Controlling the Average Behavior of Business Rules Programs

Olivier Wang[1,2], Leo Liberti[2(✉)], Claudia D'Ambrosio[2], Christian de Sainte Marie[1], and Changhai Ke[1]

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

Abstract. Business Rules are a programming paradigm for non-programmer business users. They are designed to encode empirical knowledge of a business unit by means of "if-then" constructs. The classic example is that of a bank deciding whether to open a line of credit to a customer, depending on how the customer answers a list of questions. These questions are formulated by bank managers on the basis of the bank strategy and their own experience. Banks often have goals about target percentages of allowed loans. A natural question then arises: can the Business Rules be changed so as to meet that target *on average*? We tackle the question using "machine learning constrained" mathematical programs, which we solve using standard off-the-shelf solvers. We then generalize this to arbitrary decision problems.

1 Introduction

For the purpose of this work, a *Business Rule* (BR) program is 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* symbol vector 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* symbol vector of dimension d 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. If P is the BR program, we write the final value of the variable x as $x^f = P(p, q)$, where q is an *input parameter* symbol vector representing a problem instance and equal to the initial value of x. Although in general BR programs may have any type of output, many BR programs encode decision problems, in which case the part of the output that matters can be represented by a single bit (one component of x is a binary variable).

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

J.J. Alferes et al. (Eds.): RuleML 2016, LNCS 9718, pp. 83–96, 2016.
DOI: 10.1007/978-3-319-42019-6_6

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. [22] 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 it satisfies a termination condition, which we assume to be that none of the conditions of the BRs is True at the last iteration (as is usual). We proved in [27] that there is a universal BR program which can simulate any Turing Machine (TM), which makes the BR language Turing-complete.

The BR language is useful as a "programming tool for non-programmers", since it hides the two aspects of imperative computer programming which most non-programmers find confusing: loops and function calls. As mentioned above, BR programs only have a single loop, which is part of the interpreter, and external to the language itself. The BR language replaces function calls (up to a point) by factorizing many code fragments into a single 'rule'. The BR interpreter instantiates each rule into as many code fragments as possible by matching all consistent variable types at compile time.

The BR language is often used in medium-to-large sized corporations to encode their policies and empirical knowledge – often easily representable as "if-then" type statements. Such business processes are often embedded in a database of BRs representing a mix of regulations, organizational policies and operational knowledge. The latter can be collected from multiple employees over a possibly long period of time. BR interpreters are implemented by all BR management systems, e.g. [14].

The aim of this paper is to describe a method for changing the control parameters of BR programs as little as possible so that they approximately meet a given "average behavior" goal. This issue arises in the following setting:

- the BR program $P(p, q)$ encoding the business process has a (non-empty) control parameter symbol vector p;
- the BR program is run using the parameter vector p^0;
- the corporation owning the business process has an average goal to meet on a function f, with values in $\mathbb{R}$, of the outcomes of the BR program;
- the average of $P(p^0, q)$ where q ranges over a (possibly large but finite) set Q of instances is different from the goal.

We discuss the concrete example of a bank 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 the minimum income level) to p^0. The BR program is used to decide whether the bank will reject the loan request, and therefore has a binary return value. Assume that the bank high-level strategy requires that no more than 40 % of loans should be rejected automatically, and that the BR program currently rejects about 60 %. Our aim is to adjust p, e.g. modifying the income level, so that the BR program satisfies the bank's goal regarding automatic loan rejection. This adjustment of

parameters could be required after a change of internal or external conditions, for example.

