition from $aw$ to $wb$ with probability $\sum_{i \in V} p_i$ if $b = 1$ and with probability $1 - \sum_{i \in V} p_i$ if $b = 0$. There is a transition from $aw$ to the sink state $\sigma$ with probability $\sum_{i \in V} p_i$ if $a + \sum_{j=1}^{k-2} w_j = u$ and with probability $1 - \sum_{i \in V} p_i$ if $a + \sum_{j=1}^{k-2} w_j = l$. Finally, there is a transition from $\sigma$ to $\sigma$ with probability 1.

Let $M$ be the matrix of one-step transition probabilities for $\mathcal{A}$'. We can compute, for each state $q \in \mathcal{Q}^{k-1}$, the initial probability $g_q$ of being in $q$. Let $[v_{i1}, \ldots, v_{i,k-1}]$ be the sequence of values represented by the state $q_i$. Then,

$$g_i = \prod_{j=1}^{k-1} p_{v_{ij}} .$$

Note that we can never start with the state $\sigma$ and therefore $g_\sigma = 0$. Let $g = [g_1, \ldots, g_r]$ be the initial probabilities for all states $q \in \mathcal{Q}'$. To build a sequence of length $d$ from the sequences of length $k-1$ (states in $\mathcal{Q}'$), we need the $(d-k-1)$-step transition probabilities for $\mathcal{A}$'. Then, we have the equation

$$P[E_c] = \sum_{i=1}^{r-1} (g M^{d-k-1})_i . \tag{9}$$

This equation sums the probabilities of each acceptable path in the Markov chain according to $c$, a Sequence constraint. Note that $P[E_c]$ does not include solutions passing through $\sigma$ since they represent unacceptable sequences.

*Example 2.* Suppose we have $c = \textsc{Sequence}(1, 2, 3, [x_1, \ldots, x_8], \{1, 2\})$ with $T = \{0, 1, 2\}$ and where $p_0 = 5/6$ and $p_1 = p_2 = 1/12$. The associated Markov chain is illustrated in Figure 4. The path from 00 to 00 forms the sequence 000 with probability $p_0 = 5/6$. This transition violates the constraint. It is redirected to the sink $\sigma$. The path from 00 to 01 forms the sequences 001 and 002 and has a probability of $1/6 = p_1 + p_2$. The path from 10 to 01 creates the sequences 101, 102, 201 and $[2, 0, 2]$. This transition has a probability of $p_1 + p_2 = 1/6$.

There is a single transition of probability 1 leaving $\sigma$. It ensures that paths passing through unfeasible states represented by $\sigma$ are not counted in $P[E_c]$. The associated matrix of one-step transition probabilities is

$$M = \begin{bmatrix} 0 & 0 & 1/6 & 0 & 5/6 \\ 5/6 & 0 & 1/6 & 0 & 0 \\ 0 & 5/6 & 0 & 1/6 & 0 \\ 0 & 5/6 & 0 & 0 & 1/6 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The initial probabilities are $g = [25/36, 5/36, 5/36, 1/36, 0]$. Therefore, we have

$$P[E_c] = \sum_{i=1}^{4} (gM^4)_i \approx 0.1211 = 12.11\%.$$

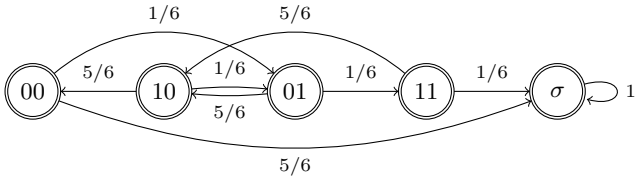


**Fig. 4.** Markov chain for SEQUENCE$(1, 2, 3, [x_1, \ldots, x_d], \{1, 2\})$ when $p_0 = 5/6$ and $p_1 = p_2 = 1/12$. $\sigma$ represents all forbidden transitions according to SEQUENCE

As shown in section 3, the solution counting algorithm for the constraint SEQUENCE$(l, u, k, [y_1, \ldots, y_d], \{1\})$ derived from Pesant's algorithm [15] has a computational complexity of $O(d2^k)$. This complexity can be improved. One can compute a power of a matrix using a decrease and conquer approach [12] based on this recurrence.

