Learning Linear Programs from Data

Elias Arnold Schede
KU Leuven
Leuven, Belgium
eliasarnold.schede@student.kuleuven.be

Samuel Kolb
KU Leuven
Leuven, Belgium
samuel.kolb@cs.kuleuven.be

Stefano Teso
KU Leuven
Leuven, Belgium
stefano.teso@cs.kuleuven.be

***Abstract*—Linear Programming lies at the core of mathematical modelling and optimization. Designing linear programs (LPs) is a difficult and expensive process, as it requires both mathematical programming and domain expertise, and it involves both designing an objective function and feasibility constraints. To support this design process, we propose INCALP, an algorithm for inducing linear programs from examples. Since the objective can often be learned with standard techniques (e.g. regression), INCALP learns the hard constraints only. It does so by encoding constraint learning as a mixed integer linear program. INCALP achieves significant efficiency gains by considering gradually larger subsets of examples, and terminating as soon as a suitable program is found. In addition, INCALP encourages both compactness and sparsity of the learned program. Our empirical analysis on synthetic data and textbook problems highlights the promise of the approach.**

***Index Terms*—machine learning, operations research, constraint learning, linear programming**

I. INTRODUCTION

Linear programming is the backbone of mathematical modelling and optimization; its applications in science and business are too many to list [1]. Despite its popularity, designing linear programs (LPs) by hand is an arduous and time consuming task [2] as it requires both domain knowledge and modelling expertise. In line with research on constraint learning [3] and programming by example [4], we propose to facilitate the design step by acquiring LPs from data. Doing so involves inducing the objective function and the feasibility constraints. Since the first task can often be tackled with standard machine learning techniques (as explained below), we focus on the much harder task of acquiring linear constraints from examples of feasible and infeasible solutions.

Most existing approaches are not immediately applicable to learning LPs. Methods based on constraint learning [5] and syntax-guided synthesis [6] do not consider the case of continuous variables, while machine learning methods for inducing convex polytopes [7], [8] and unions thereof [9] often rely on greedy heuristics which may fail to identify a correct LP. Finally, inverse optimization methods [10], [11] assume a good initial guess for the LP to be available.

Recently, however, two suitable approaches have emerged: INCAL [12] and the method of Pawlak and Krawiec [2], which we indicate as MILPCS. INCAL induces Satisfiability Modulo Linear Real Arithmetic (SMT($\mathcal{LRA}$), or SMT for short) formulas from data. SMT extends propositional logic by allowing continuous variables and linear inequalities over them, and subsumes linear programming as a special case [13]. INCAL encodes the learning step in terms of SMT($\mathcal{LRA}$) satisfiability and applies an efficient off-the-shelf solver to obtain a formula compatible with all examples. Similarly, MILPCS induces linear, quadratic, and trigonometric constraints over continuous variables by solving a mixed-integer linear program (MILP), i.e., an LP where some variables are required to be integral.

INCAL and MILPCS have complementary strengths and weaknesses: a) INCAL sports an efficient, exact incremental strategy for learning formulas while only looking at a small (active) subset of examples, while MILPCS always considers all examples; b) INCAL encodes the learning problem as an SMT problem, while MILPCS encodes it as a MILP one, which is a better fit for the problems with a mostly continuous nature like LP learning; c) INCAL favours learning compact but dense formulas, while MILPCS encourages learning sparser programs instead – but both size and sparsity are important for learning interpretable programs.

We introduce INCALP, a non-greedy learning algorithm for LPs that merges the incremental, compactness-inducing strategy of INCAL with the efficient, sparsity-inducing MILP encoding of MILPCS. INCALP automatically uncovers LPs that are both compact and sparse, while simultaneously improving upon the deficiencies of INCALP and MILPCS. In addition, we further improve the incremental learning strategy through the use of a new heuristic (SELECTDT), which uses a decision tree to model the importance of examples to be added to the active set. This heuristic can substantially improve the run-time of INCALP without affecting the correctness of the learned LP.

To summarize, we make the following contributions: 1) INCALP, a non-greedy LP learner that combines the advantages of INCAL and MILPCS to obtain substantial efficiency improvements while learning interpretable constraints. 2) SELECTDT, a new heuristic for exact incremental learning based on a decision tree surrogate of the decision boundary. 3) An extensive comparison on benchmark LPs, textbook examples, and synthetic instances, showing that INCALP can find LPs of comparable or better quality than competing approaches faster.

