Controlling Some Statistical Properties of Business Rules Programs

Olivier Wang, Leo Liberti

Controlling some statistical properties of Business Rules programs

Olivier Wang[1,2], Leo Liberti[2]

[1] IBM France, 9 Rue de Verdun, 94250 Gentilly, France
[2] CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France
{olivier.wang,leo.liberti}@polytechnique.edu

Abstract.

Business Rules programs encode decision-making processes using "if-then" constructs in a way that is easy for non-programmers to manipulate. A common example is the process of automatic validation of a loan request for a bank. The decision process is defined by bank managers relying on the bank strategy and their own experience. Bank-side, such processes are often required to meet goals of a statistical nature, such as having at most some given percentage of rejected loans, or having the distribution of requests that are accepted, rejected, and flagged for examination by a bank manager be as uniform as possible. We propose a mathematical programming-based formulation for the cases where the goals involve constraining or comparing values from the quantized output distribution. We then examine a simulation for the specific goals of (1) a max percentage for a given output interval and (2) an almost uniform distribution of the quantized output. The proposed methodology rests on solving mathematical programs encoding a statistically supervised machine learning process where known labels are an encoding of the required distribution.

Keywords: Distribution learning, mixed-integer programming, statistical goals, business rules.

1 Introduction

Business Rules (BR) are a "programming for non programmers" paradigm that is often used by large corporations to store industrial process knowledge formally. BR replaces the two most abstract concepts of programming, namely loops and function calls, by means of an implicit outer loop and meta-variables used within a set of easy-to-manage "if-then" type instructions. BR interpreters are implemented by all BR management systems, e.g. [14]. BR programs are often used by corporations to encode their policies and empirical knowledge: given some technical input, they produce a decision, often in the form of a YES/NO output. Corporations often require their internal processes to perform according to a prescribed statistical behavior, which could be imposed because of strategy or by law. This required behavior is typically independent of the BR input data. The problem is then to parametrize the BR program so it will behave as prescribed on average, while still providing meaningful YES/NO answers on given inputs.

In [30] we studied a simplified version of the problem where the statistical behavior was limited to a given mean. In this paper we provide a solution methodology for a more general (and difficult) case, where the statistical behavior is described by a given discrete distribution. We achieve this goal by encoding a Machine Learning (ML) procedure by means of a Mathematical Program (MP) of the Mixed-Integer Linear Programming (MILP) type. The ML procedure relies on non-input specific labels that encode the given knowledge about the distribution. Controlling the statistical behavior of a complex process such as a BR program is a very hard task, and to the best of our knowledge this work is the first of its kind in this respect. Methodologically speaking, we think our MILP formulation is also innovative in that it encodes an ML training process having labels which, instead of applying to individual inputs, apply to the entire input distribution at once. Such an ML process bypasses the usual difficulties of trying to label the training set data, thereby being more practical for industrial applications.

The motivation for this study is a real industrial need expressed by IBM (which co-funds this work) with respect to their BR package ODM. Our previous paper [30] laid some of the groundwork, limited to the most basic statistical indicator (the mean of a distribution). Though that was a necessary step to the current work, the methodology described herein is the first to actually address the need expressed by industry: we feel this is one of the main feature that sets this work apart from our previous work. We still rely on MILP-based methodology, but now the input is a whole discrete distribution, the cardinality of which largely determines the size of the new MILP formulations presented below. Our tests show that this has an acceptable impact on empirical solution complexity.

As an experimental illustration, we consider the two cases of the statistical behavior being (1) a maximum percentage of a certain output value and (2) the output values being distributed in a fashion close to the uniform distribution, for integer outputs. We provide an optimization based approach to solving the learning problem for each of those cases, then examine some test results.

1.1 Preliminaries

We formally represent a BR program as an ordered list of sentences of the form:

if $\mathsf{cond}(p, x)$ **then**
  $x \leftarrow \mathsf{act}(p, x)$
end if

where p is a *control parameter* vector (with c components) which encodes a possible "tuning" of the program (e.g. thresholds which can be adjusted by the user), $x \in X \subseteq \mathbb{R}^d$ is a *variable* vector representing intermediate and final stages of computation, cond is a boolean function, and act a function with values in X. We call *rule* such a sentence, *condition* an expression $\mathsf{cond}(p, x)$ and *action* an instruction $x \leftarrow \mathsf{act}(p, x)$, which indicates a modification of the value of x. We write the final value of the variable x as $x^f = P(p, q)$, where P represents the BR program and q is an *input parameter* vector representing a problem instance

and equal to the initial value of x. Although in general BR programs may have any type of output, we consider only integer outputs, since BR programs are mostly used to take discrete decisions. We remark that p, x are *symbolic vectors* (rather than numeric vectors) since their components are decision variables.