$$M^p = \begin{cases} I & \text{if } p = 0 \\ M \times M^{p-1} & \text{if } p \text{ is odd} \\ \left(M^{p/2}\right)^2 & \text{otherwise} \end{cases}$$

This algorithm requires $O(\log p)$ matrix multiplications and squaring. Given that multiplying two $n \times n$ matrices requires $O(n^\omega)$ steps ($\omega = 2.373$ when using William's matrix multiplication algorithm [18]), computing the probability that a random assignment satisfies SEQUENCE$(l, u, k, [y_1, \ldots, y_d], \{1\})$ can be achieved in $O(2^{\omega k} \log(d - k))$ steps.

Another method to compute $M^{d-k-1}$ is to use the spectral decomposition of the matrix $M = \mathcal{V}^{-1}\mathcal{D}\mathcal{V}$, where $\mathcal{V}$is the matrix of eigenvectors for $M$ and $\mathcal{D}$is the diagonal matrix such that $\mathcal{D}_{ii}$ is an eigenvalue of $M$. Therefore, we have $M^{d-k-1} = \mathcal{V}^{-1}\mathcal{D}^{d-k-1}\mathcal{V}$. Since computing the eigenvectors and eigenvalues is

done in cubic time, this decomposition computes the probability in $O(8^k)$ time. This complexity does not depend on the number of variables. This last method is preferable in situations where the number of variables $d$ for Sequence is largely superior to the number of variables $k$ for each corresponding Among.

### 4.3 Constraints Ordering

Consider a list of candidates $C = \{c_1, \ldots, c_n\}$. We will demonstrate that the best choice is the constraint with the lowest probability since all constraints in $C$ were observed in the solution set.

**Theorem 1.** *Let $C = \{c_1, \ldots, c_n\}$ be the list of candidates for a solution. Let $E_i$ be the random event of observing an acceptable solution for the constraint $c_i = \text{Sequence}(l_i, u_i, k_i, [y_1, \ldots, y_d], \{1\})$. The constraint $c_i$ explaining the most the current positive example, meaning that $P[E_1 \wedge \cdots \wedge E_n | E_i]$ is maximum, is such that $P[E_i] \leq P[E_j]$ for all $c_j \in C \setminus \{c_i\}$.*

*Proof.* $P[E_1 \wedge \cdots \wedge E_n | E_i]$ is the probability of observing a solution satisfying all constraints in $C$ considering that $c_i$ is satisfied. Bayes' theorem gives us

$$\begin{aligned} P[E_1 \wedge \cdots \wedge E_n | E_i] &= \frac{P[E_i | E_1 \wedge \cdots \wedge E_n] P[E_1 \wedge \cdots \wedge E_n]}{P[E_i]} \\ &= \frac{P[E_1 \wedge \cdots \wedge E_n]}{P[E_i]}. \end{aligned}$$

Therefore, we choose $c_i$ over $c_j$ if and only if

$$\begin{aligned} & P[E_1 \wedge \cdots \wedge E_n | E_i] \geq P[E_1 \wedge \cdots \wedge E_n | E_j] \\ \iff & \frac{P[E_1 \wedge \cdots \wedge E_n]}{P[E_i]} \geq \frac{P[E_1 \wedge \cdots \wedge E_n]}{P[E_j]} \iff P[E_j] \geq P[E_i]. \end{aligned}$$

### 4.4 Multiple Examples

If multiple positive examples are available, we can increase the performance of the learning process. The method is applied on each example individually, generating multiple separate candidate lists. Then, considering all examples are restricted with the same Sequence constraint, we keep only the candidates that are present for all examples. The idea is that the real constraint must be satisfied in all examples.

### 4.5 Constraint Dominance

We say that a constraint $c_1$ dominates another $c_2$, noted by $c_1 \succ c_2$, if all solutions to $c_1$ are solutions to $c_2$. Following Beldiceanu et al. [2], we reduce computation time by removing dominated constraints from the list of candidates. The concept of dominance is only applied to constraints with the same scope.