II. RELATED WORK

Our work is inspired by efforts in constraint learning [3], [5], [14] and programming by example [4] to automate the de-

This work has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No [694980] SYNTH: Synthesising Inductive Data Models). Samuel Kolb is supported by the Research Foundation-Flanders (FWO).

sign of constraint theories and programs. Existing approaches in these areas focus on models over discrete variables only, and cannot be straightforwardly adapted to learning LPs.

Most machine learning methods for inducing linear constraints (convex polytopes) from data follow a pragmatic approach and employ greedy or approximate learning schemas, often based on stochastic gradient descent [7], [8]. Similarly, techniques for learning oblique decision trees (which include arbitrary linear separators at the inner nodes and are strictly more general than convex polytopes) either follow a greedy top-down induction strategy [15], [16] or approximately minimize a complex global loss function [9]. The downside of these heuristic procedures is that they can get stuck in local optima, producing inaccurate models that do not fit the training examples—even when these are noiseless.

The most common setup for exact approaches is syntax-guided learning [6]. In this setting, the learning problem is cast as a global satisfaction or optimization problem, like SMT($\mathcal{LRA}$) or MILP, and solved in practice with an appropriate solver. This strategy is more computationally expensive than the alternative, but it guarantees that the learned model and the examples match. INCALP combines and improves upon two methods that follow this setup: INCAL [12] and MILPCS [2]. These two methods are discussed in detail in Section III. Another exact method was proposed in [17] for learning optimal decision trees (ODTs). Compared to INCALP, however, it uses a more complex MILP formulation (suitable for learning ODTs) and does not support incremental learning. Syntax-guided synthetis approaches can also make use of heuristic search techniques (such as evolutionary algorithms [18]), but the result lose all exactness guarantees in this case.

Inverse optimization (IO) is also very related. Given an input optimization problem—possibly an LP—and a set of configurations, the goal of IO is to minimally perturb the problem so that the configurations become optimal [19], [20]. Despite the similarities to LP learning, IO assumes a source problem close to the intended result to be available. Moreover, while $\boldsymbol{c}$ and $\boldsymbol{b}$ are adapted to the data, the coefficient matrix A is not [10], [11], [21]. Approaches to linear system identification do not have these limitation, but they are limited to two-dimensional problems [22] or non-negative coefficients [23] only.

III. BACKGROUND

A. Linear programming and extensions

A linear program $\mathcal{P}$ over n real variables $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x} \geq 0$, consists of a linear objective function and m linear feasibility constraints [1]. Letting $\boldsymbol{x} \cdot \boldsymbol{z}$ indicate the dot product $\sum_i x_i z_i$, any LP can be written in canonical form as:

$$\max_{\boldsymbol{x}} \ \boldsymbol{c} \cdot \boldsymbol{x} \tag{1}$$

$$\text{s.t. } \boldsymbol{a}_j \cdot \boldsymbol{x} \leq b_j \qquad j = 1, \ldots, m \tag{2}$$

Here $\boldsymbol{c} \in \mathbb{R}^n$ defines the objective function, while $\boldsymbol{a}_j \in \mathbb{R}^n$ and $b_j \in \mathbb{R}$, for $j = 1, \ldots, m$, encode the feasibility constraints. The set of feasible assignments for $\mathcal{P}$ is the polytope $\mathcal{X}(\mathcal{P}) = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \leq \boldsymbol{b}\}$, where $A = [\boldsymbol{a}_1; \ldots; \boldsymbol{a}_m] \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} = (b_1, \ldots, b_m)$. It is well known that linear programming is solvable in polynomial time [24] and very large LPs are routinely solved in practice.

Mixed-integer linear programming (MILP) generalizes linear programming by allowing some of the variables to be integral. MILP is a very popular framework for modelling and solving complex combinatorial problems. Despite being NP-complete, the importance of MILP has led to the developement of industrial-grade solvers capable of efficiently solving large instances, like Gurobi [25] and CPLEX [26].

B. Learning SMT formulas with INCAL

Satisfiability Modulo Linear Real Arithmetic (abbreviated as SMT($\mathcal{LRA}$) or simply as SMT) is a formalism that mixes propositional logic together with linear constraints [13]. An SMT formula includes both Boolean and continuous variables and combines Boolean atoms with linear constraints through standard logical connectives (conjunction, disjunction, negation, *etc.*). SMT formulas can capture arbitrary unions of polytopes, and are thus strictly more expressive than conjunctions of linear constraints. As in propositional logic, satisfaction in SMT amounts to finding a variable assignment that makes a target formula true and is NP-complete, but efficient practical solvers are available.