BR programs are executed in an external loop construct which is transparent to the user. Without getting into the details of BR semantics, the loop executes a single action from a BR whose condition is True at each iteration. Which BR is executed depends on a conflict resolution strategy with varying complexity. De Sainte-Marie et al. [23] describe typical operational semantics, including conflict resolution strategy, for industrial BR management systems. In this paper, the list of rules is ordered and the loop executes the first BR of the list with a condition evaluating to True at each iteration. The loop only terminates once every condition of the BRs is False. We proved in [29] that there is a universal BR program which can simulate any Turing Machine (TM), which makes the BR language Turing-complete.

We consider the problem where the $q \in Q$ are the past, known instances of the BR program, and the outputs $P(p, q)$ of those instances are divided into N evenly sized intervals $[H_0, H_1], \ldots, [H_{N-1}, H_N]$, forming a *quantized output distribution*. Denoting $\nu_1(p), \ldots, \nu_N(p)$ the number of outputs in these categories, we can formalize the problem as:

$$\left.\begin{array}{r} \min\limits_{p,x} \|p - p^0\|_1 \\ \mathscr{C}(\nu_1(p), \ldots, \nu_N(p)) \end{array}\right\} \tag{1}$$

where $\|\cdot\|_1$ is the L1 norm and $\mathscr{C}$ is a constraint or set of constraints. While this formulation uses the number of outputs rather than the probabilities themselves, the relation between the two is simply a ratio of $1/m$, where $m = \texttt{card}(Q)$ is the number of training data points.

In this paper, we suppose that P_1 and P_2 are BR programs with a rule set $\{\mathcal{R}_r \mid r \leq \rho\}$ containing rules of the form:

if $L_r \leq x \leq G_r$ **then**
  $x \leftarrow A_r x + B_r$
end if

with $L_r, G_r, B_r \in \mathbb{R}$ and $A_r \in \{0, 1\}^{d \times d}$. We note $R = \{1, \ldots, \rho\}$ and $D = \{1, \ldots, d\}$.

We discuss the concrete example of banks using a BR program in order to decide whether to grant a loan to a customer or not. The BR program depends on a variable vector x and initializes its parameter vector (a component of which is an income level threshold) to p^0. A BR program P_1 is used to decide whether a first bank will investigate the loan request further or simply accept the automated decision taken by an expert system, and therefore has a binary output value. This bank's high-level strategy requires that no more than 50% of loans are treated automatically, but P_1 currently treats 60%. Another bank instead uses a BR program $P2$ to accept, reject, or assign a bank manager to the loan request, and therefore has a ternary return value, represented by an integer in $\{0, 1, 2\}$. That bank's strategy requires that the proportion of each output is

$\{1/3, 1/3, 1/3\}$, but it is currently $\{1/4, 1/4, 1/2\}$. Our aim is in each case to adjust p, e.g. modifying the income level, so that the BR program satisfies the bank's goal regarding automatic loan treatment. This adjustment of parameters could be required after a change of internal or external conditions, for example.

The first scenario can be formulated as:

$$\left.\begin{array}{l} \min\limits_{p,x} \|p - p^0\|_1 \\ \mathbb{E}_{q \in Q}\big[P_1(p,q)\big] \leq g \end{array}\right\} \tag{2}$$

where P_1 has an output in $\{0, 1\}$, $g \in [0, 1]$ is the desired max percentage of 1 outputs, the $q \in Q$ are the past known instances of the BR program, $\|\cdot\|_1$ is the L1 norm, p, q must satisfy the semantics of the BR program $P(p, q)$ when executed within the loop of a BR interpreter and $\mathbb{E}$ is the usual notation for the expected value.

Similarly, the second scenario where P_2 has an output in $\{1, \ldots, N\}$ and the desired output is as close to a uniform distribution as possible can be formalized as:

$$\left.\begin{array}{l} \min\limits_{p,x} \|p - p^0\|_1 \\ \forall s, t \in \{1, \ldots, N\},\ \big|\nu_s - \nu_t\big| \leq 1 \end{array}\right\} \tag{3}$$

Note that the solution to this problem is not always a truly uniform distribution, simply because there is no guarantee that m is divisible by N. However, it will always be as close as possible to a uniform distribution, since the constraint imposes that all the outputs will be reached by either `floor`(m/N) or `ceil`(m/N) data points. Again, we use whole numbers (of outputs in a given interval) instead of frequencies to be able to employ integer decision variables.

Such problems could be solved heuristically by treating P_1 or P_2 as a black-box, or by replacing it by means of a simplified model, such as e.g. a low-degree polynomial. We approach this problem as in [30]: we model the algorithmic dynamics of the BR by means of MIP constraints, in view to solving those equations with an off-the-shelf solver. That this should be possible at all in full generality stems from the fact that Mathematical Programming (MP) is itself Turing-complete [16].

We make a number of simplifying assumptions in order to obtain a practically useful methodology, based on solving a Mixed-Integer Linear Programming (MILP) reformulation of these equations using a solver such as CPLEX [13]:

1. We suppose Q is small enough that solving the MILP is (relatively) computationally cheap.
2. We assume finite BR programs with a known bound $(n - 1)$ on the number of iterations of the loop for any input q (industrial BR programs often have a low value of n relative to the number of rules). This in turn implies that the values taken by x during the execution of the BR program are bounded. We assume that $M \gg 1$ is an upper bound of all absolute values of all p, q, and x, as well as any other values appearing in the BR program. It serves as a "big M" for the MP described in the rest of the paper.