Let $g \in \mathbb{R}$ be the desired goal. Then the problem can be formalized as:

$$\left.\begin{array}{c} \min\limits_{p,x} \|p - p^0\| \\ \left| \mathbb{E}_{q \in Q}\left[f\left(P(p,q)\right)\right] - g \right| \leq \varepsilon, \end{array}\right\} \tag{1}$$

where $\|\cdot\|$ is a given norm, p, q must satisfy the semantics of the BR program $P(p,q)$ when executed within the loop of a BR interpreter, $\mathbb{E}$ is the usual notation for the expected value and ε is a given tolerance. This formalization is closer to the reality of BR users than the reverse (minimizing $\mathbb{E}(P) - g$ while constraining $p - p^0$), as corporations will often consider goals more rigidly than changes to the business process, and the value of the objective will speak to them more as a kind of quantification of the changes to be made. The form this quantification takes, from minimizing the variation of each parameter in p to minimizing the number of parameters whose value is modified, depends on the definition of the norm $\|\cdot\|$. By using a linearizable norm, such as $\|\cdot\|_1$ or $\|\cdot\|_\infty$, we can solve Eq. (1) for linear BR programs using MILP solvers, through a pre-processing reformulation. While this problem looks like a supervised learning problem at first glance, standard supervised learning algorithms cannot help here as there is no 'correct' answer for each separate instance q. Rather, a global approach is necessary as, in the general case, the correct classifier is defined as a frequency distribution over the set of all instances Q. In this paper we consider the simplified case where the expected value serves as the classifier, which is equivalent to having the frequency distribution in the common case of a binary output.

Traditionally, this problem would be solved heuristically by treating P as a black-box, or by replacing it by means of a simplified model, such as e.g. a low-degree polynomial. Our approach is different: we model the algorithmic dynamics of P by means of Mixed-Integer Programming (MIP) constraints, in view to solving Eq. (1) with an off-the-shelf solver. That this is at all possible in full generality follows because 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 Eq. (1) using a solver such as CPLEX [13] or BonMin [3]:

1. We replace Q by a smaller "training set" S for which we know the BR outcome. We choose S small enough that solving the MILP is (relatively) computationally cheap.
2. We assume a finite BR program 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 $A \in \mathbb{R}^{d \times d}$. We see in Sect. 3 that an actual MILP actually requires $A \in \{0,1\}^{d \times d}$ in some cases.

We shall attempt to relax some or all of these assumptions in later works.

We also remark that this setting easily generalizes to any class of decision problems depending on a "tuning" parameter p, for which an average behavior is prescribed.

For the rest of the paper, we make the following simplifying assumptions (all of which afford no loss of generality).

1. We assume that the dimension of p is one, making it a scalar. Consequently, we choose the norm in Eq. 1 to be the absolute value for the rest of the paper. Additional parameters correspond to additional constraints that mirror the ones used for the first parameter.
2. We assume that the relevant function of the outcome f is the projection on the first dimension: $f(x) = x_1$. Any linear f can be used instead with no difference in the constraints, but BR programs usually have a projection of the variable x as their output.

1.1 Related Works

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 [19,20]. They have further been used to encode expert systems, such as MYCIN [6,25], EMYCIN [6,23], OPS5 [5,11], or more recently ODM [14] or OpenRules. On the business side of things, they have been defined broadly and narrowly in many different ways [12,15,21]. 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 [2,17,24,26]. There exist many algorithms for this problem, from simple linear regression to neural networks [1] 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]. However, the fact that each known value corresponds to a single input item has not been questioned before, to the best of our knowledge.

2 MIP Constraints for the BR Program Dynamics

We study a BR program 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 rule $\mathcal{R}_1$ being instead:

if $L_1 \leq x \leq G_1$ **then**
$x \leftarrow A_1^p x + B_1$
end if

where A_1^p is a $d \times d$ matrix satisfying:

$$\begin{cases} \forall k_1, k_2 \in D, k_1 \neq 1 \vee k_2 \neq 1 \Rightarrow (A_1^p)_{k_1,k_2} = (A_1)_{k_1,k_2} \\ (A_1^p)_{1,1} = p \end{cases}$$

with $D = \{1, \ldots, d\}$.

In the rest of this paper, we concatenate indices so that $(L_r)_k = L_{r,k}$, $(G_r)_k = G_{r,k}$, $(A_r)_{k_1,k_2} = A_{r,k_1,k_2}$ and $(B_r)_k = B_{r,k}$. We assume that rules are meaningful, such that $L_k \leq G_k$.