INCAL is a non-greedy algorithm for inducing SMT formulas from examples [12]. Given a set of feasible and infeasible variable assignments and a target formula complexity, INCAL finds an SMT formula φ (in conjunctive or disjunctive normal form) of the given complexity that correctly classifies all examples. The formula complexity (k, h) specifies a maximum number of clauses k and unique linear inequalities h.

The learning step itself is encoded as an SMT problem and solved with an SMT solver; see [12] for the details. The runtime of this procedure depends on the size of the encoding (among other factors), which is proportional to the number of training examples. Thus, rather than encoding the entire data-set, INCAL employs an efficient incremental learning procedure whereby a formula is gradually adapted to larger and larger subsets of the data. This strategy solves a sequence of smaller problems and often finds a valid formula by only looking a small subset of the data. Importantly, INCAL is *exact*: it always finds a formula consistent with all the examples, provided that one exists.

INCAL also implements a non-parametric search loop that gradually increases the target complexity (k, h), starting from $(1, 1)$, and attempts to induce a formula at each step. The procedure terminates as soon as a suitable formula can be learned, thus directly minimizing the complexity of the formula. This procedure exploits the fact that too-small complexity parameters can be identified and discarded quickly.

C. Learning constraints with MILPCS

The MILPCS algorithm [2] uses an analogous strategy for learning linear programs, but it relies on a MILP optimization formulation rather than SMT. The objective function aims

at synthesizing sparse models where the coefficients are either zero or (close to) one, for improved generalization and interpretability. In addition to linear programs, MILPCS can also learn non-linear programs by mapping the configurations to higher dimensional feature space via a fixed set of basis functions, e.g., polynomials or sinusoids. (Notice that the same feature transform could be applied to INCAL and INCALP too.)

The SMT encoding of INCAL is suitable for learning SMT models, which often are very combinatorial, while the encoding of MILPCS is often more efficient for learning "more continuous" problems such as LPs. Further, it can naturally uncover sparse models, which occur often in practical applications. The major difference, however, is that MILPCS does not include an incremental learning strategy, and therefore it has more trouble scaling to larger data-sets.

IV. Learning Linear Programs

We consider a fully supervised setting where the goal is to recover, either exactly or approximately, an unknown linear program $\mathcal{P}^*$ (with parameters $\boldsymbol{c}^*$, A^*, and $\boldsymbol{b}^*$) from data. More formally, our learning problem is as follows: ***Given*** *a set of instances* $\{\boldsymbol{x}^k \in \mathbb{R}^n\}_{k=1}^s$ *labelled by whether they are feasible* ($y^k = 1$) *or not* ($y^k = 0$) *with respect to an unknown LP* $\mathcal{P}^*$, ***find*** *an LP* $\mathcal{P}$ *that well approximates* $\mathcal{P}^*$.

This task involves acquiring both the objective function and the linear constraints of $\mathcal{P}^*$. These two steps can be tackled independently. We discuss them in turn.

A. Learning the objective function

Estimating the parameters of an unknown linear objective function is a common task in machine learning. For instance, in settings like portfolio optimization, the feasible configurations $\boldsymbol{x}_i$ may be annotated by their objective value $\boldsymbol{c}^* \cdot \boldsymbol{x}_i$, in which case $\boldsymbol{c}^*$ can be readily estimated via linear regression. In applications such as preference learning [27], the supervision consists of rankings among alternative configurations, which implicitly reveal information about the gradient the objective function. In this case, $\boldsymbol{c}^*$ can be reconstructed via learning-to-rank. Given the large range of approaches, we focus exclusively on the more challenging problem of learning the linear constraints themselves.

B. Learning the linear constraints

Given a data-set of positive and negative examples $\mathcal{D} = \{(\boldsymbol{x}^k, y^k)\}_{k=1}^s$, learning linear constraints amounts to finding a polytope $\mathcal{X}(\mathcal{P})$, defined by parameters $A \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$, such that all the feasible examples are inside $\mathcal{X}(\mathcal{P})$ and none of the infeasible ones is. Recall that, by definition, an instance $\boldsymbol{x}$ lies inside of $\mathcal{X}(\mathcal{P})$ if and only if it satisfies all the constraints $\boldsymbol{a}_j \cdot \boldsymbol{x} \leq b_j$, $j = 1, \ldots, m$.