3. We assume that the conditions and actions of the BR program give rise to constraints for which an exact MILP reformulation is possible. In order to have a linear model, each BR must thus be "linear", i.e. have the form:
   if $L \leq x \leq G$ **then**
   $x \leftarrow Ax + B$
   end if
   with $L, G, B \in \mathbb{R}^d$ and $A \in \{0,1\}^{d \times d}$. In general, $A_{h,k}$ may have values in $\mathbb{R}$ if it is not a parameter and x_h has only integer values.

1.2 Related Works

We follow the formalism used in [30] pertaining to Business Rules (BR) programs and their statistical behavior.

Business Rules (also known as *Production Rules*) are well studied as a knowledge representation system [8,10,18], originating as a psychological model of human behavior [20,21]. They have further been used to encode expert systems, such as MYCIN [6], [27], EMYCIN [6,25], OPS5 [5,11], or more recently ODM [14] or OpenRules [22]. On business side of things, they have been defined broadly and narrowly in many different ways [12,15,24]. We consider Business Rules as a computational tool, which to the best of our knowledge has not been explored in depth before.

Supervised Learning is also a well studied field of Machine Learning, with many different formulations [3,17,26,28]. A popular family of algorithms for the classification problem uses Association Rules [1,19]. Such Rule Learning is not to be confused with the problem treated in this article, which is more a regression problem than a classification problem. There exist many other algorithms for Machine Learning, from simple linear regression to neural networks [2] and support vector machines [9]. When the learner does not have as many known output values as it has items in the training set, the problem is known as Semi-Supervised Learning [7]. Similarly, there has been research into machine learning when the matching of the known outputs values to the inputs is not certain [4]. A previous paper has started to explore the Learning problem when the known information does not match to a single input [30].

2 Learning Goals with Histograms

In the rest of this paper, we concatenate indices so that $(L_r)_k = L_{rk}$, $(G_r)_k = G_{rk}$, $(A_r)_{h,k} = A_{rhk}$ and $(B_r)_k = B_{rk}$. We assume that rules are feasible, i.e. $\forall r, k \in R \times D, L_k \leq G_k$. In the rest of this section, we suppose that the dimension of p is $c = 1$, making p a scalar, and that p takes the place of A_{111}. Similar sets of constraints exists for when the parameter p takes the place of a scalar in B_r, L_r or G_r. Additional parameters correspond to additional constraints that mirror the ones used for the first parameter.

This formalization is taken from [30], in which we have also proved that the set of constraints described in Fig. 1 models the execution of such a BR

program. The iterations of the execution loop are indexed by $i \in I = \{1, \ldots, n\}$ where $n-1$ is the upper bound on the number of iterations, the final value of x corresponds to iteration n. We use an auxiliary binary variable y_{ir} with the property: $y_{ir} = 1$ iff the rule $\mathcal{R}_r$ is executed at iteration i. The other auxiliary binary variables y^U_{ir} and y^L_{ir} are used to enforce this property.

We note (C1), (C2), etc. the constraints related to the evolution of the execution and (IC1), (IC2), etc. the constraints related to the initial conditions of the BR program:

- (C1): represents the evolution of the value of the variable x
- (C2): represents the property that at most one rule is executed per iteration
- (C3): represents the fact that a rule whose condition is False cannot be executed
- (C4)-(C6) represent the fact that only the first rule whose condition is True can be executed
- (IC1) through (IC3) represent the initial value of a
- (IC4) represents the initial value of x.

$$\forall i \in I \backslash \{n\} \qquad x^{i+1} = \sum_{r \in R} (a_r x^i + B_r) y_{ir} + (1 - \sum_{r \in R} y_{ir}) x^i \qquad \text{(C1)}$$

$$\forall i \in I \qquad \sum_{r \in R} y_{ir} \leq 1 \qquad \text{(C2)}$$

$$\forall (i,r) \in I \times R \qquad L_r - M(1 - y_{ir})e \leq x^i \leq G_r + M(1 - y_{ir})e \qquad \text{(C3)}$$

$$\forall (i,r,k) \in I \times R \times D \qquad x^i_k \geq G_{rk} - M y^U_{irk} - M \sum_{r' < r} y_{ir'} \qquad \text{(C4)}$$

$$\forall (i,r,k) \in I \times R \times D \qquad x^i_k \leq L_{rk} + M y^L_{irk} + M \sum_{r' < r} y_{ir'} \qquad \text{(C5)}$$

$$\forall (i,r) \in I \times R \qquad 2d - 1 + y_{ir} \geq \sum_{k \in D} (y^U_{irk} + y^L_{irk}) \qquad \text{(C6)}$$

$$\forall r \in \{2, \ldots, \rho\} \qquad a_r = A_r \qquad \text{(IC1)}$$

$$a_{111} = p \qquad \text{(IC2)}$$

$$\forall (h,k) \in D^2 \setminus \{1,1\} \qquad a_{1hk} = A_{1hk} \qquad \text{(IC3)}$$

$$x^1 = q \qquad \text{(IC4)}$$

$$\forall i \in I \qquad x^i \in X$$

$$\forall r \in R \qquad a_r \in \{0,1\}^{d \times d}$$

$$\forall (i,r,k) \in I \times R \times D \qquad y_{ir}, y^U_{irk}, y^L_{irk} \in \{0,1\}$$