2.1 Modeling a BR Program

We exhibit a set of MIP constraints (Fig. 1) modeling the execution of the 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. The rules are indexed by $r \in R = \{1, \ldots, \rho\}$. We use an auxiliary binary variable $y_{i,r}$ with the property: $y_{i,r} = 1$ iff the rule $\mathcal{R}_r$ is executed at iteration i. The vectors of binary variables $y_{i,r}^g$ and $y_{i,r}^l$ are used to enforce this property. In the rest of this section, the parameter is assumed to take the place of $A_{1,1,1}$, so we note a an additional variable initialized to $a = A$ except for $a_{1,1,1} = p$. Similar sets of constraints exists for when the parameter p takes the place of a scalar in B_r, L_r or G_r.

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) through (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

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

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

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

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

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

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

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

$$a_{1,1,1} = p \qquad \text{(IC2)}$$

$$\forall (k_1, k_2) \in D^2 \setminus \{1,1\} \qquad a_{1,k_1,k_2} = A_{1,k_1,k_2} \qquad \text{(IC3)}$$

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

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

$$\forall r \in R \qquad a_r \in \mathbb{R}^{d \times d}$$

$$\forall (i,r) \in I \times R \qquad y_{i,r}, y^g_{i,r,k}, y^l_{i,r,k} \in \{0,1\}$$

Fig. 1. Set of constraints modeling the execution of a BR program with $e = (1, \ldots, 1) \in \mathbb{R}^d$ a vector of all ones

Theorem 1. *The MIP constraints from Fig. 1 correctly model the execution of the BR program with input* (p, q)*. The value of* x^n *after applying the constraints is then the output of the BR program:* $x^n = P(p, q)$*.*

Proof. We begin by proving that for a given $i \in I$, it is true that $y_{i,r} = 1$ iff x^i fulfills the condition for rule $\mathcal{R}_r$ and does not fulfill the condition for any rule $\mathcal{R}_{r'}$ where $r' < r$. Suppose $y_{i,r} = 1$. (C3) $\Rightarrow L_r \leq x^i \leq G_r$ implies that x^i fulfills the condition for rule $\mathcal{R}_r$. Let us now set $r' < r$.

$$\text{C2} \Rightarrow y_{i,r'} = 0 \wedge \sum_{r'' < r'} y_{i,r''} = 0$$

$$\text{C6} \Rightarrow \exists k \in D : y^g_{i,r',k} = 0 \vee y^l_{i,r',k} = 0$$

As we also have:

$$y^g_{i,r',k} = 0 \wedge \text{C4} \Rightarrow x^i_k \geq G_{r',k}$$

$$y^l_{i,r',k} = 0 \wedge \text{C5} \Rightarrow x^i_k \leq L_{r',k}$$

We have one of $x^i_k \geq G_{r',k}$ or $x^i_k \leq L_{r',k}$. Either of those means that x^i does not fulfill the condition for rule $\mathcal{R}_{r'}$.

Conversely, suppose that x^i fulfills the condition for rule $\mathcal{R}_r$ and does not fulfill the condition for any rule $\mathcal{R}_{r'}$ where $r' < r$. Reasoning by induction over r', we see that assuming $\sum_{r'' < r'} y_{i,r''} = 0$ (which is true for $r' = 1$) we have:

$$C4 \wedge C5 \wedge C6 \Rightarrow y_{i,r'} = 0$$

because the condition for $\mathcal{R}_{r'}$ is not fulfilled. We thus have $\sum_{r'<r} y_{i,r'} = 0$. This and the fact that the condition for $\mathcal{R}_r$ is fulfilled means that $y_{i,r} = 1$.

A simple inductive proof over the $i \in I$ then proves that the x^i are the successive values taken by x during the execution of the BR program as long as $\sum_{r \in R} y_{i,r} = 1$ and that the value of x^i does not change as long as $\sum_{r \in R} y_{i,r} = 0$, which corresponds to the stopped execution of the BR program. This also proves $x^n = P(p, q)$. □

2.2 A MIP Formulation

Having modeled the dynamics of a single execution of the BR program by means of the constraints of the previous section, we now come back to our original purpose: we exhibit a MIP that finds a value of p satisfying Eq. 1 in Fig. 2.