INCALP encodes the learning task using a simplified version of the MILP encoding used by MILPCS[1]. Recall that the index $i = 1, \ldots, n$ runs over the variables, $j = 1, \ldots, m$ over the constraints, and $k = 1, \ldots, s$ over the examples, and let x_i^k be the value assigned by $\boldsymbol{x}^k$ to the ith variable, M be a large enough real scalar, and ϵ be a small enough one. Then, the INCALP learning problem can be written as:

$$\min_{A,\boldsymbol{b}} \ \sum_{i,j} z^a_{j,i} + \sum_j z^b_j \tag{3}$$

$$\text{s.t. } \boldsymbol{a}_j \cdot \boldsymbol{x}^k \leq b_j \qquad \forall j,k : y^k = 1 \tag{4}$$

$$\sum_j v_{k,j} \geq 1 \qquad \forall k : y^k = 0 \tag{5}$$

$$\boldsymbol{a}_j \cdot \boldsymbol{x}^k \geq M v_{k,j} - M + b_j + \epsilon \quad \forall j,k : y^k = 0 \tag{6}$$

$$\sum_i z^a_{j,i} \geq z^b_j \qquad \forall j \tag{7}$$

$$-a_{max} z^a_{j,i} \leq a_{j,i} \leq a_{max} z^a_{j,i} \qquad \forall i,j \tag{8}$$

$$-b_{max} z^b_j \leq b_j \leq b_{max} z^b_j \qquad \forall j \tag{9}$$

Let us look at the encoding in detail. The output are m linear constraints represented by the decision variables A and $\boldsymbol{b}$. Here $m \in \mathbb{N}$, $a_{max} \in \mathbb{R}_{\geq 0}$ and $b_{max} \in \mathbb{R}_{\geq 0}$ are user-provided hyperparameters that limit the complexity of the polytope and bound its coefficients. Eq. 4 ensures that the polytope covers all feasible examples. The auxiliary variables $v_{k,j} \in \{0, 1\}$ force the kth example to violate the jth linear constraint: Eq. 5–6 ensure that every infeasible example ($y^k = 0$) violates at least one of the learned linear constraints. A second set of auxiliary variables $z^a_{j,i} \in \{0, 1\}$ and $z^b_j \in \{0, 1\}$ indicate whether the corresponding parameters $a_{j,i}$ and b_j are actively used in the learned polytope and allowed to be non-zero, as per Equations 8 and 9. Eq. 7 ensures that $z^a_{j,i}$ and z^b_j are compatible, i.e., that if b_j non-zero then at least one coefficient of $\boldsymbol{a}_j$ is allowed to be non-zero. Finally, the cost function in Eq. 3 minimizes the total number of non-zero parameters used by the learned constraints. From a machine learning perspective, this acts as an ℓ_1 regularizer and encourages finding sparser models [28]. Of course, this component can be optionally turned off if the target LP is known to be dense.

Like INCAL and MILPCS, we assume the labels to be noiseless. This makes sense since for business processes, feasible (working) and infeasible (non-working) configurations are often easily distinguishable and well documented. Under this assumption, a suitable polytope is always found, provided that m, a_{max} and b_{max} are large enough.

C. The INCALP algorithm

Learning LPs by naively encoding the whole training set, like MILPCS does, may produce a very large and difficult to solve MILP encoding. In order to avoid this issue, INCALP employs and improves the incremental learning strategy of INCAL [12], as follows.

Instead of encoding all examples at once, INCALP iteratively adds examples in steps to control the complexity of the encoding, see Algorithm 1. For a set number of allowed half-spaces m, an initial (active) set of examples (20 in our experiments) is chosen (line 3) and encoded (line 6). The encoding (see Eq. 3–9) is solved using a MILP solver and the obtained polytope $A_i, \boldsymbol{b}_i$ is tested against the not yet encoded examples. If no examples are violated, INCALP managed to find a polytope that is consistent with all examples and

[1]The differences being that INCALP: a) only considers linear basis functions and b) does not include a term encouraging the coefficients to be close to 1; the latter had no effect in our experiments and so it was dropped.