Fig. 1. Set of constraints modeling the execution of a BR program ($e \in \mathbb{R}^d$ is the all-one vector).

2.1 A MIP for learning quantized distributions

The Mixed-Integer Program from Fig. 2 models the problem from Eq. 1. We index the instances in Q with $j \in J = \{1, \ldots, m\}$. We also limit ourselves to solutions which result in computations that terminate in less than $n-1$ rule executions. As modifying the parameter means modifying the BR program, the assumptions made regarding the finiteness of the program might not be verified otherwise.

We note $O = \{1, \ldots, N\}$, such that $\forall t \in O, \nu_t = \texttt{card}\{j \in J \mid x^1_{n,j} \in [H_{t-1}, H_t]\}$. We enforce this definition of ν_t by using an auxiliary binary variable s_{tj} with the property: $s_{tj} = 1$ iff $x^1_{n,j} \in [B_{t-1}, B_t]$. The other auxiliary binary variables s^U_{tj} and s^L_{tj} are used to enforce this property.

The constraints are mostly similar to the ones in Fig. 1. We simply add the goal of minimizing the variation of the parameter value and the constraints $\mathscr{C}(\nu_1(p), \ldots, \nu_N(p))$ from Eq. 1. The new constraints are:

- (C7) represents the need for the computation to have terminated after $n-1$ executions
- (C8)-(C12) represents the definition of $\nu_1, \ldots, \nu_N$
- (IC4') represents (IC4) with an additional index j.

That solving the MIP in Fig. 2 also solves the original Eq. 1 is a direct consequence of the fact that the constraints in Fig. 1 simulate $P(p, q)$. The proof is simple since (C8) through (C12) trivially represent the definition of $\nu_1, \ldots, \nu_N$. A similar MIP can be obtained when p has values in different part of the BRs, from which a more complex MILP is obtained for when p is non-scalar. However, this formulation is still quite abstract, as it depends heavily on the form of $\mathscr{C}$. In fact, it can almost always be simplified given a particular constraint over the quantized distribution, as we see in the rest of this section.

2.2 A MILP for the Max Percentage Problem

A constraint programming formulation of Eq. 2 is the Mixed-Integer Linear Program (MILP) described in Fig. 3. In the case of the Max Percentage problem, we can linearize the MIP in Fig. 2 as well as remove some superfluous variables, since only one of the ν_t is relevant.

We now note $e = (1, \ldots, 1) \in \mathbb{R}^d$ the vector of all ones. We use the auxiliary variables $w \in \mathbb{R}^{I \times J \times R}$ and $z \in \mathbb{R}^{I \times J \times R \times D^2}$ such that $w_{ijr} = (A_r x^{ij} + B_r - x^{i,j}) y_{ijr}$ (i.e. $w_{ijr} = A_r x^{i,j} + B_r - x^{i,j}$, the difference between the new and the old values of x^j) and $z_{ijrhk} = a_{rhk} x^{i,j}_k$.

Any constraints numbered as before fulfills the same role. The additional constraints are:

- (C1'$_1$), (C1'$_2$), (C1'$_3$), (C1'$_4$) and (C1'$_5$) represent the linearization of (C1) from Fig. 1
- (C8') represents the goal from Eq. 2, that is a constraint over the average of the final values of x. It replaces $\mathscr{C}(\nu_1, \ldots, \nu_N)$ and all the constraints used to define ν_t from the MIP in Fig. 2.

$$\underset{p,a,x,y,y^U,y^L,s,s^U,s^L,\nu}{\text{minimize}} \quad |p^0 - p|$$

subject to

$$\text{(C1), (C2), (C3), (C4), (C5), (C6), (IC1), (IC2), (IC3)}$$

$$\mathscr{C}(\nu_1, \ldots, \nu_N)$$

$$\forall j \in J \qquad \sum_{r \in R} y_{njr} = 0 \qquad \text{(C7)}$$

$$\forall (t,j) \in O \times J \qquad H_{t-1} - M(1 - s_{tj}) \leq x_1^{n,j} \leq H_t + M(1 - s_{tj}) \qquad \text{(C8)}$$

$$\forall (t,j) \in O \times J \qquad x_1^{n,j} \geq H_t - M s_{tj}^U \qquad \text{(C9)}$$

$$\forall (t,j) \in O \times J \qquad x_1^{n,j} \leq H_{t-1} + M s_{tj}^L \qquad \text{(C10)}$$

