One-class synthesis of constraints for Mixed-Integer Linear Programming with C4.5 decision trees

Patryk Kudła, Tomasz P. Pawlak*

Institute of Computing Science, Poznan University of Technology, Poznań, Poland

Abstract

We propose a Constraint Synthesis with C4.5 (CSC4.5) method for automated construction of constraints for Mixed-Integer Linear Programming (MILP) models. Given a sample of snapshots of feasible states of a modeled entity, e.g., a business process or a system, CSC4.5 synthesizes a well-formed MILP model of that entity, suitable for simulation and optimization using an off-the-shelf solver. CSC4.5 works by modeling distribution of feasible states, bounding that distribution with C4.5 decision tree and transforming that tree into a set of MILP constraints. We verify CSC4.5 experimentally on a suite of parameterized synthetic benchmarks, concluding considerable fidelity of the synthesized constraints to the actual constraints in the benchmarks. Next, we apply CSC4.5 in a case study of modeling a real-world business process of wine production, optimize models of this process and validate the optimal solutions with use of a competing method.

Keywords: model acquisition, constraint synthesis, mathematical programming, one-class classification, business process

1. Introduction

Optimization of business processes and systems is an essential part of every business, as reducing consumed resources or increasing volume of production may bring substantial advantage in a competitive business environment. To optimize a business process, one needs to build a mathematical model of this process and optimize this model instead of manipulating directly the parameters of the process. This allows the process to operate uninterruptedly until the optimal parameters are found. For instance, a clothing company may reduce fabric loss by rearranging pieces of fabric to be cut from a fabric roll such that the loss is minimal. To do this, one needs to build a model resembling shapes

*Corresponding author
Email address: tpawlak@cs.put.poznan.pl (Tomasz P. Pawlak)
URL: http://www.cs.put.poznan.pl/tpawlak/ (Tomasz P. Pawlak)

of pieces to cut and technical restrictions of the cutting machine, and optimize this model using an off-the-shelf solver. The optimal cutting pattern may be applied during e.g., maintenance break.

In this study, we employ Mixed-Integer Linear Programming (MILP, Williams [31]) models. A well-formed MILP model consists of linear objective function, set of linear constraints and domains of variables thereof. Linearity facilitates interpretation of the objective function and the constraints while being enough expressive to model many real-world processes. The constraints in a MILP model essentially form a conjunction, however thanks to the support of binary variables, one can form alternative of constraints by introducing auxiliary binary variables to the model. Integer variables cause solving of a MILP model to be NP-hard (Karp [17]), however modern solvers can efficiently handle thousands of integer variables.

MILP models are often built manually by an expert, requiring thus from the expert competencies in the modeled process and techniques of modeling. Combining these competencies is uncommon in practice. Also, reflecting non-linear real-world relationships using linear equations require advanced modeling techniques. All these issues cause model building to be error-prone, laborious and expensive task. In contrast, optimization of the MILP model is fully automated by solvers.

An appealing alternative to manual model building is synthesis of model from data. The model synthesis problem can be conveniently decomposed into synthesis of an objective function and synthesis of constraints. The former can be done using e.g., least-squares regression which is out of the scope of this study, and the latter is investigated.

We propose a *Constraint Synthesis with C4.5* (CSC4.5, Sect. 4) method for synthesis of MILP constraints from examples of snapshots of feasible states of the modeled process. Feasible states represent normal operating conditions of the process and can be conveniently acquired by observing process execution. Observation can sometimes spot an infeasible state too – an error, a fault or other undesirable state. Infeasible states, however, are avoided in practice and thus uncommon. This is the reason, why we assume their absence and consider *one-class* examples only (like one-class classification in Machine Learning, Khan and Madden [18]). CSC4.5 copes with one-class examples by estimating distribution of feasible states using Expectation Maximization (Dempster et al. [11]) and sampling this distribution for artificial likely infeasible states that act as a second class. A training set made of the feasible and the likely infeasible examples is fed to C4.5 decision tree learning algorithm (Quinlan [26]). Finally, the resulting decision tree is transformed into a well-formed MILP model. CSC4.5 supports alternative of constraints by automatically introducing auxiliary binary variables.

We experimentally tune parameters of CSC4.5 using synthetic benchmarks and then evaluate properties of the synthesized models (Sect. 5). The general conclusions are promising, however leave room for improvements. Next, we apply CSC4.5 to synthesize a MILP model for a real-world process of wine production and optimize the resulting model using an off-the-shelf solver (Sect. 6).

The comparison of this model to a model constructed by a naive algorithm based on calculating convex hull of a training set reveals superiority of CSC4.5 in terms of consumed computational resources and higher estimated quality of the optimal solution.

2. Problem statement

In this section, we formalize constraint synthesis problem, and in Section 4 we propose an algorithm that solves this problem.

Let $x_1, x_2, \ldots, x_n \in \mathbb{R}$ be input variables, $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$ be an example of input variables' values, X be a set of examples, and $b_1, b_2, \ldots, b_m \in \{0, 1\}$ be auxiliary variables, $\boldsymbol{b} = [b_1, b_2, \ldots, b_m]$, i.e., $\boldsymbol{b}$ is not a part of an example. Then, constraint $c(\boldsymbol{x}, \boldsymbol{b})$ is a function in form of $\sum_{k=1}^{n} w_k x_k + \sum_{k=1}^{m} v_k b_k \leq a$, where $w_k, v_k \in \mathbb{R}$ are weights and $a \in \mathbb{R}$ is a constant. The aim of *constraint synthesis problem* is to find set of constraints C such that $\forall_{\boldsymbol{x} \in X} \forall_{c \in C} \exists_{\boldsymbol{b}}\, c(\boldsymbol{x}, \boldsymbol{b}) = \text{true}$.

The constraint synthesis problem is ill-posed, as it has indefinitely many solutions, including a degenerated solution $C = \emptyset$, i.e., an empty set of constraints satisfied for every $\boldsymbol{x}$. The choice of a particular solution depends thus on user's preferences, technical requirements and/or other factors. In this study, we prefer C which individual constraints $c \in C$ form facets of hypercubes in $\mathbb{R}^n$, such that the hypercubes correspond to alternative subsets of constraints, and Euclidean distance of every facet to the closest $\boldsymbol{x} \in X$ is minimal. A feasible region of C in $\mathbb{R}^n$ is thus a union of hypercubes, modeled by assigning distinct values for $\boldsymbol{b}$ for each hypercube.

Albeit not explicitly stated, examples in X correspond to feasible states of a modeled process, and as such delineate location of the feasible region of C. We call them *feasible examples* to distinguish from an unlabeled examples that are introduced later in this paper.

The constraint synthesis problem is a kind of one-class classification problem (Khan and Madden [18]), where the aim is to learn characteristics of the the given examples instead of learning boundary between examples of different classes.

Although the set of constraints C is a part of a MILP model, synthesis of other parts of the model is beyond the scope of this study and we use terms 'set of constraints' and 'model' interchangeably.

3. Related work

The works on constraint synthesis can be classified w.r.t. two axes: type of the constraints: Linear Programming (LP), Non-Linear Programming (NLP), Constraint Programming (CP) and other types of constraints, and type of the synthesis problem: one-class, where only feasible examples are available like in the problem posed in Section 2, and two-class, where feasible and infeasible examples are available.

Regarding LP models, Pawlak and Krawiec [25] proposed GenetiCS method that uses strongly-typed Genetic Programming (Montana [22]) to tackle two-class synthesis problems. Given considerably successful results, Pawlak and Krawiec [24] proposed GOCCS, an extension of GenetiCS, that supports one-class synthesis problem. GOCCS represents sets of constraints as abstract syntax trees (ASTs) that allow synthesis of LP and NLP models. GOCCS is verified on a suite of synthetic benchmarks and in modeling of a real-world business process of wine production with good results. GOCCS, however, suffers from *curse of dimensionality* (Bellman [4]) and quality of the synthesized models drops rapidly for over half a dozen variables.

Pawlak and Krawiec [23] proposed also to transform two-class synthesis problem of LP-like models to Mixed-Integer Linear Programming (MILP) problem and solve it optimally w.r.t. a custom measure of model complexity. Although, this method is exact, it becomes computationally infeasible if number of problem variables is over a dozen due to NP-hardness of MILP problem.

Aswal and Prasanna [1] propose to handle one-class synthesis problem by building a convex hull of a set of feasible examples, cluster facets of this hull using k-means (Lloyd [21]) and write them down as LP constraints. Unfortunately, complexity of this method is exponential in the number of input variables.

Concerning models expressed using CP paradigm, Bessiere et al. [6] propose Conacq system that uses feasible and infeasible examples and version-space learning to build a CP model. Conacq's advantages are support for alternative of constraints and patterns of constraints (structure of constraints repeated for different variables), however it is limited to finite-domain variables. Bessiere et al. [5] propose also QuAcq system that synthesizes constraints based on interactive queries of an expert for classification of artificial examples and supports incomplete data (i.e., examples without some variables set).

Other representations of constraints include Prolog clauses, e.g., generated by Model Seeker system (Beldiceanu and Simonis [3]). Model Seeker employs a hand-crafted library of abstract constraints annotated with metadata and builds from them concrete constraints separating feasible and infeasible examples. Model Seeker may be unable to solve problems requiring constraints of types not included in the library.