Algorithm 1 The INCALP algorithm: m is the number of constraints, $\mathcal{D}$ are the examples, and θ is the decision tree.

procedure LEARNINCREMENTAL($m, \mathcal{D}, \theta$)
  $i \leftarrow 1$
  $\mathcal{D}_i \leftarrow$ SELECTDT($\mathcal{D}, \theta, 20$)
  $\mathcal{V}_i \leftarrow$ all misclassified examples in $\mathcal{D} \setminus \mathcal{D}_i$
  while $\mathcal{V}_i$ is not empty **do**
    $A_i, \boldsymbol{b}_i \leftarrow$ SOLVE(ENCODE($m, \mathcal{D}_i$)) ▷ Eq. 3–9
    if could not find A_i, $\boldsymbol{b}_i$ consistent with $\mathcal{D}_i$ **then**
      return *infeasible*
    $\mathcal{V}_i \leftarrow$ all misclassified examples in $\mathcal{D} \setminus \mathcal{D}_i$
    $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \cup$ SELECTDT($\mathcal{V}_i, \theta, 1$)
    $i \leftarrow i + 1$
  return $A_i, \boldsymbol{b}_i$

returns the acquired constraints. Otherwise, one of the violated examples is added to the active set and the loop repeats.

The selection of initial examples and which violated examples to add at each iteration can be done in different ways. INCAL chooses a random point from the set of all violating examples $\mathcal{V}_i$. In contrast, INCALP prioritizes informative violating examples that lie close to the boundary between feasible and infeasible regions. Since the boundary is unknown, we introduce a novel data-driven heuristic SELECTDT that approximates the real decision surface with a decision tree θ learned on all examples $\mathcal{D}$ in an offline pre-processing step. Given a set of violating examples $\mathcal{V}_i$ to select from, SELECTDT choose one of the examples closest to the decision surface of θ. It does so by choosing two points $\boldsymbol{x} \in \mathcal{D}'$ and $\boldsymbol{x}'$ are chosen so that they are as close as possible and such that θ assigns them to opposite classes. So long as the target distance is the L_0, L_1, or L_∞, this problem can be solved efficiently using MILP. In our experiments, we opt for the L_1 distance. Notice that this technique amounts to summarizing the data-set with a surrogate model, and is different from, and more efficient than, interactive learning techniques like active learning.

The above incremental procedure requires the user to provide an appropriate number of half-spaces m, which requires problem knowledge. Here, we borrow from the parameter-free setting of INCAL and implement a bottom up search strategy (similar to iterative deepening) on top of the incremental learning procedure that initializes $m = 1$, runs the learning algorithm, and increases m by 1 as long as no program can be found, see Algorithm 2. This ensures that a valid model is found with the smallest number of constraints, leading to more compact models. Domain knowledge about the approximate value of m can still be used to bootstrap the procedure.

D. Discussion

Just like MILPCS, INCALP can in principle be be used together with a non-linear (e.g. quadratic or higher-order) feature map for learning non-linear constraints. INCALP can also straightforwardly learn mixed-integer linear programs, the main difference being that some of the variables x_i are forced to be integral. However, in this paper we consider learning LPs only, because they are by far the most widespread class of models in mathematical optimization. Another interesting research direction involves learning from feasible-only examples, for applications where infeasible examples are hard to obtain. This can be easily achieved by adapting the techniques of [29]. We postpone such extensions to future work.

Algorithm 2 The non-parametric INCALP algorithm: $\mathcal{D}$ are the examples.

procedure LEARNNOPARAMS($\mathcal{D}$)
  $m \leftarrow 1$
  $\theta \leftarrow$ LearnDT($\mathcal{D}$)
  while *true* **do**
    $A_i, \boldsymbol{b}_i \leftarrow$ LEARNINCREMENTAL($m, \mathcal{D}, \theta$)
    if could not find A_i, $\boldsymbol{b}_i$ **then**
      $m \leftarrow m + 1$
    else
      return $A_i, \boldsymbol{b}_i$

V. EMPIRICAL ANALYSIS

We empirically study the following research questions: (**Q1**) Are the LPs learned by INCALP accurate and useful? (**Q2**) Is INCALP more efficient than the competitors? (**Q3**) Does the SELECTDT heuristic improve the run-time of incremental learning? (**Q4**) How does INCALP scale w.r.t the number of constraints, variables and examples? To do so, we use INCALP to recover a known LP $\mathcal{P}^*$ from random feasible and infeasible solutions thereof, and compare it against the two state-of-the-art methods it is derived from: MILPCS and a variant of INCAL. We also compare it against a specialization of INCALP to LPs, dubbed INCALP(SMT). This variant uses the following simplified SMT encoding of the learning problem tuned for LPs (i.e., allowing only one literal per clause and removing redundant variables):

$$\text{ENCODE}(m, \mathcal{D}) \stackrel{\text{def}}{=} \bigwedge_k \left(y^k \iff \bigwedge_{j=1}^{m} (\boldsymbol{a}_j \cdot \boldsymbol{x}^k \leq b_j) \right)$$

This variant also uses the SELECTDT heuristic. Experiments were run on an 8-thread, 3.40 GHz Intel CPU.