If $c_1 \succ c_2$, then the probability of observing a solution for $c_1$ is lower (or equal) than the probability of observing a solution for $c_2$ since $c_1$ is more restrictive than $c_2$. Therefore, the classifier will choose $c_1$ over $c_2$ and removing dominated constraints does not affect the final choice of the classifier. Moreover, the dominance check is quick and dominance relations can be computed beforehand and stored. Note that if multiple examples are used, the dominance check is performed after collecting all candidate constraints from every example. Indeed, a constraint dominates another one only if it is a candidate for all examples.

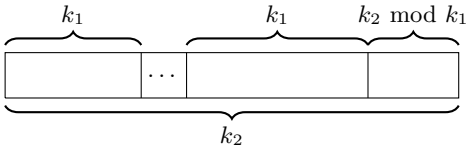


**Fig. 5.** When $k_1 < k_2$

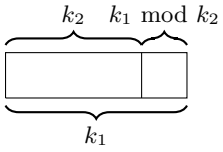


**Fig. 6.** When $k_1 > k_2$

- If $k_1 = k_2$, then $c_1 \succ c_2$ if and only if $u_1 \leq u_2$ and $l_1 \geq l_2$.
- If $k_1 < k_2$, then a maximum of $u_1$ assignments of values in $V$ for every window of length $k_1$ allows a maximum of

$$\bar{a} = \lfloor k_2/k_1 \rfloor \times u_1 + \min(u_1, k_2 \mod k_1)$$

  in a window of length $k_2$. Similarly, a minimum of $l_1$ assignments of values in $V$ for every window of length $k_1$ imposes a minimum of

$$\underline{a} = k_2 - [\lfloor k_2/k_1 \rfloor \times (k_1 - l_1) + \min((k_1 - l_1), k_2 \mod k_1)]$$

  in a window of length $k_2$. Therefore, $c_1 \succ c_2$ if and only if $\bar{a} \leq u_2$ and $\underline{a} \geq l_2$. See Figure 5 for a visual representation.
- If $k_1 > k_2$, then $c_1 \succ c_2$ if and only if $(u_1 \leq u_2 \vee k_2 = u_2) \wedge (l_1 \geq l_2 \vee k_2 = l_2)$. See Figure 5 and Figure 6 for a visual representation.

### 4.6 Classifier

Algorithm 1 is a summary of the procedure to determine the SEQUENCE constraint from a set of positive examples.

### 4.7 Soft Constraints

The proposed approach can be used to learn soft constraints. The listing of candidates needs to be adapted in order to select constraints which are violated by the given example. Let $\beta \in [0, 1]$ be the accepted percentage of violations. We consider SEQUENCE$(l, u, k, [y_1, \ldots, y_d], \{1\})$ satisfied if at least $(1 - \beta) \times 100\%$ of

**Algorithm 1:** How to determine the parameters of a SEQUENCE constraint from a set of positive examples.

**Data**: Positive examples of $d$ variables with the same configuration.
**Result**: The parameters of a SEQUENCE constraint.

```
begin
    List all SEQUENCE candidates (l, u, k, V) satisfying all examples;
    Apply the dominance check to remove dominated constraints;
    Select, from the remaining candidates, the one with the lower probability of
    being observed;
end
```

the corresponding AMONG$(l, u, [y_j, \ldots, y_{j+k-1}], \{1\})$ constraints are satisfied by the example.

The Markov chain also needs to be adapted to accept violations. The new set of states is $\mathcal{Q} = \{0,1\}^{k-1} \cup \{\sigma\}$, the set of all sequences of length $k-1$ augmented with a sink state $\sigma$. The violation degree of a sequence $Y \in \{0,1\}^k$, according to the constraint AMONG$(l, u, [y_i, \ldots, y_{i+k-1}], \{1\})$, is given by

$$d(Y) = \max\left(\sum_{i=1}^{k} Y_i - u, l - \sum_{i=1}^{k} Y_i, 0\right).$$

Let $w \in \{0,1\}^{k-2}$, $a, b \in \{0,1\}$ and $h = \min(d(aw0), d(aw1))$. Let $v$ be the user defined probability of observing $d(awb) > h$, i.e. the probability of reading a character that does not minimize the degree of violation. Let $P[y_i = 1] = \sum_{v \in V} p_v$ and $P[y_i = 0] = 1 - P[y_i = 1]$. If $d(awb) = h$, then we have a transition from state $aw$ to state $wb$ with probability $P[y_i = b]$. If $d(awb) > h$, then we have a transition from $aw$ to $wb$ with probability $vP[y_i = b]$ and a transition from $aw$ to $\sigma$ with probability $(1-v)P[y_i = b]$. Finally, there is a transition from $\sigma$ to $\sigma$ with probability 1.