Teso et al. [30] propose Learning Modulo Theories (LMT) framework that builds constraints as first-order logic clauses using background knowledge constraints and feasible examples. The main disadvantage of this system is that background knowledge must be provided by an expert in form of logic predicates, thus LMT only slightly lowers modeling effort of the expert.

A related field of study is estimation of hypervolume, i.e., a boundary that wraps a set of examples. It may be expressed in a way of inequalities (like e.g., LP constraints), convex hull of a set of points, or threshold on probability density function of an estimated empirical probability distribution. The last approach is taken by Blonder et al. [7] that employ multivariate kernel density estimation (Simonoff [27]) for that purpose. This approach can be possibly extended to synthesize Stochastic Programming models, however the authors do not verify this.

As outlined in Introduction, CSC4.5 employs C4.5 decision tree induction algorithm by Quinlan [26]. C4.5 requires at least two classes of examples, otherwise would build degenerated tree of a single node. In CSC4.5, C4.5 is fed by the given feasible examples and artificially generated second class examples using routine described in Section 4.1. This allows us to use C4.5 without modifications. However, there are other approaches, like e.g., POSC4.5 by Denis et al. [12] that extends C4.5 to support feasible and unlabeled examples by using a statistically-backed technique to detect possible classes of the unlabeled examples. One-class Very Fast Decision Tree (OcVFDT) by Li et al. [20] is a hybrid of POSC4.5 and Very Fast Decision Tree (VFDT) by Domingos and Hulten [13] that shares support of unlabeled examples of POSC4.5 with speed of VFDT. All these algorithms require considerable number of unlabeled examples in the training set, however these are not available for the constraint synthesis problem as posed in Section 2.

An alternative approach is to employ Support Vector Data Description (SVDD) by Tax [29]. SVDD is one-class derivative of Support Vector Machine by Cortes and Vapnik [9] that encloses feasible examples in an ellipse or if used together with kernel functions, in an arbitrary other closed shape. Hence, SVDD corresponds to a single quadratic or higher-order function and can be used to synthesize NLP models, however for higher-order or non-convex functions these models would be NP-hard to solve.

4. Synthesis of Mixed-Integer Linear Programming constraints

Algorithm 1 shows *Constraint Synthesis with C4.5* (CSC4.5). It divides into three main steps that we discuss in detail in the following subsections:

1. **Training set preparation**: Distribution of the feasible examples is estimated based on X and likely infeasible examples are sampled using this distribution to form a two-class *training set* for C4.5.
2. **Decision tree learning**: Decision tree is built by C4.5 algorithm using the above-prepared training set.
3. **Decision tree to MILP model transformation**: Branches of the learned decision tree corresponding to feasible decision are transformed into MILP constraints.

4.1. Training set preparation

We model distribution of feasible examples in X using Gaussian Mixture Model (GMM), i.e., a weighted sum of g multivariate Gaussian distributions, each having its own mean vector and covariance matrix. We assume that number of Gaussians g is user-provided parameter, while the remaining parameters, denoted by $\boldsymbol{\theta}$, are to be estimated. Although the larger g the better GMM should fit actual distribution of X, the exact formula for g is not trivial and may be problem-dependent. For that reason, we verify experimentally few such formulas in Section 5.2.

Algorithm 1 CSC4.5 algorithm. X is set of feasible examples, h is maximum tree height, ρ is assumed percentile of outliers in X. ExpectationMaximization(X) estimates parameters of Gaussian Mixture Model of X, Sequence returns next natural number on each call, beginning from 1, M is a big constant.

function CSC4.5(X, h, ρ)
  $U \leftarrow$ SampleUnlikelyStates(X, ρ)
  $D \leftarrow$ C4.5($X \cup U, h$)
  $C \leftarrow$ Transform($D, 0$)
  return C

function SampleUnlikelyStates(X, ρ)
  $\boldsymbol{\theta} \leftarrow$ ExpectationMaximization(X) ▷ Estimate parameters of GMM
  $t \leftarrow \rho$th percentile of $\{\mathrm{p}(\boldsymbol{x}|\boldsymbol{\theta}) : \boldsymbol{x} \in X\}$ ▷ Low probability threshold
  $U \leftarrow \{\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\theta}) | p(\boldsymbol{x}|\boldsymbol{\theta}) < t\} \wedge |U| = |X|n^2$ ▷ Low probable states
  return U

function C4.5(T, h)
  if all $\boldsymbol{x} \in T$ have class z **then return** (z) ▷ Return decision node
  if $h = 1$ **then return** (z) where z is majority class in T
  $(x_i, a) \leftarrow \underset{(x_i,a), i=1..n, a \in \{x_i \in \boldsymbol{x} \in T\}}{\arg\max} GR(T, x_i, a)$ ▷ Select best split
  $left \leftarrow$ C4.5($\{\boldsymbol{x} \in T | x_i \le a\}, h-1$) ▷ Recursive call for left subtree
  $right \leftarrow$ C4.5($\{\boldsymbol{x} \in T | x_i > a\}, h-1$) ▷ Recursive call for right subtree
  return $\overset{x_i \le a}{left} \swarrow (x_i, a) \searrow \overset{x_i > a}{right}$ ▷ Return decision tree

function Transform($(x_i, a), v(\boldsymbol{b})$)
  $C \leftarrow \emptyset$
  if both children of (x_i, a) have feasible decision in their substrees **then**
    $k \leftarrow$ Sequence
    $C \leftarrow C \cup \{x_i \le a + v(\boldsymbol{b}) + M(1 - b_k), x_i \ge a - v(\boldsymbol{b}) - Mb_k\}$
    $C \leftarrow C \cup$ Transform($left, v(\boldsymbol{b}) + M(1 - b_k)$)
    $C \leftarrow C \cup$ Transform($right, v(\boldsymbol{b}) + Mb_k$)
  else if $left$ child of (x_i, a) has feasible decision in its subtree **then**
    $C \leftarrow C \cup \{x_i \le a + v(\boldsymbol{b})\}$
    $C \leftarrow C \cup$ Transform($left, v(\boldsymbol{b})$)
  else if $right$ child of (x_i, a) has feasible decision in its subtree **then**
    $C \leftarrow C \cup \{x_i \ge a - v(\boldsymbol{b})\}$
    $C \leftarrow C \cup$ Transform($right, v(\boldsymbol{b})$)
  return C

We estimate $\boldsymbol{\theta}$ in line 7 of Algorithm 1 using Expectation-Maximization (EM) algorithm by Dempster et al. [11] and its implementation by Souza [28]. EM finds maximum log-likelihood parameters of GMM by repeating two steps until increase of log-likelihood is less than a convergence threshold:

- **Expectation step**: Calculate the expectation of log-likelihood function $Q(\boldsymbol{\theta}|\boldsymbol{\theta}_i) = E_{Z|X,\boldsymbol{\theta}_i}[\log p(X, Z|\boldsymbol{\theta})]$, where Z is a set of unobserved data of latent variables, $\boldsymbol{\theta}_i$ is current estimate of $\boldsymbol{\theta}$, and p is probability density function of GMM,
- **Maximization step**: Find the parameters maximizing that function: $\boldsymbol{\theta}_{i+1} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}_i)$.

Next, we sample GMM for unlabeled examples having low value of p, i.e., likely infeasible. In line 8, we calculate threshold t on probability density function $p(\boldsymbol{x}|\boldsymbol{\theta})$ of GMM as ρth percentile of values of p for all feasible examples $\boldsymbol{x} \in X$, where parameter $\rho \approx 1\%$ is assumed percentile of outliers in X. In line 9, we sample GMM for set U of $|X|n^2$ unlabeled examples $\boldsymbol{x}$ having $p(\boldsymbol{x}|\boldsymbol{\theta}) < t$, where n is the number of variables (cf. Sect. 2). We relate cardinality of U to n^2 to alleviate exponentially increasing sparsity of spatial distribution of U with growing n. We use n^2, as storing exponential number of unlabeled examples would be computationally infeasible even for over a dozen n.

Sets X and U are involved in learning decision tree in the next section, where X and U act as sets of examples of different classes.