$$\forall (t,j) \in O \times J \qquad s_{tj} \geq s_{tj}^U + s_{tj}^L \qquad \text{(C11)}$$

$$\forall t \in O \qquad \nu_t = \sum_{j \in J} s_{tj} \qquad \text{(C12)}$$

$$\forall j \in J \qquad x^{1,j} = q^j \qquad \text{(IC4')}$$

$$\forall (i,j) \in I \times J \qquad x^{i,j} \in X$$

$$\forall k \in R \qquad a_k \in \{0,1\}^{d \times d}$$

$$p \in \{0,1\}$$

$$\forall (i,j,r,k) \in I \times J \times R \times D \qquad y_{ijr}, y_{ijrk}^U, y_{ijrk}^L \in \{0,1\}$$

$$\forall (t,j) \in O \times J \qquad s_{tj}, s_{tj}^U, s_{tj}^L \in \{0,1\}$$

$$\forall t \in O \qquad \nu_t \in \mathbb{N}$$

Fig. 2. Mixed-Integer Program solving Eq. 1.

The MILP from Fig. 3 finds a value of p that satisfies Eq. 2. This is again derived from the fact that Fig. 1 simulates a BR program, and from the trivial proof that (C1'$_1$), (C1'$_2$), (C1'$_3$), (C1'$_4$) and (C1'$_5$) represent the linearization of (C1).

2.3 A MILP for the Almost Uniform Distribution Problem

As before, we exhibit in Fig. 4 a MILP that solves Eq. 3. Any constraints numbered as before fulfills the same role. The additional constraints are:

- (C8") through (C10") represent the adaptation of (C8) through (C10) to the relevant case of integer outputs
- (C13) represents the equivalent to $\mathscr{C}$ from Eq. 3.

This MILP is obviously equivalent to solving Eq. 3, since it is for the most part a straight linearization of the MIP in Fig. 2.

3 Implementation and Experiments

We use a Python script to randomly generate samples of 100 instances of P_1 and P_2 for different numbers of control parameters c, each instance having a

$$\underset{p,a,x,y,y^U,y^L,w,z}{\text{minimize}} \quad |p^0 - p|$$

subject to

(C2), (C3), (C4), (C5), (C6), (C7), (IC1), (IC2), (IC3)

$\forall (i,j) \in I \backslash \{n\} \times J$ $$x^{i+1,j} = \sum_{r \in R} w_{ijr} + x^{i,j} \quad \text{(C1'}_1\text{)}$$

$\forall (i,j) \in I \times J \times R$ $$-My_{ijr}e \leq w_{ijr} \leq My_{ijr}e \quad \text{(C1'}_2\text{)}$$

$\forall (i,j,r,h) \in I \times J \times R \times D$ $$\sum_{k \in D} z_{ijrhk} + B_{rh} - x_h^{i,j} - M(1 - y_{ijr}) \leq w_{ijrh} \leq \sum_{k \in D} z_{ijrhk} + B_{rh} - x^{i,j} + M(1 - y_{ijr})e \quad \text{(C1'}_3\text{)}$$

$\forall (i,j,r) \in I \times J \times R$ $$-Ma_r \leq z_{ijr} \leq Ma_r \quad \text{(C1'}_4\text{)}$$

$\forall (i,j,r,h,k) \in I \times J \times R \times D^2$ $$x_k^{i,j} - M(1 - a_{rhk}) \leq z_{ijhk} \leq x_k^{ij} \quad \text{(C1'}_5\text{)}$$

$\forall j \in J$ $$\sum_{r \in R} y_{njr} = 0 \quad \text{(C7)}$$

$$\sum_{j \in J} x_1^{n,j} \leq mg \quad \text{(C8')}$$

$\forall (i,j,r,k_1,k_2) \in I \times J \times R \times D^2$ $$x^{i,j}, z_{ijrhk} \in X$$

$\forall (i,j,r) \in I \times J \times R$ $$w_{ijr} \in \mathbb{R}^d$$

$\forall r \in R$ $$a_r \in \{0,1\}^{d \times d}$$

$$p \in \{0,1\}$$

$\forall (i,j,r,k) \in I \times J \times R \times D$ $$y_{ijr}, y_{ijrk}^U, y_{ijrk}^L \in \{0,1\}$$

Fig. 3. MILP formulation for solving Eq. 2.

corresponding set of inputs with $d = 3$, $n = 10$ and $m = 100$. The number of control parameters serves as an approximation of the complexity of the BR program to optimize: a more complex program will have more buttons to adjust, thus increasing the complexity, yet be more likely to have the goal be reachable at all, i.e. have the MILP be feasible. We define the space X as $X \subseteq \mathbb{R} \times \mathbb{R} \times \mathbb{Z}$. The BR programs are sets of $\rho = 10$ rules, where L_r, G_r, B_r are vectors of scalars in an interval `range` and A_r are $d \times d$ matrices of binary variables. In P_1, we use `range` $= [0, 1]$ and in P_2, we use `range` $= [0, 3]$. All input values q are generated using a uniform distribution in `range`.