We index the instances in S with $j \in J = \{1, \ldots, m\}$, where $m = |S|$ is the number of instances in the training set S. The parameter p is now one of the variables. We note $e = (1, \ldots, 1) \in \mathbb{R}^d$ the vector of all ones.

As modifying the parameter means modifying the BR program, the assumptions made regarding the finiteness of the program might not be verified when optimizing over p. One of those which might lead to unusable solutions is the assumption that the computations terminate in less than $n-1$ iterations. In the case where the MIP finds a value of p for which the BR program is stopped by this limit on the loop rather than by the proper termination condition, the MIP

$$\begin{array}{lll}
\underset{p,a,x,y,y^g,y^l}{\text{minimize}} & |p^0 - p| & \\
\text{subject to} & & \\
 & \text{(C1), (C2), (C3), (C4), (C5), (C6), (IC1), (IC2), (IC3)} & \\
\forall j \in J & \sum_{r \in R} y_{n,j,r} = 0 & \text{(C7)} \\
 & \left| \frac{1}{m} \sum_{j \in J} x_1^{n,j} - g \right| \leq \varepsilon & \text{(C8)} \\
\forall j \in J & x^{1,j} = q^j & \text{(IC4')} \\
\forall (i,j) \in I \times J & x^{i,j} \in X & \\
\forall k \in R & a_k \in \mathbb{R}^{d \times d} & \\
 & p \in \mathbb{R} & \\
\forall (i,j,r,k) \in I \times J \times R \times D & y_{i,j,r}, y^g_{i,j,r,k}, y^l_{i,j,r,k} \in \{0,1\} &
\end{array}$$

Fig. 2. MIP Formulation for Solving Eq. 1 with $e = (1, \ldots, 1) \in \mathbb{R}^d$ a vector of all ones

would not actually solve Eq. 1. We therefore limit ourselves to solutions which result in computations that terminate in less than $n-1$ rule executions.

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

- (C7) represents the need for the computation to have terminated after $n-1$ executions
- (C8) represents the goal from Eq. 1, that is a constraint over the average of the final values of x.

Theorem 2. *The MIP from Fig. 2 finds a value of p that satisfies Eq. 1.*

The proof derives directly from Theorem 1.

3 A MILP Reformulation

The problem as written in Eq. 2 is not linear. A linear reformulation exists for when the parameter p takes the place of a scalar in B_r, L_r or G_r. Figure 3 describes such a MILP when p takes the place of $B_{1,1}$. We linearize the products of $A_r x^{i,j}+b_r$ by $y_{i,j,r}$ and $x^{i,j}$ by $y_{i,j,r}$ in (C1) using factorization and an auxiliary variable $w \in \mathbb{R}^{I\times J\times R}$. We arrange to have $w_{i,j,r} = (A_r x^{i,j} + b_r - x^{i,j})y_{i,j,r}$, i.e. $w_{i,j,r} = A_r x^{i,j} + b_r - x^{i,j}$ (the difference between the new and the old values of x^j) iff rule r is executed, and 0 otherwise.

Theorem 3. *This MILP finds a value of p that satisfies Eq. 1, when p takes the place of $B_{1,1}$. A similar MILP exists for when p takes the place of another scalar in $B_{r,k}$, $L_{r,k}$ and $G_{r,k}$.*

The proof derives directly from Theorem 2, by factoring constraint (C1) in Fig. 2 and studying the possible values of $y_{i,j,k}$.

When the parameter takes the place of $A_{1,1,1}$, a linear formulation is only possible if $A_{1,1,1}$ is a discrete variable. For the purpose of this article, we only use the case where A_{1,k_1,k_2} are binary variables. The associated MILP is in Fig. 4. In that case, we have the additional product of ax to linearize, so we use another auxiliary variable $z \in \mathbb{R}^{I\times J\times R\times D^2}$ such that $z_{i,j,r,k_1,k_2} = a_{r,k_1,k_2} x_{i,j,k_2}$.