4.2. C4.5 decision tree learning

We build decision tree using C4.5 algorithm by Quinlan [26] and its implementation by Souza [28]. Algorithm 1 in lines $11-17$ shows pseudocode of C4.5. It builds decision tree in top-down manner recursively repeating the following steps. Given training set T that we initially set to $X \cup U$ in line 3, C4.5 returns a decision node with class z if all examples in T have class z (line 12) or maximum tree height h is reached and z is majority class in T (line 13). Otherwise, in line 14 C4.5 selects variable x_i and its value a in T maximizing Gain Ratio:

$$GR(T, x_i, a) = \frac{IG(T, x_i, a)}{IV(T, x_i)} \qquad \text{Gain Ratio}$$

$$IG(T, x_i, a) = H(T) - \frac{|T_{x_i \le a}|}{|T|} H(T_{x_i \le a}) - \frac{|T_{x_i > a}|}{|T|} H(T_{x_i > a}) \qquad \text{Information Gain}$$

$$IV(T, x_i, a) = -\frac{|T_{x_i \le a}|}{|T|} \log_2 \frac{|T_{x_i \le a}|}{|T|} - \frac{|T_{x_i > a}|}{|T|} \log_2 \frac{|T_{x_i > a}|}{|T|} \qquad \text{Intrinsic Value}$$

$$H(T) = -\frac{|T \cap X|}{|T|} \log_2 \frac{|T \cap X|}{|T|} - \frac{|T \cap U|}{|T|} \log_2 \frac{|T \cap U|}{|T|} \qquad \text{Entropy}$$

$$T_{x_i \le a} = \{\boldsymbol{x} \in T | x_i \le a\}$$

$$T_{x_i > a} = \{\boldsymbol{x} \in T | x_i > a\}$$

x_i and a serve as a split point that splits T into subsets $T_{x_i \leq a}$ and $T_{x_i > a}$. Gain ratio measures ratio of information gain resultant from splitting T on (x_i, a), i.e., decrease in amount of information needed to describe all examples from T thanks to that split, to intrinsic value of that split, i.e., amount of information required to describe that split itself. Note an implementation detail that we omit from Algorithm 1 for readability: variables x_i joining the path from the root node to the current node max join j times are not selected for split and if all variables $x_i, i = 1..n$ join j times then a decision node with the majority class in T is returned.

Next, C4.5 calls itself recursively for $T_{x_i \leq a}$ and $T_{x_i > a}$ (lines $15 - 16$) to build $left$ and $right$ subtrees, respectively. Finally, in line 17 $left$ and $right$ subtrees are joined using a node carrying out the split on (x_i, a) and the newly created tree is returned.

Note that the above description is simplified version of the actual C4.5 algorithm limited to support real variables only, as only those present in the constraint synthesis problem (cf. Sect. 2).

4.3. Transformation to Mixed-Integer Linear Programming model

Finally, we transform a decision tree into MILP constraints. A decision tree in the essence is a hierarchy of constraints, where particular constraints are active (i.e., operational) only if all constraints higher in the tree are satisfied. This hierarchy is not directly representable in a MILP model, as MILP constraints are independent. To preserve semantics of the decision tree in the MILP constraints, we extend the constraints corresponding to tree branches with weighted sum of auxiliary binary variables $b_1, b_2, \ldots, b_m$ that encode this hierarchy. Also, strict inequalities ($>$) are transformed into soft ones ($\geq$), as MILP constraints support soft inequalities only.

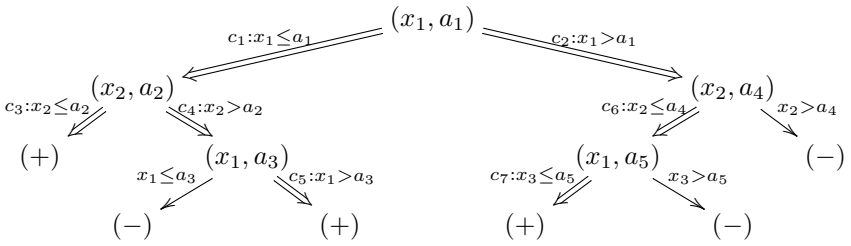
Function TRANSFORM in lines $18 - 31$ of Algorithm 1 recursively follows all paths from the root node to the leaf nodes having feasible decision, collects constraints in the visited branches and extends them with the weighted sum of auxiliary variables (cf. definition of constraint in Sect. 2). $v(\boldsymbol{b})$ argument of TRANSFORM represents that sum: $v(\boldsymbol{b}) = \sum_{k=1}^{m} v_k b_k + v_0$. Initially $v(\boldsymbol{b}) = 0$ (line 4) and we employ it with non-zero weights v_k and constant v_0 during visiting particular tree nodes. $v(\boldsymbol{b})$ is added to every produced constraint.

Given node (x_i, a), TRANSFORM switches between three cases:

1. If both children of (x_i, a) have feasible decision in their subtrees (line 20), it introduces new auxiliary variable b_k (line 21), collects two constraints corresponding to the left and right descendant branches and extends them with $M(1-b_k)$ and Mb_k, respectively, where M is a big constant $M \to \infty$ (line 22). For $b_k = 1$, the left branch constraint $x_i \leq a + M(1 - b_k)$ is active, as $M(1 - b_k) = 0$ and the right branch constraint $x_i \geq a - Mb_k$ is inactive, as equivalent to $x_i > -\infty$. In turn, for $b_k = 0$ the former is equivalent to $x_i < \infty$ and thus inactive, and the latter is equivalent to $x_i \geq a$ and active. TRANSFORM continues recursively for left and right child node with $v(\boldsymbol{b})$ updated with $M(1-b_k)$ and Mb_k, respectively (lines $23 - 24$).

2. If only the left child of (x_i, a) has feasible decision in its subtree (line 25), TRANSFORM collects a constraint corresponding to the left branch (line 26) and recursively continues for the child node (line 27).
3. If only the right child of (x_i, a) has feasible decision in its subtree (line 28), the corresponding steps occur in lines 29 – 30.

Example 1. Consider a decision tree of variables x_1 and x_2:



TRANSFORM follows all paths from the root node to the leaf nodes with feasible decision (denoted by double arrows). For the root node (x_1, a_1), it collects constraints c_1 and c_2 and as both subtrees contain feasible decision, auxiliary variable b_1 is introduced to control which of c_1 and c_2 is active:

$$c_1 : x_1 \leq a_1 \longrightarrow x_1 \leq a_1 \quad \underbrace{+M(1-b_1)}_{c_1 \text{ is active if } b_1=1}$$

$$c_2 : x_1 > a_1 \longrightarrow x_1 \geq a_1 \quad \underbrace{-Mb_1}_{c_2 \text{ is active if } b_1=0}$$

Then, TRANSFORM advances to left child node (x_2, a_2) with $v(\boldsymbol{b}) = M(1-b_1)$. It collects constraints c_3 and c_4 and extends them with $v(\boldsymbol{b})$ and newly introduced auxiliary variable b_2:

$$c_3 : x_2 \leq a_2 \longrightarrow x_2 \leq a_2 \underbrace{+M(1-b_1)}_{c_1 \text{ is active}} \underbrace{+M(1-b_2)}_{c_3 \text{ is active if } b_2=1}$$

$$c_4 : x_2 > a_2 \longrightarrow x_2 \geq a_2 \underbrace{-M(1-b_1)}_{c_1 \text{ is active}} \underbrace{-Mb_2}_{c_4 \text{ is active if } b_2=0}$$

Next, TRANSFORM goes to right child node (x_1, a_3) with $v(\boldsymbol{b}) = M(1-b_1)+Mb_2$. (x_1, a_3) has feasible decision in the right subtree only, thus a single constraint is collected and extended with $v(\boldsymbol{b})$:

$$c_5 : x_1 > a_3 \longrightarrow x_1 \geq a_3 \underbrace{-M(1-b_1)}_{c_1 \text{ is active}} \underbrace{-Mb_2}_{c_4 \text{ is active}}$$

Similar steps for the right child of the root node result in constraints c_6 and c_7:

$$c_6 : x_2 \leq a_4 \longrightarrow x_2 \leq a_4 \quad \underbrace{+Mb_1}_{c_2 \text{ is active}}$$

$$c_7 : x_3 \leq a_5 \longrightarrow x_3 \leq a_5 \quad \underbrace{+Mb_1}_{c_2 \text{ is active}}$$

Table 1: Benchmarks formulated as MP models, d is a parameter used to bound variables' values, set to 5.

Ball$n$	Cube$n$
$d^2 \geq \sum_{i=1}^{n}(x_i - i)^2$	$\forall_{i=1}^{n} : x_i \geq i$
$\forall_{i=1}^{n} : x_i \in [i - 2d, i + 2d]$	$\forall_{i=1}^{n} : x_i \leq i + id$
	$\forall_{i=1}^{n} : x_i \in [i - id, i + 2id]$

Simplex$n$

$$\forall_{i=1}^{n}\forall_{k=i+1}^{n} : x_i \cot\tfrac{\pi}{12} - x_k \tan\tfrac{\pi}{12} \geq 0$$
$$\forall_{i=1}^{n}\forall_{k=i+1}^{n} : x_k \cot\tfrac{\pi}{12} - x_i \tan\tfrac{\pi}{12} \geq 0$$
$$\sum_{i=1}^{n} x_i \leq d$$
$$\forall_{i=1}^{n} : x_i \in [-1, 2 + d]$$

5. Experiment

5.1. Setup

We conduct three experiments. In the first one, we tune number of Gaussians in GMM(cf. Sect. 4.1). In the second experiment, we tune parameters of C4.5 algorithm (cf. Sect. 4.2). Finally, we verify how size of a training set influences performance of CSC4.5 running with parameters found in the previous two experiments.

In all experiments, parameters of EM and C4.5 not mentioned in text are set to their defaults in Souza [28]. All experimental setups run 30 times with different random seeds to obtain statistically-sound conclusions.