We assume that the events of accepting the different violated AMONG constraints are independent. Note also that the probability does not depend on the degree of violation of the AMONG constraint, but our model could easily be adapted to do so.

The last modification is the new vector of initial probabilities $g$. For a state $q \in \mathcal{Q}$, let $r = \sum_{i=1}^{k-1} q_i$. The initial probability for $q$ is

$$g_q = P[y_i = 1]^r \times P[y_i = 0]^{k-1-r} \left(v^{d(0q)} P[y_i = 0] + v^{d(1q)} P[y_i = 1]\right).$$

The initial probability of the sink state $\sigma$ is $g_\sigma = 1 - \sum_{q \in \mathcal{Q}} g_q$.

*Example 3.* Suppose $v = 1/10$, $p_0 = 5/6$ and $p_1 = 1/6$. The Markov chain for the soft constraint SEQUENCE$(1, 2, 3, [y_1, \ldots, y_d], \{1\})$ is illustrated in Figure 7. The initial probabilities for this example are $g_{00} = 25/144$, $g_{01} = g_{10} = 5/36$, $g_{11} = 17/720$ and $g_\sigma = 1 - 342/720 = 378/720$.

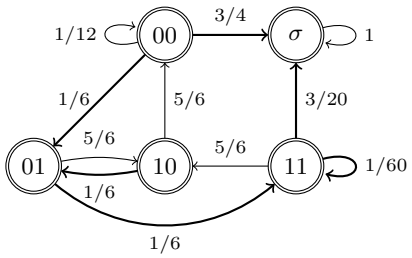


**Fig. 7.** Graphical representation of the Markov chain corresponding to the soft constraint SEQUENCE$(1, 2, 3, [y_1, \ldots, y_d], \{1\})$ when $v = 1/10$, $p_0 = 5/6$ and $p_1 = 1/6$

## 5 Experiments

Experiments focus on the timetabling problem presented in Section 2. For every instance there are three employees and three medical task types (day, evening, night) repeated each day of an 84 days scheduling period. Tasks are represented by positive integers and 0 is reserved for unassigned days. In our context, the employees want to have similar workloads. As there are $84 \times 3$ tasks to assign, the target workload of each employee is 84. An employee $e$ can only work on one task each day. The targets are encoded with soft constraints limiting workloads. We minimize deviations from targets in the objective function. Each task requires one employee, which is encoded using a GCC. There is a constraint SEQUENCE$(0, u, k, [x_{r1}, \ldots, x_{r84}], V)$ where $u$, $k$ and $V$ are to be learned.

Remember that $x_{et} \in \{0, 1, \ldots, m\}$ is the task assigned to employee $e$ at time $t$ and that $y_{et} \in \{0, 1\}$ determines if $x_{et} \in V$ or not. Let $z_{et}$ be the Boolean variable, which encodes if $e$ is working or not at time $t$. Therefore, we have $z_{et} = 0$ if $x_{et} = 0$ and $z_{et} = 1$ if $x_{et} > 0$. Let $\Delta_e$ be the deviation from the target for the employee $e$. Let $H \subseteq \{1, 2, 3\}$ be a subset of employees. The general model that produced the instances is as follows.

$$\max \quad \sum_{e=1}^{3} \sum_{t=1}^{84} z_{et} - \sum_{e=1}^{3} 0.1\Delta_e$$

$$84 - \sum_{t=1}^{84} z_{et} \leq \Delta_e, \quad \forall e \in \{1, 2, 3\}$$

$$z_{et} = 1 \iff x_{et} \geq 1, \quad \forall e \in \{1, 2, 3\}, \forall t \in \{1, \ldots, 84\}$$

$$\text{GCC}([x_{1t}, x_{2t}, x_{3t}], [0, 0, 0], [1, 1, 1]]), \quad \forall t \in \{1, \ldots, 84\}$$