Theorem 4. *The MILP in Fig. 4 finds a value of p that satisfies Eq. 1, when p takes the place of $A_{1,1,1}$ and A_{1,k_1,k_2} are binary variables.*

The proof derives from Theorem 3 and a study of the possible values of A_{1,k_1,k_2} and $y_{i,j,r}$. We can trivially expand the MILP to optimize over more than one parameter, adding constraints similar to constraints (IC1), (IC2) and (IC3) or (IC1"), (IC2") and (IC3") in Fig. 1 or Fig. 3 as necessary and having an objective of $\sum_p \|p^0 - p\|$.

$$\underset{p,b,x,y,y^g,y^l,w}{\text{minimize}} \quad |p^0 - p|$$

subject to

$$\text{(C2), (C3), (C4), (C5), (C6), (C7), (IC4')}$$

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

$$\forall (i,j) \in I\times J\times R \qquad -My_{i,j,r}e \le w_{i,j,r} \le My_{i,j,r}e \qquad (\text{C1"}_2)$$

$$\forall (i,j,r) \in I\times J\times R \qquad A_r x^{i,j} + b_r - x^{i,j} - M(1-y_{i,j,r})e \le w_{i,j,r} \le A_r x^{i,j} + b_r - x^{i,j} \qquad (\text{C1"}_3)$$

$$-\varepsilon \le \frac{1}{m}\sum_{j\in J} x_1^{n,j} - g \le \varepsilon \qquad (\text{C8"})$$

$$\forall r \in \{2,\dots,\rho\} \qquad b_r = B_r \qquad (\text{IC1"})$$

$$b_{1,1} = p \qquad (\text{IC2"})$$

$$\forall k \in \{2,\dots,d\} \qquad b_{1,k} = B_{1,k} \qquad (\text{IC3"})$$

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

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

$$p \in \mathbb{R}$$

$$\forall (i,j,r,k) \in I\times J\times R\times D \qquad y_{i,j,r}, y^g_{i,j,r,k}, y^l_{i,j,r,k} \in \{0,1\}$$

Fig. 3. MILP Formulation with p Taking the Place of $B_{1,1}$ with $e = (1,\dots,1) \in \mathbb{R}^d$ a vector of all ones

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

subject to

$$\text{(C1"}_1\text{), (C1"}_2\text{), (C2), (C3), (C4), (C5), (C6)}$$
$$\text{(C7), (C8"), (IC1), (IC2), (IC3), (IC4')}$$

$$\forall (i,j,r,k_1) \in I\times J\times R\times D \qquad \sum_{k_2\in D} z_{i,j,r,k_1,k_2} + B_r - x^{i,j} - M(1-y_{i,j,r})e \le w_{i,j,r} \le \sum_{k_2\in D} z_{i,j,r,k_1,k_2} + B_r - x^{i,j} + M(1-y_{i,j,r})e \qquad (\text{C1}_3^{(3)})$$

$$\forall (i,j,r) \in I\times J\times R \qquad -Ma_r \le z_{i,j,r} \le Ma_r \qquad (\text{C1}_4^{(3)})$$

$$\forall (i,j,r,k_1,k_2) \in I\times J\times R\times D^2 \qquad x_{i,j,k_2} - M(1-a_{r,k_1,k_2}) \le z_{i,j,k_1,k_2} \le x_{i,j,k_2} \qquad (\text{C1}_5^{(3)})$$

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

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

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

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

$$\forall (i,j,r,k) \in I\times J\times R\times D \qquad y_{i,j,r}, y^g_{i,j,r,k}, y^l_{i,j,r,k} \in \{0,1\}$$

Fig. 4. MILP Formulation with p Taking the Place of $A_{1,1,1}$ with $e = (1,\dots,1) \in \mathbb{R}^d$ a vector of all ones