A. Benchmark problems

Our comparisons are carried out on three groups of LP instances, which we describe in turn.

Simplex_n and Cube_n are simple LP polytopes used for benchmarking LP learning in [2], [18], [30]. Simplex_n is the n-dimensional simplex, and includes $n(n-1)$ linear constraints with two variables each, plus one constraint over all variables. Cube_n is the n-dimensional cube defined by $2n$ linear constraints of one variable each. We consider $n = 2, \ldots, 4$.

We also look at *police* and *pollution*, two problems from a textbook on Linear Programming [31]. Police is a scheduling problem where the goal is to minimize the total cost spent on police agents over a day. Each day is divided into 10

consecutive time periods. For each time period the minimum number of active agents is given. There are 5 different shifts an agent can begin work, and every agent should not work longer than 4 consecutive periods. In addition, we impose an upper bound of 200 agents per shift. The LP consists of 10 sparse constraints (three of which are redundant) over 5 variables that represent the number of agents assigned to one of the 5 shifts. Pollution aims to reduce pollution output by two production processes while minimizing the cost. Three different methods can be used to lower the pollution. This amounts to six decision variables indicating the fraction of the abatement capacity between reduction method and process. Each process has an output of three different pollutants, which are reduced by different amounts by different choices. A certain minimum reduction of all three pollutants has to be archived. The formulations are not reported for lack of space.

To study the effect of increasing the number of variables n and constraints m, we synthetically generate LP problems with m constraints over n variables as follows: Given d uniformly points sampled from the unit bounding box $[0, 1]^n$, we 1) compute their convex hull, 2) reject if the convex hull has less than m constraints, 3) randomly drop all but m of the constraints defining the convex hull, and 4) reject if the ratio of positive-to-negative examples is below 0.3 or above 0.7.

B. Performance measures

Aside from the computation time, we want to measure how accurately the constraints of $\mathcal{P}^*$ were recovered. To this end we estimate the *true positive rate* and *true negative rate* using a test set of 10^6 points that are uniformly drawn from the hypercube defined by the bounds of the problem variables. The true negative rate is the probability that a feasible point of $\mathcal{P}^*$ lays in the feasible region of the learned program $\mathcal{P}$, while the true negative rate is the probability that an infeasible point of $\mathcal{P}^*$ lays in the infeasible region of $\mathcal{P}$. Given an objective function, we also compare the difference between between the optimum value of $\mathcal{P}^*$ and the optimum value of $\mathcal{P}$.

To measure the performance of Cube_n and $\mathrm{Simplex}_n$, as well as the textbook problems, we uniformly draw d positive and d negative examples and obtain learned programs $\mathcal{P}$ by feeding the examples to the various learning algorithms. We vary the number d of examples to measure its effect on the computation time and performance. On these problems, we always average computation time and perfomance over 10 independent runs. For the synthetic problems, we use a data-set of d uniformly samples points and generate 10 different problems per choice of (n, m). INCALP used Gurobi [25] with $\epsilon = 1$, $M = 10^6$ and a_{max}, $b_{max} = 1000$. INCALP(SMT) on the other hand used the Z3 solver [32]. Timeouts are $5400s$ for benchmark and textbook problems, $200s$ for synthetic ones. Timed-out approaches are set to the time-out value in average run-times.

C. Are the LPs learned by INCALP accurate and useful?

Our analysis of the TPR and TNR shows that INCALP accurately recovers the constraints on both the benchmark (Figure 3) and textbook problems (Figure 4). The MILP encoding allows INCALP to perform better than INCALP(SMT) on problems with sparse constraints (e.g., police). This is also confirmed by the analysis of the optimal values for the learned and the true models (Figure 2). A qualitative analysis shows that for the police problem, INCALP accurately captures the sparsity and the pattern of uni- and multi-variate constraints.