In all experiments, we use three benchmarks parameterized by number of variables n and presented as MP models in Table 1. Sets of feasible examples X are acquired by uniformly sampling feasible regions of the benchmarks and are different for each random seed. Cardinality of X is fixed to 500 in the first two experiments and varies in $\{100, 200, 300, 400, 500\}$ in the last experiment. A separate test set of 100,000 examples drawn uniformly from Cartesian Product of variables' domains for each random seed is used to calculate statistics. By design no outliers and no statistical noise occur in these sets, thus we fix percentile of outliers ρ to 0% throughout experiments. Different characteristics of every benchmark enables us to verify CSC4.5's performance in problems of different requirements: Ball$n$ requires one quadratic constraint of n variables, Cube$n$ needs $2n$ linear constraints of one variable each, and Simplex$n$ involves $n(n-1)$ linear constraints of two variables each and one linear constraint of n variables.

We assess synthesized models using the following measures:

Jaccard index [16] $\frac{TP}{TP+FP+FN}$, where TP refers to True Positives, FP to False Positives and FN to False Negatives, all calculated using test-set. Jaccard index measures overlapping of feasible regions of the synthesized and actual models and attains values in range from 0 (disjoint regions) to 1 (equal regions).

Precision [14] $\frac{TP}{TP+FP}$ is a probability that a point from feasible region of the synthesized model belongs to the feasible region of the actual model.

Recall [14] $\frac{TP}{TP+FN}$ is a probability that a point from the feasible region of the actual model belongs to the feasible region of the synthesized model.

Mean angle between weight vectors of the corresponding constraints in the synthesized and actual models (details below).

Number of constraints in the synthesized model.

Number of terms in the synthesized model, where term is a summand in a constraint: a weight multiplied by a variable or constant. E.g., $x_1 \leq 5 - Mb_1$ has tree terms.

Mean angle assesses similarity of syntax of the corresponding constraints in the synthesized and actual models. Let $\boldsymbol{w}_i^s$ and $\boldsymbol{w}_k^b$ be weight vectors of a synthesized constraint c_i^s and an actual constraint c_k^b, respectively, then an angle between them is $\alpha_{ik} = \arccos\left|\boldsymbol{w}_i^s \cdot \boldsymbol{w}_k^b / (\|\boldsymbol{w}_i^s\| \|\boldsymbol{w}_k^b\|)\right|$. Corresponding constraints are pairs (c_i^s, c_k^b) having minimal α_{ik}, and the mean angle between such pairs is calculated by solving an *assignment* problem:

$$\begin{array}{rcll} \min & \frac{1}{N} & \sum_{ik} \alpha_{ik} d_{ik} & \text{the mean angle} \\ \text{subject to} & & & \\ \forall i : \sum_j d_{ik} & \geq & 1 & \boldsymbol{w}_i^s \text{ paired with at least one } \boldsymbol{w}_k^b \\ \forall k : \sum_i d_{ik} & \geq & 1 & \boldsymbol{w}_k^b \text{ paired with at least one } \boldsymbol{w}_i^s \\ \forall d_{ik} & \in & \{0,1\} & \text{indicator of pairing } \boldsymbol{w}_i^s \text{ and } \boldsymbol{w}_k^b, \end{array}$$

where N is the greater of the number of synthesized and actual constraints. Mean angle attains 0 if constraints are pairwise parallel and maximum value of $\pi/2$ if pairwise orthogonal. For LP models mean angle in the weight space equals to mean angle between constraints' hyperplanes.

5.2. Gaussian Mixture Model parameters tuning

We tune number of Gaussians g in Gaussian Mixture Model (GMM) involved in modeling distribution of feasible examples, as described in Section 4.1. We expect that g increases with number of variables n, as number of parameters of the actual distribution of examples may increase with n. For that reason, we verify $g \in \{\lceil 0.5n \rceil, n, 2n\}$. We run CSC4.5 with C4.5 parameters join and max height set to $j = 10$ and $h = 10$, respectively.

Table 2 shows means and standard deviations of performance measures on test-set for the synthesized models w.r.t. g and n. Generally speaking, all measures worsen with dimensionality n. We attribute this to *curse of dimensionality* (Bellman [4]) — given fixed cardinality of X, the spatial distribution of examples in X quickly becomes sparse with increasing n. In consequence, it is more likely to find a set of constraints satisfying X but inconsistent with actual constraints of benchmarks. The synthesized models have more constraints and terms than actual benchmark models except for Cube$n$ and $n \geq 6$. Jaccard

Table 2: Means of measures for the synthesized models w.r.t. number of components g in Gaussian Mixture Model. Bars reflect standard deviations (cell height corresponds to 10 and 100 for numbers of constraints and terms, respectively, and 0.5 for the remaining measures). Heatmaps reflect values: green for the best in a column, red for the worst in a column.

Ball$n$

Dimensions n	Components g	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	2	43 ± 303	0.730	0.774	0.929	0.994	
	3	40 ± 274	0.724	0.761	0.936	1.000	
	6	43 ± 300	0.730	0.771	0.933	1.006	
4	2	45 ± 318	0.566	0.634	0.844	1.100	
	4	44 ± 302	0.559	0.619	0.858	1.103	
	8	43 ± 297	0.562	0.629	0.844	1.100	
5	3	45 ± 321	0.420	0.445	0.888	1.155	
	5	45 ± 318	0.413	0.435	0.895	1.154	
	10	45 ± 325	0.417	0.441	0.889	1.153	
6	3	48 ± 355	0.146	0.149	0.877	1.199	
	6	48 ± 353	0.143	0.146	0.843	1.196	
	12	49 ± 358	0.148	0.151	0.874	1.194	
7	4	54 ± 408	0.027	0.027	0.511	1.236	
	7	55 ± 410	0.030	0.030	0.569	1.232	
	14	54 ± 404	0.023	0.023	0.427	1.235	

Cube$n$

Dimensions n	Components g	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	2	11 ± 23	0.984	0.999	0.985	0.000	
	3	12 ± 31	0.983	0.999	0.985	0.000	
	6	15 ± 46	0.978	0.999	0.979	0.000	
4	2	11 ± 19	0.981	0.998	0.982	0.000	
	4	12 ± 22	0.979	0.998	0.981	0.000	
	8	12 ± 25	0.976	0.996	0.980	0.000	
5	3	11 ± 28	0.487	0.498	0.822	0.247	
	5	11 ± 26	0.526	0.533	0.919	0.162	
	10	12 ± 28	0.491	0.497	0.887	0.135	
6	3	10 ± 21	0.113	0.119	0.426	0.671	
	6	11 ± 29	0.084	0.087	0.362	0.632	
	12	10 ± 26	0.034	0.042	0.099	0.833	
7	4	9 ± 20	0.012	0.045	0.101	0.905	
	7	11 ± 31	0.008	0.019	0.065	0.865	
	14	11 ± 32	0.001	0.011	0.001	0.927	

Simplex$n$

Dimensions n	Components g	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	2	45 ± 317	0.638	0.667	0.936	0.092	
	3	47 ± 339	0.645	0.681	0.924	0.091	
	6	50 ± 373	0.651	0.696	0.912	0.090	
4	2	50 ± 332	0.399	0.415	0.915	0.092	
	4	50 ± 335	0.390	0.408	0.905	0.091	
	8	50 ± 338	0.388	0.407	0.893	0.091	
5	3	50 ± 325	0.174	0.180	0.859	0.093	
	5	50 ± 327	0.176	0.182	0.854	0.093	
	10	49 ± 316	0.179	0.185	0.855	0.093	
6	3	54 ± 372	0.026	0.026	0.733	0.104	
	6	54 ± 369	0.027	0.027	0.765	0.100	
	12	53 ± 360	0.026	0.026	0.750	0.101	
7	4	51 ± 343	0.005	0.005	0.167	0.195	
	7	50 ± 337	0.005	0.005	0.167	0.185	
	14	51 ± 341	0.005	0.005	0.167	0.179	

index and precision are the highest for Cube$n$ as expected, since Cube$n$ can be directly reproduced by CSC4.5. Recall is high for all benchmarks for $n \leq 6$ and decreases with n slower than Jaccard index and precision. Mean angles are low and below 0.2rad for Cube$n$ and Simplex$n$ for $n \leq 5$.

All measures of quality seem independent from g: however there are small fluctuations of measures for changing g, there are no clear trends. To statistically assess these fluctuations, we calculate Spearman's rank correlation coefficients for all measures and g/n. We divide g by n to compensate possible dependence on n. The correlation coefficients are close to zero and two-tailed t-tests for significance of correlation at $\alpha = 0.05$ conclude lack of correlations for all measures. For that reason, we use median value of $g = n$ for the remaining experiments.

5.3. C4.5 parameters tuning

We tune parameters of C4.5 algorithm: join j and max height h and whether to use error-based post-pruning. Similarly, like for number of Gaussians g, we expect that j and h depend on number of variables n because of potentially increasing complexity of distribution of examples. Thus, we verify $j \in \{n, \lceil 1.5n \rceil, 2n\}$ and $h \in \{j, \lceil 1.5j \rceil, 2j\}$. When pruning is employed, we split actual training set T into training set T' and validation set V, such that $\frac{|V|}{|T'|} = \frac{1}{9}$, and use T' to train decision tree and V to calculate pruning errors. We employ error-based pruning implementation by Souza [28] and use its default parameters. In this experiment, we use $g = n$, as concluded in the previous experiment.