$$\text{SEQUENCE}(0, u, k, [x_{e1}, \ldots, x_{e84}], V), \quad \forall e \in H$$

$$x_{et} \in \{0, 1, \ldots, m\}, \quad \forall e \in \{1, 2, 3\}, \forall t \in \{1, \ldots, 84\}$$

$$y_{et}, z_{et} \in \{0, 1\}, \quad \forall e \in \{1, 2, 3\}, \forall t \in \{1, \ldots, 84\}$$

$$\Delta_e \in \mathbb{N}, \quad \forall e \in \{1, 2, 3\}$$

As a benchmark[3], we generate some models and find several optimal solutions for each of them. These solutions are the positive examples that we feed into our algorithm, in order to test whether it returns the constraints that were actually used to generate the solutions. The generated models can be divided into three categories. We create a first set of schedules (A) where the same SEQUENCE constraint is applied to all employees ($H = \{1, 2, 3\}$). Then, we produce five different schedules for all possible combinations of $(u, k)$ with $1 \leq u < k \leq 7$ and $V = \{1, 2, 3\}$. We create a second set of instances (B) where the same SEQUENCE is applied to $e \in \{1, 2\}$. The last employee is not subject to any SEQUENCE constraint. Again, we produce five schedules for all $(u, k)$ with $V = \{1, 2, 3\}$. Finally, we create a last set (C) where SEQUENCE is applied to a subset of tasks and is the same for all employees. For all $V \in \{\{1\}, \{1, 2\}\}$ and for all $(u, k)$, we produce five schedules. Both sets A and B contain 21 instances (105 schedules) and the set C contains 42 instances (210 schedules).

Because of the GCC and the target workloads, all tasks tend to have the same frequency in the schedules. To test our method on instances where values in $V$ do not have the same probability, we create new schedules. For each instance in the sets $A$, $B$, and $C$ previously described, we modify the schedules so that each value $i \in V$ has a specific probability $p_i$ to appear in the schedule. Let $I_e = \{t \in \{1, \ldots, 84\} : x_{et} \in V\}$. For $t \in I_e$, we randomly choose a value in $V$ and assign it to $x_{et}$ using on of the following probability distribution: $(P[v_1], P[v_2]) = (0.1, 0.9)$ if $|V| = 2$ and $(P[v_1], P[v_2], P[v_3]) = (0.1, 0.4, 0.5)$ if $|V| = 3$. Then, we apply the same modification process with $I'_e = \{t \in \{1, \ldots, 84\} : x_{et} \in (T \setminus V)\}$. The new schedule still satisfies SEQUENCE since the tasks in $V$ are shuffled between themselves. The GCC might not be satisfied and the solution might not be optimal, but our goal is to test our learning tool on instances with unbalanced distributions of tasks.

The probability of each task is unknown to the learning process. For a given example, we compute the probability of each task with $P[x_{et} = v] = |\{t : x_{et} = v\}|/84$ for each employee $e$. We approximate the probability $P[v]$ with the average of these probabilities over the five examples. The learning algorithm is applied individually on each employee. We compare the results obtained using this approximation with the results using the solution counting algorithm, which is one of the criteria used by Beldiceanu et al. [3] to rank constraint candidates. We note the statistical learning algorithm *Statistical* and the solution counting version *Counting*. We note *uniform* the instances with uniformly distributed tasks and *non-uniform* the instances with non-uniformly distributed tasks.

The results obtained for only one positive example are illustrated in Table 1. The results are divided by instance set (A, B, or C). # is the total number of instances in the category (one per employee). Inspired by Beldiceanu et al. [2], we classify our results according to the position of the real constraint in the list of candidates returned.

As shown in Table 1, the first candidate (#1) is the real constraint for all 63 instances of category A while it is the real constraint for 108 out of 126 instances

[3] The benchmark is available upon request to the authors.