We use these BR programs to study the computational properties of the MILP. The value of M used is customized according to each constraint, and is ultimately bounded by 6 and 16 in P_1 and P_2 respectively (strictly greater than five times the range of possible values for x). We write the MILP as an AMPL model, and solve it using the CPLEX solver on a Dell PowerEdge 860 running CentOS Linux.

$$\underset{p,b,x,y,y^U,y^L,w,s,s^L,s^g,\nu}{\text{minimize}} \quad |p^0 - p|$$

subject to

$$\text{(C1'}_1\text{)}, \text{(C1'}_2\text{)}, \text{(C1'}_3\text{)}, \text{(C1'}_4\text{)}, \text{(C1'}_5\text{)}, \text{(C2)}, \text{(C3)}, \text{(C4)}, \text{(C5)}, \text{(C6)}, \text{(C7)}, \text{(C11)}, \text{(C12)}, \text{(IC1)}, \text{(IC2)}, \text{(IC3)}, \text{(IC4')}$$

$$\forall (t,j) \in O \times J \qquad t - M(1 - s_{tj}) \leq x_1^{n,j} \leq t + M(1 - s_{tj}) \qquad \text{(C8")}$$

$$\forall (t,j) \in O \times J \qquad x_1^{n,j} \geq t - M s_{tj}^U \qquad \text{(C9")}$$

$$\forall (t,j) \in O \times J \qquad x_1^{n,j} \leq t + M s_{tj}^L \qquad \text{(C10")}$$

$$\forall (t,\tau) \in O^2 \qquad -1 \leq \nu_t - \nu_\tau \leq 1 \qquad \text{(C13)}$$

$$\forall (i,j) \in I \times J \qquad x^{i,j} \in X$$

$$\forall r \in R \qquad a_r \in \{0,1\}^{d \times d}$$

$$p \in \{0,1\}$$

$$\forall (i,j,r,k) \in I \times J \times R \times D \qquad y_{ijr}, y_{ijrk}^U, y_{ijrk}^L \in \{0,1\}$$

$$\forall (t,j) \in O \times J \qquad s_{tj}, s_{tj}^U, s_{tj}^L \in \{0,1\}$$

$$\forall t \in O \qquad \nu_t \in \mathbb{N}$$

Fig. 4. MILP formulation for solving Eq. 3.

3.1 The Max Percentage Problem

We observe the proportion of solvable instances of P_1 for c between 5 and 10 and $c = 15$ in Tab. 1. We use the MILP in Fig. 3 to solve Eq. 2 with the goal set to $g = 0.5$.

An instance is considered solvable if CPLEX reports an integer optimal solution or a (non-)integer optimal solution. We separate the instances where the optimal value is 0 from the others, as those indicate that the randomly generated BR program already fulfill the goal condition. We expect around fifty of those for any value of c.

In Fig. 5, we observe both the success rate and the average solving time when considering only the non-trivial, non-timed out instances of P_1. The success rate increases steadily, as expected. The solving time seems to indicate a non-linear increase for c greater than 6, even with its values being somewhat unreliable due to the small sample. Knowing that average industrial BRs are more complex than our toy examples, regularly having thousands of rules, this approach to the Maximum Percentage problem does not seem applicable to industrial cases.

3.2 The Almost Uniform Distribution Problem

We observe the proportion of solvable instances of P_2 for c between 5 and 10 and $c = 15$ in Tab. 2. We use the MILP in Fig. 4 to solve Eq. 3 with $N = 2$. Again,

Number of control parameters c	5	6	7	8	9	10	15
Trivial solvable instances (objective $= 0$)	52	53	49	49	58	48	46
Non-trivial solvable instances (objective $\neq 0$)	5	6	5	13	6	6	8
Infeasible instances	43	43	40	36	31	35	14
Timed out instances	0	0	7	2	5	11	32

Table 1. Experimental values for the maximum percentage problem.