Table 3 shows means and standard deviations of performance measures on test-set for the synthesized models w.r.t. j, h, n and without tree post-pruning. Enabling pruning significantly decreases size of the decision trees and thus LP models, however in most cases leads to significantly worse results on the re-

Table 3: Means of measures for the synthesized models w.r.t. join j and max height h parameters of C4.5 algorithm and no pruning. Bars reflect standard deviations (cell height corresponds to 15 and 350 for the numbers of constraints and terms, respectively, and 0.5 for the remaining measures). Underlining marks significant differences between results with and without pruning w.r.t. t-test. Heatmaps reflect values: green for the best in a column, red for the worst in a column.

Ball$n$

Dimensions n	Join j	Max Height h	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	3	3	1	1	0.000	0.006	0.000	0.276
		5	9	24	0.400	0.407	0.928	1.017
		6	12	36	0.652	0.678	0.945	1.010
	5	5	10	30	0.400	0.407	0.927	0.991
		8	25	130	0.675	0.731	0.902	0.981
		10	40	278	0.723	0.761	0.936	0.999
	6	6	14	51	0.651	0.678	0.945	0.991
		9	33	196	0.694	0.738	0.923	0.993
		12	56	462	0.766	0.815	0.927	1.006
4	4	4	6	12	0.000	0.000	0.000	1.073
		6	13	46	0.009	0.014	0.031	1.107
		8	27	140	0.530	0.560	0.911	1.095
	6	6	13	46	0.018	0.021	0.062	1.113
		9	34	205	0.551	0.602	0.868	1.101
		12	63	524	0.587	0.627	0.902	1.108
	8	8	27	142	0.531	0.561	0.911	1.100
		12	62	523	0.586	0.626	0.903	1.108
		16	94	1021	0.642	0.700	0.887	1.114
5	5	5	11	39	0.000	0.001	0.000	1.203
		8	30	169	0.120	0.125	0.569	1.137
		10	44	309	0.415	0.438	0.893	1.161
	8	8	30	173	0.119	0.125	0.570	1.176
		12	62	526	0.438	0.474	0.856	1.151
		16	98	1112	0.485	0.522	0.872	1.157
	10	10	45	318	0.413	0.435	0.895	1.154
		15	91	982	0.468	0.501	0.877	1.158
		20	126	1707	0.513	0.567	0.845	1.172
6	6	6	16	68	0.000	0.000	0.000	1.191
		9	38	250	0.026	0.028	0.266	1.198
		12	65	555	0.320	0.338	0.863	1.192
	9	9	39	251	0.024	0.026	0.236	1.198
		14	86	852	0.325	0.345	0.853	1.198
		18	121	1487	0.361	0.387	0.850	1.208
	12	12	66	564	0.324	0.343	0.863	1.199
		18	121	1505	0.357	0.382	0.849	1.207
		24	152	2222	0.375	0.411	0.814	1.215
7	7	7	27	139	0.000	0.000	0.000	1.227
		11	66	536	0.060	0.061	0.848	1.234
		14	94	937	0.192	0.201	0.834	1.229
	11	11	66	543	0.062	0.063	0.857	1.235
		17	128	1527	0.187	0.194	0.852	1.240
		22	156	2203	0.199	0.210	0.821	1.244
	14	14	94	938	0.192	0.201	0.836	1.231
		21	153	2113	0.199	0.209	0.821	1.239
		28	167	2520	0.202	0.213	0.808	1.247

Cube$n$

Dimensions n	Join j	Max Height h	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	3	3	1	1	0.000	0.029	0.000	0.244
		5	5	6	0.426	0.440	0.891	0.175
		6	6	8	0.873	0.883	0.988	0.000
	5	5	5	6	0.426	0.440	0.891	0.175
		8	9	19	0.984	0.999	0.985	0.000
		10	12	31	0.983	0.999	0.985	0.000
	6	6	6	8	0.873	0.883	0.988	0.000
		9	11	25	0.983	0.999	0.985	0.000
		12	12	36	0.982	0.999	0.984	0.000
4	4	4	4	8	0.001	0.046	0.001	0.982
		6	6	6	0.277	0.285	0.823	0.471
		8	8	11	0.892	0.906	0.985	0.000
	6	6	6	6	0.275	0.284	0.818	0.474
		9	10	15	0.949	0.964	0.985	0.000
		12	18	49	0.974	0.998	0.976	0.000
	8	8	8	11	0.889	0.903	0.985	0.000
		12	18	50	0.975	0.999	0.977	0.000
		16	21	71	0.971	0.998	0.973	0.000
5	5	5	3	5	0.000	0.022	0.000	0.848
		8	7	12	0.139	0.158	0.494	0.696
		10	11	28	0.481	0.488	0.920	0.179
	8	8	7	13	0.127	0.150	0.462	0.706
		12	14	38	0.925	0.941	0.982	0.020
		16	21	87	0.957	0.981	0.974	0.000
	10	10	11	26	0.526	0.533	0.919	0.162
		15	20	71	0.960	0.982	0.977	0.000
		20	24	114	0.953	0.981	0.970	0.000
6	6	6	6	11	0.001	0.011	0.001	1.200
		9	9	21	0.025	0.029	0.137	0.817
		12	14	46	0.336	0.341	0.785	0.212
	9	9	9	19	0.030	0.034	0.165	0.833
		14	17	58	0.738	0.754	0.974	0.034
		18	27	131	0.914	0.948	0.962	0.000
	12	12	14	46	0.363	0.370	0.785	0.227
		18	28	130	0.919	0.951	0.965	0.000
		24	36	215	0.902	0.943	0.953	0.000
7	7	7	8	18	0.000	0.000	0.000	0.986
		11	12	35	0.018	0.028	0.099	0.805
		14	19	77	0.205	0.208	0.752	0.264
	11	11	12	37	0.016	0.026	0.098	0.772
		17	22	88	0.699	0.712	0.982	0.004
		22	35	207	0.857	0.888	0.961	0.000
	14	14	17	61	0.270	0.274	0.784	0.217
		21	32	175	0.854	0.879	0.968	0.000
		28	46	350	0.833	0.872	0.947	0.000

Simplex$n$

Dimensions n	Join j	Max Height h	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	3	3	1	1	0.000	0.001	0.000	0.392
		5	11	36	0.339	0.354	0.869	0.184
		6	17	65	0.463	0.487	0.911	0.126
	5	5	12	40	0.338	0.353	0.871	0.179
		8	32	180	0.552	0.581	0.922	0.100
		10	47	341	0.645	0.680	0.927	0.091
	6	6	17	67	0.464	0.489	0.907	0.127
		9	39	245	0.638	0.681	0.911	0.095
		12	60	500	0.680	0.720	0.926	0.087
4	4	4	5	10	0.000	0.000	0.000	1.008
		6	18	72	0.118	0.120	0.862	0.292
		8	32	166	0.339	0.356	0.887	0.104
	6	6	18	71	0.119	0.120	0.862	0.297
		9	41	240	0.341	0.359	0.885	0.096
		12	70	577	0.469	0.497	0.893	0.086
	8	8	31	163	0.343	0.360	0.888	0.104
		12	70	576	0.468	0.497	0.893	0.086
		16	92	913	0.502	0.537	0.887	0.082
5	5	5	9	26	0.000	0.000	0.000	0.834
		8	32	175	0.068	0.069	0.864	0.158
		10	50	327	0.174	0.180	0.857	0.093
	8	8	33	176	0.068	0.069	0.864	0.163
		12	70	559	0.196	0.203	0.865	0.087
		16	106	1131	0.235	0.246	0.837	0.082
	10	10	50	327	0.176	0.182	0.854	0.093
		15	98	995	0.229	0.238	0.850	0.082
		20	116	1312	0.239	0.251	0.834	0.081
6	6	6	16	59	0.000	0.000	0.000	0.724
		9	42	259	0.021	0.021	0.737	0.136
		12	73	580	0.047	0.048	0.776	0.087
	9	9	42	259	0.020	0.021	0.737	0.142
		14	93	839	0.050	0.051	0.834	0.083
		18	121	1342	0.054	0.054	0.800	0.081
	12	12	74	584	0.047	0.048	0.789	0.087
		18	122	1353	0.053	0.054	0.834	0.081
		24	127	1468	0.055	0.055	0.834	0.080
7	7	7	22	98	0.034	0.000	0.033	0.679
		11	62	451	0.008	0.008	0.167	0.135
		14	91	810	0.012	0.012	0.167	0.085
	11	11	62	461	0.008	0.008	0.167	0.134
		17	109	1117	0.012	0.012	0.167	0.082
		22	126	1468	0.012	0.012	0.167	0.081
	14	14	91	814	0.012	0.012	0.167	0.084
		21	125	1442	0.012	0.012	0.167	0.081
		28	129	1523	0.012	0.012	0.167	0.080

maining measures w.r.t. two-tailed t-test at $\alpha = 0.05$ which we marked using underlining in Table 3.