Table 1. Experimental values for the scalability of the MILP method

Value of ρ	Proportion of instances solvable in an hour	Average solver times over solvable instances	Average objective values over solvable instances
1	1.0	2.0877	0.9772
2	0.98	22.9848	2.1363
3	0.96	265.3536	4.0421
4	0.89	737.6687	6.9834
5	0.66	929.3174	7.771

4 Implementation and Experiments

We use a Python script to randomly generate samples of 100 BR programs and corresponding sets of instances with $d = 3$, $n = 10$ and $m = 100$. We define the space X as $X \subseteq \mathbb{R} \times \mathbb{R} \times \mathbb{Z}$. The BR programs are sets of a variable number ρ of rules of the type:

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

where L_r, G_r, B_r are vectors of scalars in $[-5, 5]$; $L_r \leq G_r$ and A_r are $d \times d$ matrices of binary variables. The instances are vectors q_j with values in $[-5, 5]$. All values are generated using a uniform distribution. We use a variable value of ε. For each BR program, we try to obtain a goal $g = 0$ by optimizing over $\phi = 5\rho$ randomly chosen parameters.

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 51 (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.

4.1 Scalability

We set a fixed ε value of 1. This corresponds to a very high tolerance (20 % of the range of possible values). We observe the average solving time and optimal objective for different values of ρ (Table 1) among the solvable MILP instances. An instance is considered unsolvable if it is infeasible or it has no integer solution after 1 h (3600 s) of solver time.

While it could be argued that the increase in the number of parameters has an obvious effect over the difficulty of the problem, the study of an increase in ρ without the proportional increase in ϕ leads to a drastic and predictable increase in infeasible instances. Even with our setup, the proportion of solvable instances is lower as ρ increases, although that is mostly due to the solver exceeding the time limit. As a high value of ρ is the main issue when scaling up to industrial BR programs, it still seems worth studying.

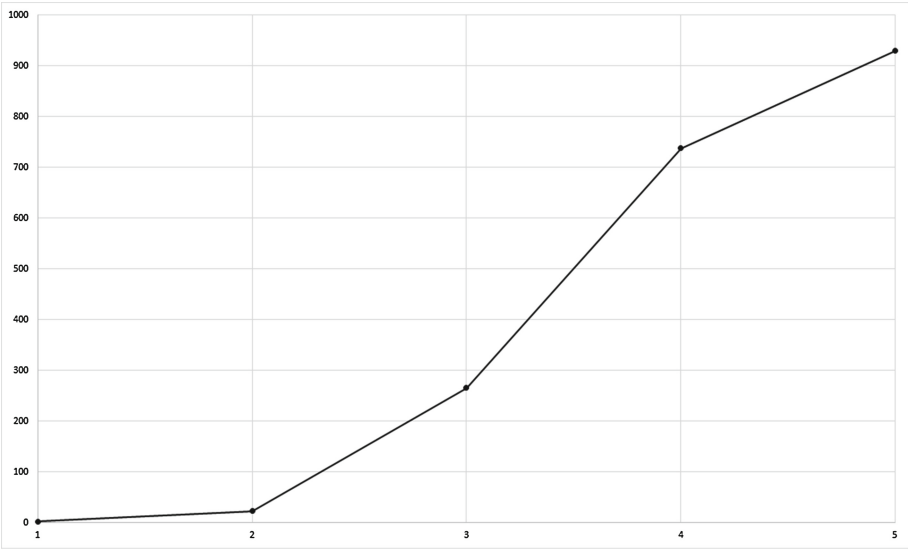

Fig. 5. Variation of computational time (in seconds) with ρ (the number of rules in the BR program)

Unfortunately, we observe that the direct solving of the MILP described in Sect. 3 is not practical for learning parameters in industrial-sized BR programs. Furthermore, the increase in computational time is not linear with the number of ρ, but rather exponential as seen on Fig. 5. The increase in the optimal objective value is intuitive and does not seem drastic, which indicates that the experimental setup is somewhat realistic. A reformulation of the MILP to improve the solving time of the MILP is a possible follow-up area of research.