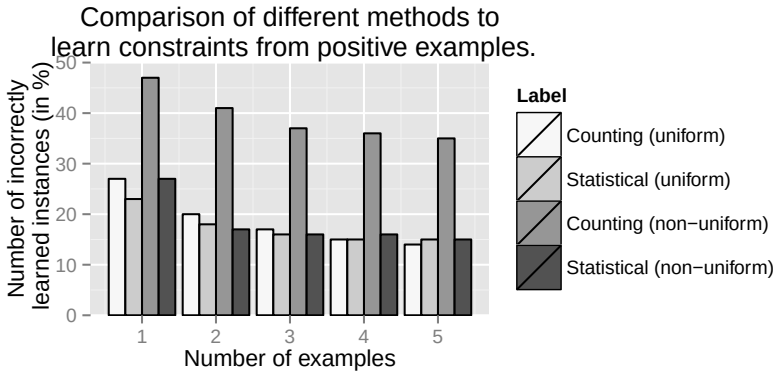


**Fig. 8.** Number of incorrectly classified instances in percentage for each method and each number of examples

of category C. Table 1 also shows that *Counting* is less efficient on non-uniform instances. For example, 213 uniform instances were correctly learned (ranked #1) while only 164 non-uniform instances were correctly ranked. This is a loss of 49 instances. In comparison, *Statistical* is more stable since it only "lost" 2 correctly ranked instances with the non-uniform task probabilities. This shows that *Statistical* depends on the individual probability of the different values.

**Table 1.** Results for *Counting* and *Statistical* with a single positive example by instance

| | | *Counting* | | | | | | | | *Statistical* | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Uniform | | | | Non-uniform | | | | Uniform | | | | Non-uniform | | | |
| | # | #1 | #2 | #3 | Other | #1 | #2 | #3 | Other | #1 | #2 | #3 | Other | #1 | #2 | #3 | Other |
| A | 63 | 63 | 0 | 0 | 0 | 61 | 1 | 1 | 0 | 63 | 0 | 0 | 0 | 63 | 0 | 0 | 0 |
| B | 63 | 42 | 0 | 0 | 21 | 41 | 1 | 0 | 21 | 42 | 0 | 0 | 21 | 42 | 0 | 0 | 21 |
| C | 126 | 108 | 12 | 1 | 5 | 62 | 7 | 7 | 50 | 108 | 11 | 3 | 4 | 106 | 13 | 2 | 5 |
| Total | 252 | 213 | 12 | 1 | 26 | 164 | 9 | 8 | 71 | 213 | 11 | 3 | 25 | 211 | 13 | 2 | 26 |

Figure 8 illustrates the summary results for each method and each number of examples. We can see that, for the instances where tasks are uniformly distributed, *Statistical* is better but *Counting* quickly catches up as the number of examples increases. As illustrated, the lack of uniformity of tasks impacts the performance of both methods. *Statistical* quickly regains the loss with only 4 examples, while *Counting* is still far behind (approximately 20% apart).

When the real constraint is dominated by one or many candidates, it is removed from the candidate list and the instance is incorrectly classified. We consider this a false negative, as it is impossible to rightly classify this type of

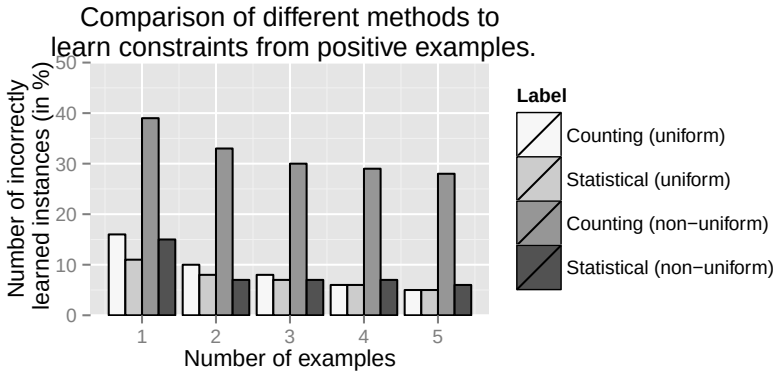


**Fig. 9.** Number of incorrectly classified instances in percentage after false negatives have been removed

instances without further prior information about the problem. Figure 9 illustrates the summary results after the false negatives are removed.