Low values of $j = n$ and $h = j$ lead to MILP models with low numbers of constraints and terms but also poor-fitted with extremely bad values of the remaining measures. This may be because these values limit total number of resulting constraints to n which is too low for all benchmarks considered here (even for Ball$n$, because the actual quadratic constraint has to be approximated by many linear ones). Thus, practical use of $h = j = n$ is questionable. In contrast, $j = 2n$ and $h = 2j$ provide highest Jaccard indexes and precision for all benchmarks. For Cube$n$ high values are observed for $j \geq \lceil 1.5n \rceil$ and $h \geq \lceil 1.5j \rceil$ too, however this result is specific to the problem: $2n$ actual constraints of one variable each can be easily reconstructed using $h \geq \lceil 1.5j \rceil \geq \lceil 1.5 \cdot \lceil 1.5n \rceil \rceil \geq \lceil 2.25n \rceil$ constraints. For recall and mean angle, it is difficult to clearly find the best combination of j and h, however good results are observed for $j \geq \lceil 1.5n \rceil$ and $h \geq \lceil 1.5j \rceil$ too.

Given the above results, in the following we use $j = 2n$ and $h = 2j$ and no pruning as CSC4.5's parameters.

5.4. Impact of the number of examples

In this experiment, we verify how the size of set of examples X influences performance of CSC4.5. We use $|X| \in \{100, 200, 300, 400, 500\}$, as we consider $|X| = 500$ used in the previous experiments as large in context of typically available sets in real-world problems. We use $g = n$, $j = 2n$, $h = 2j$ and no pruning, as concluded by previous experiments.

Table 4 shows means and standard deviations of performance measures on test-set for the synthesized models w.r.t. cardinality of X. For Ball$n$ and Simplex$n$ numbers of constraints and terms increase with $|X|$. This is expected, as greater number of examples specifies shape of the feasible region more precisely and thus may require more constraints to model this shape. Cube$n$ is an exception, for which numbers of examples and terms decrease with $|X|$. This may be due to matching form actual and synthesized constraints. The same reasons explain increasing with $|X|$ Jaccard index and recall and decreasing mean angle. In turn, precision seems to be mainly unaffected by cardinality of X, as precision is roughly constant and faces only small fluctuations across cardinalities.

Clearly, problem dimensionality n seem to have stronger impact on results: the higher n the bigger models are synthesized. Also, the higher n, the worse Jaccard index, precision and recall. Surprisingly, mean angle characteristic w.r.t. n is problem-dependent: for Ball$n$ mean angle increases with n, for Cube$n$ is constant and equal to 0rad (except for $n = 7$ and $|X| = 100$) and for Simplex$n$ fluctuates in range 0.08–0.1rad. For Ball$n$ we explain this result by lack of quadratic terms in the synthesized constraints and thus zero weights used for them in calculation of mean angles (cf. Sect. 5.1). In turn, Cube$n$ requires constraints orthogonal to axes of coordinate system, exactly as synthesized by CSC4.5 and thus zero angles for this benchmark. The exception for $n = 7$

Table 4: Means of measures for the synthesized models w.r.t. X cardinality. Bars reflect standard deviations (cell height corresponds to 15 and 350 for numbers of constraints and terms, respectively, and 0.5 for the remaining measures). Heatmaps reflect values: green for the best in a column, red for the worst in a column.

Ball$n$							
Dimensions n	Examples $\|X\|$	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	100	32	191	0.683	0.823	0.804	1.029
	200	38	242	0.729	0.811	0.879	1.027
	300	45	311	0.746	0.803	0.915	1.025
	400	52	392	0.761	0.816	0.919	1.017
	500	56	462	0.766	0.815	0.927	1.006
4	100	48	354	0.547	0.713	0.704	1.122
	200	64	570	0.601	0.705	0.806	1.120
	300	77	757	0.621	0.700	0.849	1.118
	400	90	961	0.635	0.703	0.868	1.118
	500	95	1023	0.642	0.699	0.887	1.114
5	100	58	511	0.423	0.595	0.604	1.165
	200	80	851	0.470	0.563	0.741	1.173
	300	102	1200	0.481	0.551	0.792	1.167
	400	115	1484	0.490	0.548	0.824	1.180
	500	126	1707	0.513	0.567	0.845	1.172
6	100	71	750	0.280	0.360	0.567	1.206
	200	98	1161	0.342	0.403	0.697	1.211
	300	114	1501	0.343	0.383	0.770	1.213
	400	138	1963	0.347	0.383	0.794	1.213
	500	151	2209	0.376	0.412	0.815	1.215
7	100	74	795	0.177	0.210	0.539	1.245
	200	102	1266	0.185	0.202	0.716	1.236
	300	126	1774	0.216	0.231	0.766	1.242
	400	148	2169	0.210	0.225	0.780	1.245
	500	167	2520	0.202	0.213	0.808	1.247

Cube$n$							
Dimensions n	Examples $\|X\|$	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	100	16	57	0.874	0.983	0.887	0.000
	200	14	44	0.946	0.997	0.948	0.000
	300	13	38	0.968	0.998	0.970	0.000
	400	13	35	0.977	0.998	0.979	0.000
	500	12	36	0.982	0.999	0.984	0.000
4	100	27	138	0.774	0.945	0.812	0.000
	200	26	130	0.894	0.971	0.918	0.000
	300	22	85	0.952	0.993	0.958	0.000
	400	22	82	0.962	0.995	0.966	0.000
	500	21	72	0.970	0.998	0.972	0.000
5	100	41	283	0.639	0.858	0.718	0.000
	200	36	247	0.825	0.930	0.880	0.000
	300	34	192	0.900	0.969	0.927	0.000
	400	27	142	0.940	0.976	0.962	0.000
	500	24	114	0.953	0.981	0.970	0.000
6	100	41	340	0.516	0.662	0.709	0.000
	200	46	371	0.745	0.854	0.852	0.000
	300	44	320	0.828	0.903	0.908	0.000
	400	37	238	0.885	0.935	0.944	0.000
	500	36	215	0.902	0.943	0.953	0.000
7	100	45	421	0.361	0.449	0.689	0.003
	200	59	567	0.612	0.717	0.815	0.000
	300	59	541	0.684	0.779	0.852	0.000
	400	50	410	0.808	0.863	0.927	0.000
	500	46	350	0.833	0.872	0.947	0.000

Simplex$n$							
Dimensions n	Examples $\|X\|$	Constraints	Terms	Jaccard Index	Precision	Recall	Mean angle
3	100	33	209	0.583	0.698	0.783	0.099
	200	43	305	0.640	0.715	0.861	0.094
	300	51	399	0.650	0.707	0.892	0.090
	400	56	452	0.671	0.716	0.917	0.088
	500	60	500	0.680	0.720	0.926	0.087
4	100	44	324	0.344	0.403	0.716	0.094
	200	63	537	0.429	0.481	0.801	0.087
	300	79	735	0.472	0.516	0.850	0.084
	400	86	835	0.473	0.510	0.868	0.083
	500	92	913	0.502	0.537	0.887	0.082
5	100	52	412	0.143	0.157	0.658	0.092
	200	75	713	0.171	0.182	0.756	0.086
	300	93	957	0.198	0.209	0.791	0.083
	400	106	1175	0.225	0.236	0.834	0.082
	500	116	1312	0.239	0.251	0.834	0.081
6	100	57	505	0.028	0.029	0.494	0.091
	200	81	796	0.045	0.046	0.665	0.085
	300	98	1057	0.056	0.057	0.833	0.083
	400	110	1213	0.059	0.060	0.752	0.082
	500	127	1468	0.055	0.055	0.834	0.080
7	100	55	456	0.003	0.003	0.117	0.097
	200	83	814	0.006	0.006	0.167	0.086
	300	100	1044	0.008	0.008	0.167	0.083
	400	113	1250	0.011	0.012	0.183	0.082
	500	129	1523	0.012	0.012	0.167	0.080

and $|X| = 100$ is due to one actual constraint having no corresponding parallel constraint in the synthesized model and thus paired with orthogonal one. For Simplex$n$ the range of mean angles is easily explained by constants $\tan\frac{\pi}{12}$ and $\cot\frac{\pi}{12}$ in $n(n-1)$ actual constraints (cf. Table 1) that cause them to be inclined to the corresponding axes at $\beta = \arctan\frac{\tan\pi/12}{\cot\pi/12} \approx 0.072\text{rad}$. β is thus a lower bound on mean angle achievable by CSC4.5 for Simplex$n$ and actual mean angles are a bit larger due to one constraint inclined at higher angle in this benchmark and possibly missing constraints corresponding to some axes in the synthesized models.

6. Case study: Wine Quality

We apply CSC4.5 to a real-world problem of constraint synthesis for wine production process, combine the synthesized mixed-integer linear constraints with a quadratic objective function representing quality of wine and optimize the resulting Mixed-Integer Quadratic Programming (MIQP) model using an off-the-shelf solver to find the best wine composition. We validate the obtained results with a naive method based on computing convex hull of a dataset. We employ Wine Quality dataset by Cortez et al. [10] composed of 1,599 red and 4,898 white wine examples, each described using eleven input variables corresponding to volumes of wine components, and one output variable representing quality grade Q, calculated as the median of the grades assigned by at least

Table 5: Variables in Wine Quality dataset (Cortez et al. [10]).