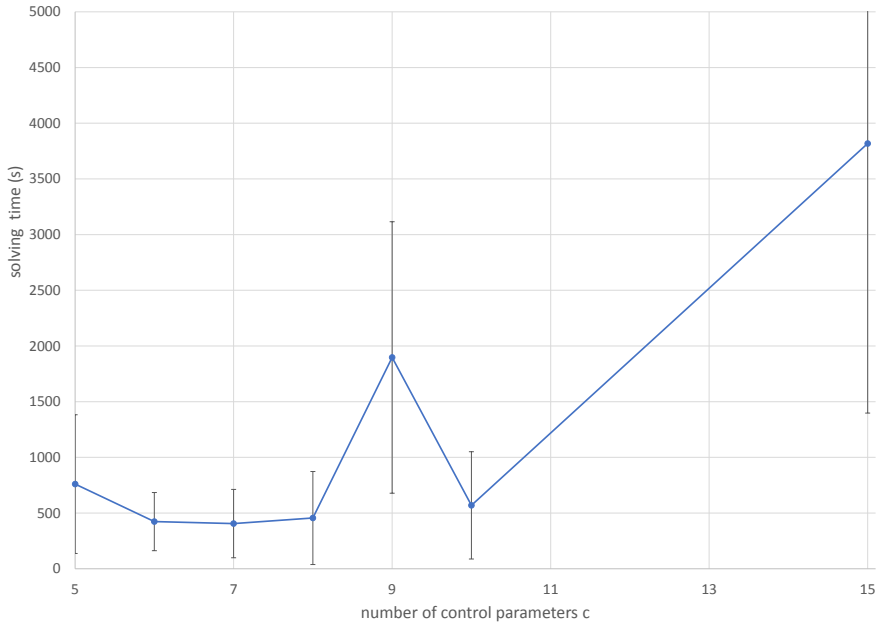

Fig. 5. Average solution time over P_1 for varying values of c in seconds.

we separate instances where the goal is already achieved before optimization, identifiable by being solved quickly with a value of $p = p^0$, i.e. an optimal value of zero.

In Fig. 6, we display the success rate and average solving time over the non-timed out, non-presolved instances for all three values of c. We observe a sharply non-linear progression, with the average problem taking about nine minutes with 15 control parameters. Knowing that average industrial BRs are much more complex than our toy examples, regularly having thousands of rules, we conclude that this method can only be used infrequently, if at all.

Number of control parameters c	5	6	7	8	9	10	15
Trivial solvable instances (objective = 0)	8	2	1	1	4	7	4
Non-trivial solvable instances (objective $\neq$ 0)	9	2	8	5	4	15	32
Infeasible instances	83	96	91	93	92	77	63
Timed out instances	0	0	0	1	0	1	1

Table 2. Experimental Values for the Almost Uniform Distribution problem.

4 Conclusion, Discussion and Future Work

We have presented a learning problem of unusual type, that of supervised learning with statistical labels. We have further explored a particular subset of those problems, those where the labels apply to a quantized output distribution. This new approach is easily applied to practical applications in industry where control parameters must be learned to satisfy a given goal. We have given a mathematical programming algorithm that solves such a learning problem given a linear BR program. Depending on the specific learning problem, the mathematical program might be easy or difficult to solve. We examined two example learning problems with practical applications for which the learning is equivalent to solving a MILP.

We observe that, though one could detect a visual similarity in the plots presented in Fig. 5 and 6, we believe that this similarity is only apparent. In fact, the error bars (which measure the standard deviation of the solution time over the instance subclass corresponding to a given size of parameters) point out that the "hard cases" (with high values of solution times) are also the cases where the error bars are longest. In other words, this "similarity" is simply a result of outliers in the corresponding peaks.

The experimental results indicate the general feasibility of this type of approach. It is clear that, due to the exponential nature of Branch-and-Bound

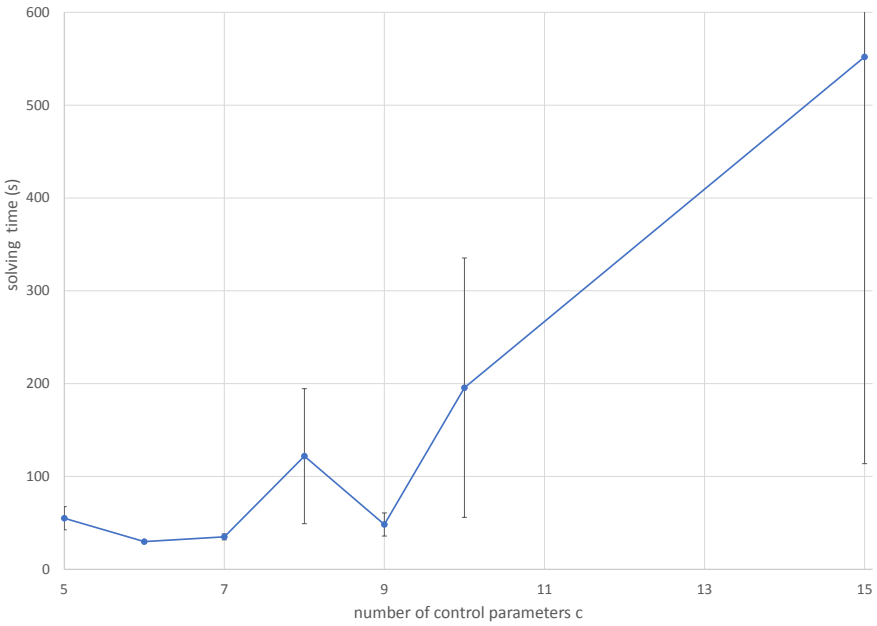

Fig. 6. Average solution time over non-trivial solvable P_2 for varying values of c.