Another possibility we plan on researching is to check whether using sparse matrices for L, G, A and B significantly improves the computational experience: most industrial business processes do not use complex rules, but rather many rules each applying to one or two components of the variable x. This does not immediately imply a better performance as the value of n would realistically need to be increased to compensate.

4.2 Accuracy

We fix the number of rules to $\rho = 4$ and so the number of parameters is 20. This is much lower than the number of rules used by BR programs destined to industrial usage. We observe the average solving time and the proportion of solvable MILP instances among all instances for different values of ε. In this case, it means we have only generated one hundred BR programs, on which we test the accuracy of our method.

For each BR program, we start with $\varepsilon = 1$. If the instance is solvable, we decrease ε using $\varepsilon \leftarrow \varepsilon - 0.2$ until we reach an unsolvable instance; otherwise we

increase ε using $\varepsilon \leftarrow 1.5\varepsilon$ until we reach a solvable instance or $\varepsilon \geq 5$, whichever happens first.

An instance is considered unsolvable if it is infeasible or it has no integer solution after fifteen minutes (900 s) of solver time.

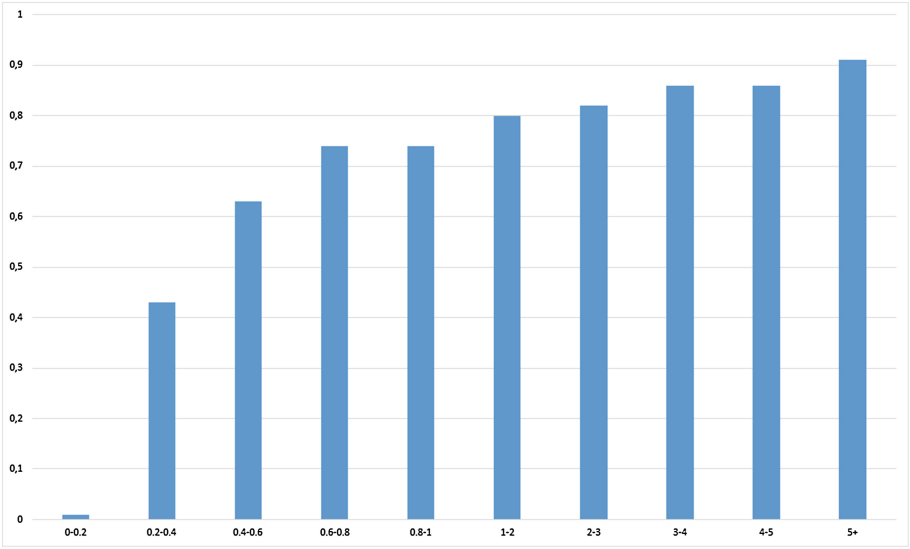

Fig. 6. Proportion of solvable instances for varying values of ε

Figure 6 shows the proportion of solvable instances as a function of ε. Being careful of the nonlinear scale of the figure, ε seems to have a greater influence on solvability when small and a reduced influence as it grows. In other words, our method will relatively easily find the best parameter in most cases, but difficult problems remain difficult even when allowing for a greater distance to the desired average f.

Furthermore, with random BR problems that include both unsolvable and already optimal situations, we solve 50 percent of problems with $\varepsilon = 0.4$ which means allowing $\mathbb{E}(f) \in [-0.4, 0.4]$. This is too low for industrial applications, but approaching the desired average by 8 percent of the possible range in half the cases is a promising start. Our method as it is described cannot currently be used as a general tool as too many BR programs cannot be solved accurately.

The restriction to $n = 10$ for those orders of ρ accurately models real business processes where BRs rarely loop. The sample size of $m = 100$ is also somewhat realistic. The dimension $d = 3$ is arbitrary and much lower than can be expected in actual business processes, where dimensionality can be over an order of magnitude higher.