		Red wine			White wine		
Variable	Meaning	Min	Mean	Max	Min	Mean	Max
FA	Fixed acidity g(tartaric acid)$/dm^3$	4.60	8.31	15.90	3.80	6.84	14.20
VA	Volatile acidity g(acetic acid)$/dm^3$	0.12	0.53	1.58	0.08	0.28	1.10
CA	Citric acid g/dm^3	0.00	0.27	1.00	0.00	0.33	1.66
RS	Residual sugar g/dm^3	0.90	2.52	15.50	0.60	5.91	65.80
C	Chlorides g(sodium chloride)$/dm^3$)	0.01	0.09	0.61	0.01	0.05	0.35
FSD	Free sulfur dioxide mg/dm^3	1.00	15.89	72.00	2.00	34.89	289.00
TSD	Total sulfur dioxide mg/dm^3	6.00	46.83	289.00	9.00	137.19	440.00
D	Density g/dm^3	0.99	0.997	1.00	0.99	0.994	1.04
pH	pH	2.74	3.31	4.01	2.72	3.20	3.82
S	Sulfates g(potassium sulfate)$/dm^3$	0.33	0.66	2.00	0.22	0.49	1.08
A	Alcohol $vol.\%$	8.40	10.43	14.90	8.00	10.59	14.20
Q	Quality (dependent variable)	3.00	5.62	8.00	3.00	5.85	9.00

three sensory assessors and varying in range 0 (bad) to 10 (excellent). Table 5 summarizes variables' statistics. We split this dataset using uniform sampling into training and test-sets of 800 and 799 examples of red wine, respectively, and 2,449 and 2,449 examples for white wine, respectively. Cortez et al. [10] do not provide infeasible examples, therefore to assess quality of the resulting MIQP models, we supplement the test-sets using unlabeled examples sampled the same way as U for the training set in Section 4.1. This results in red wine test-set T_r of 799 feasible and 96,679 unlabeled examples, and white wine test-set T_w of 2,449 feasible and 296,329 unlabeled examples.

We employ CSC4.5's parameters found in the previous experiments, i.e., $g = n$, $j = 2n$, $h = 2j$, no pruning, and assume $\rho = 1\%$ of outliers due to the real-world origin of the dataset.

We build separate MIQP models for red and white wine by repeating the following steps for each wine color. We run CSC4.5 30 times with different random seeds and pick the set of constraints that maximizes Jaccard index on the test-set. For red wine, the best model has 517 constraints and 0.90 Jaccard index, for white wine these amount to 1,373 and 0.91, respectively. We supplement the model with an objective function calculated using least-squares quadratic regression of the dependent variable Q w.r.t. independent variables from Table 5. From the resulting regression function we remove all terms whose coefficients are insignificant w.r.t. t-test and start over. The regression functions obtained in this way turn out to be not concave for both red and white wine and as most MIQP solvers requires concavity, we remove the minimal subset of terms that prevents a function from being concave and start over. The resulting functions for red and white wine have coefficients of determination $r^2 = 0.39$ and 0.33, respectively. Since Cortez et al. [10] do not specify domains of the variables, we supplement specification of MIQP model with variables' bounds calculated as their respective minimums and maximums in Table 5. The resulting MIQP models are provided as supplementary material due to

Table 6: The optimal solutions for the wine models and the most similar wines w.r.t. Canberra distance.

	Red wine				White wine	
	Optimal solution	Most similar	Optimal in convex hull	Most similar	Optimal solution	Most similar
FA	8.070	5.900	8.711	8.400	8.239	6.800
VA	0.120	0.440	0.367	0.340	0.080	0.150
CA	0.000	0.000	0.608	0.420	0.490	0.410
RS	1.400	1.600	0.903	2.100	65.800	12.900
C	0.010	0.042	0.081	0.072	0.010	0.044
FSD	3.000	3.000	23.429	23.000	85.088	79.500
TSD	6.000	11.000	37.040	36.000	126.982	182.000
D	0.991	0.994	1.004	0.994	0.990	0.997
pH	2.740	3.480	2.740	3.110	3.820	3.240
S	1.136	0.850	1.139	0.780	1.080	0.780
A	14.900	11.700	14.900	12.400	14.200	10.200
Q (objective)	8.265	6.000	7.979	6.000	12.891	6.000
Distance to optimal		2.088		1.069		2.414
Percentile in dataset		86%		86%		79%

large number of constraints.

Given the complete models, it is tempting to ask, what exactly are the optimal proportions of wine components? To answer this question, we optimize both MIQP models using Gurobi Optimization, Inc. [15] solver and show the optimal solutions in Table 6. The optimal red wine composition has estimated quality of 8.265 and features relatively high amount of sulfates and alcohol and low-to-moderate amounts of other components. In turn, the optimal white wine composition has estimated quality of 12.891 and features high amounts of sugar, sulfates and alcohol and low-to-moderate amounts of other components. We validate these solutions by comparing them in Table 6 to the most similar examples in the dataset w.r.t. Canberra distance (Lance and Williams [19]) – the city-block metric augmented to handle variables of different domains. Both red and white most similar wines have quality of 6, lower than the estimates for the optimal solutions, however still not worse than 86% and 79% of examples in the red and white wine sets, respectively. We attribute these inaccurate estimates to low fitting of regression functions in objectives, signaled by low r^2 coefficients.

It is worth to note that the practical applicability of these optimal solutions cannot be ensured using the above evidence. Assuming for the sake of argument that feasible wine compositions are actually expressible using linear constraints, it is the facets of convex hull of feasible examples in the training set that bound the region of solutions with *guaranteed* practical applicability. By definition, a convex hull is the smallest superset of a set expressible using

linear constraints. Calculating convex hull is, however, exponential in time and space w.r.t. dimensionality, due to exponential number of resulting constraints. We calculated convex hull of feasible examples in red wine training set using qhull tool by Barber et al. [2] which lasted 55 minutes on Core i7-5960X CPU, consumed 25GB of RAM and resulted in 32,148,366 constraints. The same tool run for white wine training set after 12 hours of computation consumed 80GB of RAM, and we had to terminate it due to insufficient computational resources and resulted with an incomplete convex hull made of 1.5×10^8 constraints. For that reason, we optimized only the red wine convex hull-based model w.r.t. the same objective function as above using Gurobi Optimization, Inc. [15] solver which lasted 21 minutes and consumed 31GB of RAM. Comparing these quantities to an average of 13 minutes per each of 30 runs of CSC4.5, 517 constraints in the best model found by CSC4.5 and negligible time of solving this model, clearly shows advantage of CSC4.5 in terms of required computational resources and ease of optimization of the resulting model.

Table 6 shows the optimal solutions to the convex hull-based model. It scores lower estimate of quality than the optimal solution to the above model. This is expected, as a convex hull-based model, in contrast to a CSC4.5-synthesized model, wraps the feasible examples from the training set very tightly leaving no margin surrounding them. For the same reason, it is much closer to the actual examples in the dataset than the solution to the CSC4.5-synthesized model (cf. Table 6). In the effect, the optimal solution to the convex hull-based model is only slightly better than the already known solutions from the dataset and require only small changes in the existing business process to be applied. However, further improvements in this process cannot be done solely by means of the convex hull, as it remains unchanged when including this solution in the training set. CSC4.5 is free from this limitation, as it produces models with margins surrounding examples and thus allowing for improved solutions.

7. Discussion

CSC4.5 has several advantages making it suitable to use in real-world scenarios like e.g., support for one-class input data and synthesis of constraints in linear form. Concerning the former, many modeling methods reviewed in Section 3 require examples of feasible and infeasible classes, while in actually available data sets of real-world processes the infeasible examples are few and far between. This mismatch of characteristics of real-world data sets and requirements of modeling methods is addressed in design of CSC4.5 that synthesizes constraints using one-class data only.

The Mixed-Integer Linear Programming (MILP) form of constraints makes the models synthesized by CSC4.5 easy to interpret, modify and solve. MILP constraints synthesized by CSC4.5 can be decomposed into two parts: an input variable compared to a constant and a weighted sum of auxiliary binary variables. As the latter part is purely technical, it can be completely ignored when interpreting a model. What is important, given a visible drawback or flaw in an automatically synthesized MILP model, an expert can easily improve the

model by introducing extra constraints, removing existing ones or modifying them. Note that the number of auxiliary binary variables composing the technical part is usually small and comparable to the number of input variables. This causes solving of a synthesized MILP model, although NP-hard in theory (Karp [17]), computationally feasible in practice.

Despite above advantages, CSC4.5 is not free of challenges, where the most striking one is fixed form of constraints making them unable to catch weighted relationships between variables. This makes CSC4.5 particularly suitable for a narrow class of problems, like Cube$n$ in the experimental part, and far from optimal in problems of other types. A possible workaround for the above is replacement of C4.5 with an alternative tree induction algorithm supporting comparisons of weighted sum of variables to constants, like e.g., Classification and Regression Tree (CART, Breiman et al. [8]). However, this is out of the scope of this study.