(BB, the algorithm solving the MILPs), the performance will scale up poorly with the size of the BR program: but this can currently be said of most MILPs. This issue, which certainly requires more work, can possibly be tackled by pursuing some of the following ideas: more effective BB-based or formulation-based heuristics (also called *mat-heuristics* in the literature), cut generation based on problem structure, and decomposition. The latter, specifically, looks promising as the structure of the BR program is, up to the extent provided by automatic translation based on parsing trees, carried over to the resulting MILP.

Other avenues of research are in extending this statistical learning approach in other directions, e.g. learning other moments, or given quantiles in continuous distributions. Statistical goal learning problems are an apparently unexplored area of ML that has eminently practical applications.

Acknowledgments. The first author (OW) is supported by an IBM France/ANRT CIFRE Ph.D. thesis award.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Buneman, P., Jajodia, S. (eds.) Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. pp. 207–216. ACM, New York, NY (1993)
2. Atiya, A.: Learning Algorithms for Neural Networks. Ph.D. thesis, California Institute of Technology, Pasadena, CA (1991)
3. Bakir, G., Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., Vishwanathan, S.: Predicting Structured Data (Neural Information Processing). The MIT Press, Cambridge, MA (2007)
4. Brodley, C., Friedl, M.: Identifying mislabeled training data. Journal of Artificial Intelligence Research 11, 131–167 (1999)
5. Brownston, L., Farrell, R., Kant, E., Martin, N.: Programming Expert Systems in OPS5: An Introduction to Rule-based Programming. Addison-Wesley, Boston, MA (1985)
6. Buchanan, B., Shortliffe, E. (eds.): Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project (The Addison-Wesley Series in Artificial Intelligence). Addison-Wesley, Boston, MA (1984)
7. Chapelle, O., Schlkopf, B., Zien, A.: Semi-Supervised Learning. The MIT Press, Cambridge, MA (2010)
8. Clancey, W.: The epistemology of a rule-based expert system: a framework for explanation. Artificial Intelligence 20(3), 215–251 (1983)
9. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (1995)
10. Davis, R., Buchanan, B., Shortliffe, E.: Production rules as a representation for a knowledge-based consultation program. Artificial Intelligence 8(1), 15–45 (1977)
11. Forgy, C.: OPS5 User's Manual. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (1981)
12. G. Knolmayer, H.H.: Business rules. Wirtschaftsinformatik 35(4), 386–390 (1993)
13. IBM: ILOG CPLEX 12.2 User's Manual. IBM (2010)

14. IBM: Operational Decision Manager 8.8 (2015)
15. Kolber, A., et al.: Defining business rules - what are they really? Project Report 3, The Business Rules Group (2000)
16. Liberti, L., Marinelli, F.: Mathematical programming: Turing completeness and applications to software analysis. Journal of Combinatorial Optimization 28(1), 82–104 (2014)
17. Liu, T.Y.: Learning to rank for information retrieval. Foundations and Trends in Information Retrieval 3(3), 225–331 (2009)
18. Lucas, P., Gaag, L.V.D.: Principles of Expert Systems. Addison-Wesley, Boston, MA (1991)
19. Malioutov, D.M., Varshney, K.R.: Exact rule learning via boolean compressed sensing. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML '13). JMLR: Workshop and Conference Proceedings, vol. 28, pp. 765–773. JMLR, Brookline, MA (2013)
20. Newell, A.: Production systems: Models of control structures. In: Chase, W. (ed.) Visual Information Processing. Proceedings of the Eighth Annual Carnegie Symposium on Cognition. pp. 463–526. Academic Press, New York, NY (1973)
21. Newell, A., Simon, H.: Human Problem Solving. Prentice-Hall, Upper Saddle River, NJ (1972)
22. OpenRules, Inc., Monroe, NJ: OpenRules User Manual (2015)
23. Paschke, A., Hallmark, G., De Sainte Marie, C.: RIF production rule dialect (second edition). W3C recommendation, W3C (2013), http://www.w3.org/TR/2013/REC-rif-prd-20130205/
24. Ross, R.: Principles of the Business Rule Approach. Addison-Wesley, Boston, MA (2003)
25. Scott, A., Bennett, J., Peairs, M.: The EMYCIN Manual. Department of Computer Science, Stanford University, Stanford, CA (1981)
26. Settles, B.: Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin-Madison (2009)
27. Shortcliffe, E.: Computer-Based Medical Consultations: MYCIN. Elsevier, New York, NY (1976)
28. Vapnik, V.: The Nature of Statistical Learning Theory. Springer-Verlag, New York, NY (1995)
29. Wang, O., Ke, C., Liberti, L., de Sainte Marie, C.: The learnability of business rules. In: International Workshop on Machine Learning, Optimization, and Big Data (MOD 2016) (2016)
30. Wang, O., Liberti, L., D'Ambrosio, C., De Sainte Marie, C., Ke, C.: Controlling the average behaviour of business rules programs. In: Alferes, J., Bertossi, L., Governatori, G., Fodor, P., Roman, D. (eds.) Rule Technologies. Research, Tools, and Applications (RuleML2016). LNCS, vol. 9718, pp. 83–96. Springer, Berlin, Germany (2016)