## 6 Conclusion

In this paper, we proposed a statistical learning algorithm that can be applied to both soft and hard global constraints that can be formulated as an automaton, such as Sequence, Among, Knapsack, Stretch, etc. This algorithm uses a new technique to compute the probability of observing a random solution for a given constraint. *Statistical* has proven to be more efficient in the ranking of candidates than the solution counting algorithm, when tested on scheduling instances. For instances where values are uniformly distributed, *Statistical* requires less positive examples to achieve the same results as other methods. This is important in scheduling, where as little as four examples might represent more than a year of data. For instances with non-uniformly distributed values, we showed that *Statistical* is largely better than *Counting*.

## References

1. Beldiceanu, N., Contejean, E.: Introducing global constraints in chip. Mathematical and computer Modelling 20(12), 97–123 (1994)
2. Beldiceanu, N., Simonis, H.: A constraint seeker: Finding and ranking global constraints from examples. In: Proceedings of the 17th International Conference of Principles and Practice of Constraint Programming (CP 2011). pp. 12–26. Springer (2011)
3. Beldiceanu, N., Simonis, H.: A model seeker: Extracting global constraint models from positive examples. In: Proceedings of the 18th International Conference of Principles and Practice of Constraint Programming (CP 2012). pp. 141–157. Springer (2012)

4. Bessiere, C., Coletta, R., Hebrard, E., Katsirelos, G., Lazaar, N., Narodytska, N., Quimper, C.G., Walsh, T.: Constraint acquisition via partial queries. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013). pp. 475–481. AAAI Press (2013)
5. Bessiere, C., Coletta, R., Koriche, F., O'Sullivan, B.: A sat-based version space algorithm for acquiring constraint satisfaction problems. In: Proceedings of the 16th European Conference on Machine Learning (ECML 2005). pp. 23–34. Springer (2005)
6. Bessiere, C., Coletta, R., Koriche, F., O'Sullivan, B.: Acquiring constraint networks using a sat-based version space algorithm. In: Proceedings of the 21st National Conference on Artificial Intelligence. pp. 1565–1568. No. 2, AAAI Press (2006)
7. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Quimper, C.G., Walsh, T.: Reformulating global constraints: the slide and regular constraints. In: Proceedings of the 7th International Symposium on Abstraction, Reformulation, and Approximation (SARA 07). pp. 80–92. Springer (2007)
8. Bessiere, C., Koriche, F., Lazaar, N., O'Sullivan, B.: Constraint acquisition. Artificial Intelligence (In Press) (2015)
9. Brand, S., Narodytska, N., Quimper, C.G., Stuckey, P., Walsh, T.: Encodings of the sequence constraint. In: Proceedings of the 13th International Conference of Principles and Practice of Constraint Programming (CP 2007). pp. 210–224. Springer (2007)
10. Coletta, R., Bessiere, C., O'Sullivan, B., Freuder, E.C., O'Connell, S., Quinqueton, J.: Constraint acquisition as semi-automatic modeling. In: Proceedings of the 23rd SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI 2003). pp. 111–124. Springer (2004)
11. van Hoeve, W.J., Pesant, G., Rousseau, L.M., Sabharwal, A.: Revisiting the sequence constraint. In: Proceedings of the 12th International Conference of Principles and Practice of Constraint Programming (CP 2006), pp. 620–634. Springer (2006)
12. Levitin, A.: Introduction to the Design and Analysis of Algorithms. Pearson Education (2011)
13. O'Connell, S., O'Sullivan, B., Freuder, E.C.: A study of query generation strategies for interactive constraint acquisition. In: Applications and Science in Soft Computing, pp. 225–232. Springer (2004)
14. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Proceedings of the 10th International Conference of Principles and Practice of Constraint Programming (CP 2004). pp. 482–495. Springer (2004)
15. Pesant, G.: Counting solutions of csps: A structural approach. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005). pp. 260–265. Morgan Kaufmann Publishers Inc. (2005)
16. Régin, J.C.: Generalized arc consistency for global cardinality constraint. In: Proceedings of the 13th National Conference on Artificial Intelligence - Volume 1 (AAAI 1996). pp. 209–215. AAAI Press (1996)
17. Ross, S.M.: Introduction to probability models. Elsevier (2014)
18. Williams, V.V.: Multiplying matrices faster than coppersmith-winograd. In: Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC'12). pp. 887–898. ACM (2012)
19. Zanarini, A., Pesant, G.: Solution counting algorithms for constraint-centered search heuristics. Constraints 14(3), 392–413 (2009)