D. Is INCALP more efficient than the competitors?

Across the example problems we tested, we found that INCALP significantly outperforms MILPCS, especially when given larger numbers of examples and higher dimensional problems (Figures 3 and 4). This speed-up occurs largely because only a fraction of the available examples is ever encoded by the incremental algorithms (illustrated in Figure 2 bottom-left). INCALP(SMT) also profits from the incremental scheme, however, it struggles to exploit sparsity in problems (e.g., Cube_n, police). Note that Gurobi, unlike Z3, can exploit multiple cores.

E. Does the SELECTDT heuristic improve run-time?

We compared the SELECTDT heuristic to random selection as employed by INCAL [12] on the textbook problems. The decision tree heuristic substantially improves the run-time by selecting more informative violated examples, while obtaining similar TPR and TNR (Figure 6).

F. How does INCALP scale?

Our synthetic experiments show that INCALP can scale up to considerable numbers of constraints but suffers from the curse of the dimensionality, as more dimensions impact the run-times considerably. Due to its incremental scheme, INCALP also scales well w.r.t. the number of examples to learn from (Figure 5).

$$
\begin{array}{ccccccccccc}
4.1x_1 & & & & & & & & & & \geq 1 \\
2.1x_1 & + & 3.1x_2 & & & & & & & & \geq 1 \\
 & & 3.2x_2 & + & 2.9x_3 & & & & & & \geq 1 \\
 & & & & 2.3x_3 & + & 2.3x_4 & & & & \geq 1 \\
 & & & & & & 4.6x_4 & & & & \geq 1 \\
 & & & & & & & & 13.1x_5 & & \geq 1
\end{array}
$$

Fig. 1. LP learned for the police problem.

VI. CONCLUSION

We presented INCALP, a novel non-greedy algorithm for learning LPs from examples of feasible and infeasible solutions. INCALP extends and combines the efficient MILP encoding of MILPCS and the fast incremental learning strategy of INCAL. The algorithm optimizes for both compactness and sparsity of the learned LP, with the aim of enhancing generalization and interpretability. In addition, INCALP exploits a new decision tree-based example selection heuristic, further speeding up the learning process without affecting its correctness. Our experiments show that INCALP can quickly and effectively retrieve realistic and synthetic instances from examples.

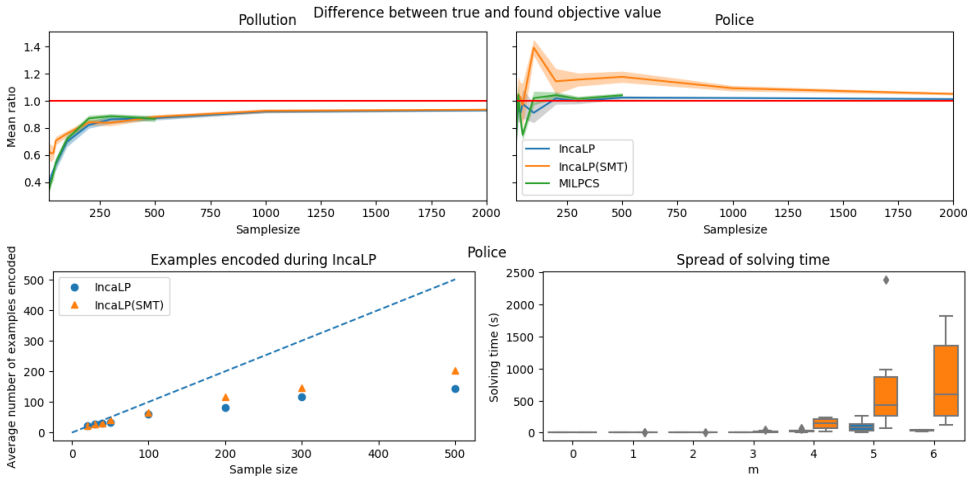

Fig. 2. Optimizing the models learned by INCALP and MILPCS achieves values close to the true optimal value (top). INCALP and INCALP(SMT) are able to learn constraints faster because they only add few examples to the encoding before finding a solution (bottom-left). Analyzing their run-time more closely reveals that when increasing m, the total run-time is dominated by the last m values.

REFERENCES

[1] G. Dantzig, *Linear programming and extensions*. Princeton university press, 2016.

[2] T. P. Pawlak and K. Krawiec, "Automatic synthesis of constraints from examples using mixed integer linear programming," *European Journal of Operational Research*, vol. 261, no. 3, pp. 1141–1157, 2017.

[3] L. De Raedt, A. Passerini, and S. Teso, "Learning constraints from examples," in *Proceedings of AAAI'18*, 2018.