CSC4.5 clearly suffers from curse of dimensionality (Bellman [4]). This, however, may be specifics of constraint synthesis problem, as posed in Section 2 rather than CSC4.5 itself, as number of input variables n can be arbitrarily large. The deteriorating impact of n can be to some extent alleviated by increasing cardinality of set of examples X: given unsatisfactory results one may acquire extra training examples and rerun CSC4.5 to obtain better MILP models. This should be possible in most real-world scenarios, where a process operates regularly and an aim is to model and optimize this process. Despite curse of dimensionality, CSC4.5 offers recall of over 0.8 even for problems with 6-7 variables, given 500 feasible examples. In turn, precision attains values over 0.7 for 500 examples and 3-7 variables depending on a problem. Given probabilistic interpretation of these measures (cf. Sect. 5.1), CSC4.5 produces MILP models which feasible regions very likely incorporate actual feasible regions (high recall), but also noticeable parts of actual infeasible regions (lower precision). This characteristic of models may be detrimental to optimization, as an optimal solution often lies on boundary of model's feasible region and if this optimal solution happens to be outside the actual feasible region, it would be inapplicable in practice. This, however, might not be a critical issue if the actual feasible region has fuzzy (e.g., probabilistic) boundaries, as an overestimated optimal solution may still find practical use. The wine production problem in our case study (Sect. 6) is just one example of that case, as overestimated proportions of wine components do not preclude the final mixture to pass as a wine.

For problems with strict constraints, however, CSC4.5 may be not as good as computing convex hull of a set of feasible examples X. Computing convex hull is, in fact, full of challenges: the number of resulting constraints is exponential with the number of variables n, and thus the time and memory requirements quickly become unacceptable, even for small training sets. Also, excessive number of constraints hinders interpretability of a model, precluding thus inspection by humans. Also, use of convex hull facets as constraints precludes optimization of linear objective functions beyond values for known examples in X, as optimal solutions for linear models lie in vertices of the feasible region and every vertex is actually an example from X. Finally, if examples in X form clusters, convex hull

of X would include 'empty' regions corresponding to likely infeasible solutions and thus erroneously included in the feasible region. This case is explicitly handled by CSC4.5, in which clusters of examples are separately wrapped by subsets of constraints and alternative of these subsets is modeled using auxiliary binary variables.

8. Conclusions and future work

We proposed a Constraint Synthesis with C4.5 (CSC4.5) algorithm to automatize modeling of real-world processes and systems using Mixed-Integer Linear Programming models. The unique feature of CSC4.5 is support for building MILP models given only the examples of snapshots of feasible process states. This eliminates the need for acquisition of examples of infeasible states, often being expensive to acquire. Empirical assessment of CSC4.5 reveals considerable fidelity of the synthesized MILP models to the actual models. CSC4.5 applied to real-world problem of modeling wine production process synthesized models by orders of magnitude smaller than the models calculated using naive approach, allowing thus for efficient optimization of that process.

CSC4.5 synthesizes constraints in a fixed-form involving single input variable per constraint. This is a straightforward consequence of comparisons made in decision tree branches by C4.5. Therefore, the main course of future work is to eliminate the abovementioned limitation by replacing C4.5 with an alternative tree learning algorithm with ability to compare weighted sum of variables to a constant, e.g., Classifier and Regression Tree (CART, Breiman et al. [8]). Side works relate to alternative ways of modeling distribution of feasible examples using e.g., Multivariate Kernel Density Estimation (Simonoff [27]) and use of alternative tree pruning strategies to reduce MILP model size.

Acknowledgments. The work of P. Kudła was funded by Poznan University of Technology, Poland, grant no. 09/91/DSMK/0634, the work of T. Pawlak was funded by National Science Centre, Poland, grant no. 2016/23/D/ST6/03735.

[1] Aswal, A., Prasanna, G. N. S., 2010. Estimating correlated constraint boundaries from timeseries data: The multi-dimensional german tank problem. In: EURO 2010. `http://slideplayer.com/slide/7976536/`.

[2] Barber, C. B., Dobkin, D. P., Huhdanpaa, H., Dec. 1996. The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. 22 (4), 469–483. URL `http://doi.acm.org/10.1145/235815.235821`

[3] Beldiceanu, N., Simonis, H., Oct. 2012. A model seeker: Extracting global constraint models from positive examples. In: CP'12. Vol. LNCS 7514. Springer, Quebec City, Canada, pp. 141–157.

[4] Bellman, R., 2013. Dynamic Programming. Dover Books on Computer Science. Dover Publications.
URL `https://books.google.it/books?id=CG7CAgAAQBAJ`

[5] Bessiere, C., Coletta, R., Hebrard, E., Katsirelos, G., Lazaar, N., Narodytska, N., Quimper, C.-G., Walsh, T., 2013. Constraint acquisition via partial queries. In: IJCAI 2013. pp. 475–481.

[6] Bessiere, C., Coletta, R., Koriche, F., O'Sullivan, B., 2005. A SAT-Based Version Space Algorithm for Acquiring Constraint Satisfaction Problems. Springer Berlin Heidelberg, pp. 23–34.

[7] Blonder, B., Lamanna, C., Violle, C., Enquist, B. J., 2014. The n-dimensional hypervolume. Global Ecology and Biogeography 23, 595–609.

[8] Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J., 1984. Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA.

[9] Cortes, C., Vapnik, V., 1995. Support-vector networks. Machine Learning 20 (3), 273–297.
URL http://dx.doi.org/10.1007/BF00994018

[10] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., Reis, J., 2009. Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems 47 (4), 547 – 553.

[11] Dempster, A. P., Laird, N. M., Rubin, D. B., 1977. Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological) 39 (1), 1–38.
URL http://www.jstor.org/stable/2984875

[12] Denis, F., Gilleron, R., Letouzey, F., 2005. Learning from positive and unlabeled examples. Theoretical Computer Science 348 (1), 70 – 83.
URL http://www.sciencedirect.com/science/article/pii/S0304397505005256

[13] Domingos, P., Hulten, G., 2000. Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '00. ACM, New York, NY, USA, pp. 71–80.
URL http://doi.acm.org/10.1145/347090.347107

[14] Flach, P., 2012. Machine Learning: The Art and Science of Algorithms That Make Sense of Data. Cambridge University Press, New York, NY, USA.

[15] Gurobi Optimization, Inc., 2015. Gurobi optimizer reference manual.
URL http://www.gurobi.com

[16] Jaccard, P., 1912. The distribution of the flora in the alpine zone. New Phytologist 11 (2), 37–50.

[17] Karp, R., 1972. Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (Eds.), Complexity of Computer Computations. Plenum Press, pp. 85–103.

[18] Khan, S. S., Madden, M. G., 2013. One-class classification: Taxonomy of study and review of techniques. CoRR abs/1312.0049.
URL http://dblp.uni-trier.de/db/journals/corr/corr1312.html#KhanM13

[19] Lance, G. N., Williams, W. T., 1967. Mixed-data classificatory programs i - agglomerative systems. Australian Computer Journal 1 (1), 15–20.
URL http://dblp.uni-trier.de/db/journals/acj/acj1.html#LanceW67

[20] Li, C., Zhang, Y., Li, X., 2009. Ocvfdt: One-class very fast decision tree for one-class classification of data streams. In: Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data. SensorKDD '09. ACM, New York, NY, USA, pp. 79–86.
URL http://doi.acm.org/10.1145/1601966.1601981

[21] Lloyd, S., Sep. 2006. Least squares quantization in pcm. IEEE Trans. Inf. Theor. 28 (2), 129–137.
URL http://dx.doi.org/10.1109/TIT.1982.1056489

[22] Montana, D. J., 1995. Strongly typed genetic programming. Evolutionary Computation 3 (2), 199–230.
URL http://vishnu.bbn.com/papers/stgp.pdf

[23] Pawlak, T. P., Krawiec, K., 2017. Automatic synthesis of constraints from examples using mixed integer linear programming. European Journal of Operational Research 261 (3), 1141 – 1157.
URL http://www.sciencedirect.com/science/article/pii/S037722171730156X

[24] Pawlak, T. P., Krawiec, K., 2017. Synthesis of constraints for mathematical programming with one-class genetic programming. IEEE Transactions on Evolutionary Computation(in review).

[25] Pawlak, T. P., Krawiec, K., 19-21 Apr. 2017. Synthesis of mathematical programming constraints with genetic programming. In: Castelli, M., McDermott, J., Sekanina, L. (Eds.), EuroGP 2017: Proceedings of the 20th European Conference on Genetic Programming. Vol. 10196 of LNCS. Springer Verlag, Amsterdam, pp. 178–193.

[26] Quinlan, J. R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[27] Simonoff, J., 1996. Smoothing Methods in Statistics. Springer Series in Statistics. Springer.
URL https://books.google.pl/books?id=wFTgNXL4feIC

[28] Souza, C. R., May 2017. The Accord.NET Framework.
URL http://accord-framework.net

[29] Tax, D. M. J., 2001. One-class classification: Concept-learning in the absence of counter-examples. Ph.D. thesis, Delft University of Technology.

[30] Teso, S., Sebastiani, R., Passerini, A., 2017. Structured learning modulo theories. Artificial Intelligence 244, 166 – 187.

[31] Williams, H., 2013. Model Building in Mathematical Programming. Wiley.