5 Conclusion, Discussion and Future Work

We have presented a way to use mathematical programming to model learning problems of an unusual type: learning the parameters of a BR program so that its average output over a set of instances Q meets a target g. This can be extended to learning the parameters of any program where the semantics of an execution are well-defined, to reach a targeted average output. In such a program, the semantics lead directly to mathematical constraints defining a MIP, with the execution being modeled by indexing the variables over the steps of the computation. Depending on the program, solving can then be easy, if the program is linear like in our case, or lead to more complex optimization methods. Where standard supervised learning algorithms are difficult to apply with a target defined on average, our method can help fill the blanks.

While the computational performance indicates that directly solving the MILP formulation described in this article is impractical, it does hold promise. In particular, the optimal values obtained when scaling up prove that if the computational cost can be reduced, the methodology has possible industrial implications. The potential avenues of research in this direction are working on the MILP itself (e.g. through reformulations) and experimenting on BR programs closer to the reality of business processes (e.g. through sparse matrices).

A better methodology for choosing the best possible ε might also be needed, as the current one only yields a broad estimate. A possible avenue of research pertaining to the accuracy of our method is evaluating the risk of over-fitting, through the generation of additional samples $q \in Q \backslash S$ and using a parameter-less version of the MILP in Fig. 3 to evaluate the average $\frac{1}{m} \sum_{j \in J} x_1^{n,j}$.

As many real-life applications use rules with a linear structure, our model has direct applications in many industries that rely on BR programs to automate decisions, some of which might not even need additional refinements depending on the size of the BR programs they use.

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

References

1. Atiya, A.: Learning algorithms for neural networks. Ph.D. thesis, California Institute of Technology, Pasadena, CA (1991)
2. Bakir, G., Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., Vishwanathan, S.: Predicting Structured Data (Neural Information Processing). The MIT Press, Cambridge (2007)
3. Bonami, P., Lee, J.: BONMIN user's manual. Technical report, IBM Corporation, June, 2007
4. Brodley, C., Friedl, M.: Identifying mislabeled training data. J. Artif. Intell. Res. **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 Longman Publishing Co., Boston (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 Longman Publishing Co., Boston (1984)
7. Chapelle, O., Schlkopf, B., Zien, A.: Semi-Supervised Learning. The MIT Press, Cambridge (2010)
8. Clancey, W.: The epistemology of a rule-based expert system: a framework for explanation. Artif. Intell. **20**(3), 215–251 (1983)
9. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
10. Davis, R., Buchanan, B., Shortliffe, E.: Production rules as a representation for a knowledge-based consultation program. Artif. Intell. **8**(1), 15–45 (1977)
11. Forgy, C.: OPS5 User's Manual. Department of Computer Science, Carnegie-Mellon University, Pittsburgh (1981)
12. Knolmayer, G., Herbst, H.: Business rules. Wirtschaftsinformatik **35**(4), 386–390 (1993)
13. IBM: ILOG CPLEX 12.2 User's Manual. IBM, New York (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. J. Comb. Optim. **28**(1), 82–104 (2014)
17. Liu, T.Y.: Learning to rank for information retrieval. Found. Trends Inf. Retr. **3**(3), 225–331 (2009)
18. Lucas, P., Gaag, L.V.D.: Principles of Expert Systems. Addison-Wesley Longman Publishing Co., Boston (1991)
19. Newell, A.: Production systems: models of control structures. In: Chase, W. (ed.) Visual Information Processing, pp. 463–526. Academic Press, New York (1973)
20. Newell, A., Simon, H.: Human Problem Solving. Prentice-Hall, Upper Saddle River (1972)
21. Ross, R.: Principles of the Business Rule Approach. Addison-Wesley Longman Publishing Co., Boston (2003)
22. de Sainte Marie, C., Hallmark, G., Paschke, A.: RIF Production Rule Dialect. 2nd edn. Recommendation, W3C (2013)
23. Scott, A., Bennett, J., Peairs, M.: The EMYCIN Manual. Department of Computer Science, Stanford University, Stanford (1981)
24. Settles, B.: Active learning literature survey. Computer Sciences Technical report 1648, University of Wisconsin-Madison (2009)
25. Shortcliffe, E.: Computer-Based Medical Consultations: MYCIN. Elsevier, New York (1976)
26. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
27. 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)