[4] S. Gulwani, "Automating string processing in spreadsheets using input-output examples," in *ACM SIGPLAN Notices*, vol. 46, no. 1. ACM, 2011, pp. 317–330.

[5] C. Bessiere *et al.*, "New approaches to constraint acquisition," in *Data mining and constraint programming*. Springer, 2016, pp. 51–76.

[6] R. Alur, R. Bodik, G. Juniwal, M. M. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa, "Syntax-guided synthesis," in *2013 Formal Methods in Computer-Aided Design*. IEEE, 2013, pp. 1–8.

[7] N. Manwani and P. Sastry, "Polyceptron: A polyhedral learning algorithm," *arXiv preprint arXiv:1107.1564*, 2011.

[8] A. Kantchelian, M. C. Tschantz, L. Huang, P. L. Bartlett, A. D. Joseph, and J. D. Tygar, "Large-margin convex polytope machine," in *Advances in Neural Information Processing Systems*, 2014, pp. 3248–3256.

[9] M. Norouzi, M. Collins, M. A. Johnson, D. J. Fleet, and P. Kohli, "Efficient non-greedy optimization of decision trees," in *Advances in Neural Information Processing Systems*, 2015, pp. 1729–1737.

[10] A. Bärmann, S. Pokutta, and O. Schneider, "Emulating the expert: inverse optimization through online learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 400–410.

[11] C. Dong, Y. Chen, and B. Zeng, "Generalized inverse optimization through online learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 86–95.

[12] S. Kolb, S. Teso, A. Passerini, and L. De Raedt, "Learning SMT(LRA) constraints using SMT solvers," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, vol. 104, 2018, pp. 2067–2073.

[13] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," *Handbook of satisfiability*, vol. 185, pp. 825–885, 2009.

[14] N. Beldiceanu and H. Simonis, "A model seeker: Extracting global constraint models from positive examples," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2012, pp. 141–157.

[15] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *Journal of artificial intelligence research*, vol. 2, pp. 1–32, 1994.

[16] N. Manwani and P. Sastry, "A geometric algorithm for learning oblique decision trees," in *International Conference on Pattern Recognition and Machine Intelligence*. Springer, 2009, pp. 25–31.

[17] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, no. 7, pp. 1039–1082, 2017.

[18] T. P. Pawlak and K. Krawiec, "Synthesis of constraints for mathematical programming with one-class genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 117–129, 2019.

[19] R. K. Ahuja and J. B. Orlin, "Inverse optimization," *Operations Research*, vol. 49, no. 5, pp. 771–783, 2001.

[20] L. Wang, "Cutting plane algorithms for the inverse mixed integer linear programming problem," *Operations Research Letters*, vol. 37, no. 2, pp. 114–116, 2009.

[21] S. Jabbari, R. M. Rogers, A. Roth, and S. Z. Wu, "Learning from rational behavior: Predicting solutions to unknown linear programs," in *Advances in Neural Information Processing Systems*, 2016, pp. 1570–1578.

[22] M. D. Troutt, S. K. Tadisina, C. Sohn, and A. A. Brandyberry, "Linear programming system identification," *European journal of operational research*, vol. 161, no. 3, pp. 663–672, 2005.

[23] M. D. Troutt, A. A. Brandyberry, C. Sohn, and S. K. Tadisina, "Linear programming system identification: The general nonnegative parameters case," *European Journal of Operational Research*, vol. 185, no. 1, pp. 63–75, 2008.

[24] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM, 1984, pp. 302–311.

[25] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2018. [Online]. Available: http://www.gurobi.com

[26] IBM, "CPLEX Home Page," 2018. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[27] G. Pigozzi, A. Tsoukias, and P. Viappiani, "Preferences in artificial intelligence," *Annals of Mathematics and Artificial Intelligence*, vol. 77, no. 3-4, pp. 361–401, 2016.

[28] T. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[29] P. Kudła and T. P. Pawlak, "One-class synthesis of constraints for mixed-integer linear programming with c4. 5 decision trees," *Applied Soft Computing*, vol. 68, pp. 1–12, 2018.

[30] T. P. Pawlak and K. Krawiec, "Synthesis of mathematical programming constraints with genetic programming," in *European Conference on Genetic Programming*. Springer, 2017, pp. 178–193.

[31] G. J. Lieberman and F. Hillier, *Introduction to mathematical programming*. McGraw-Hill, 1995.

[32] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

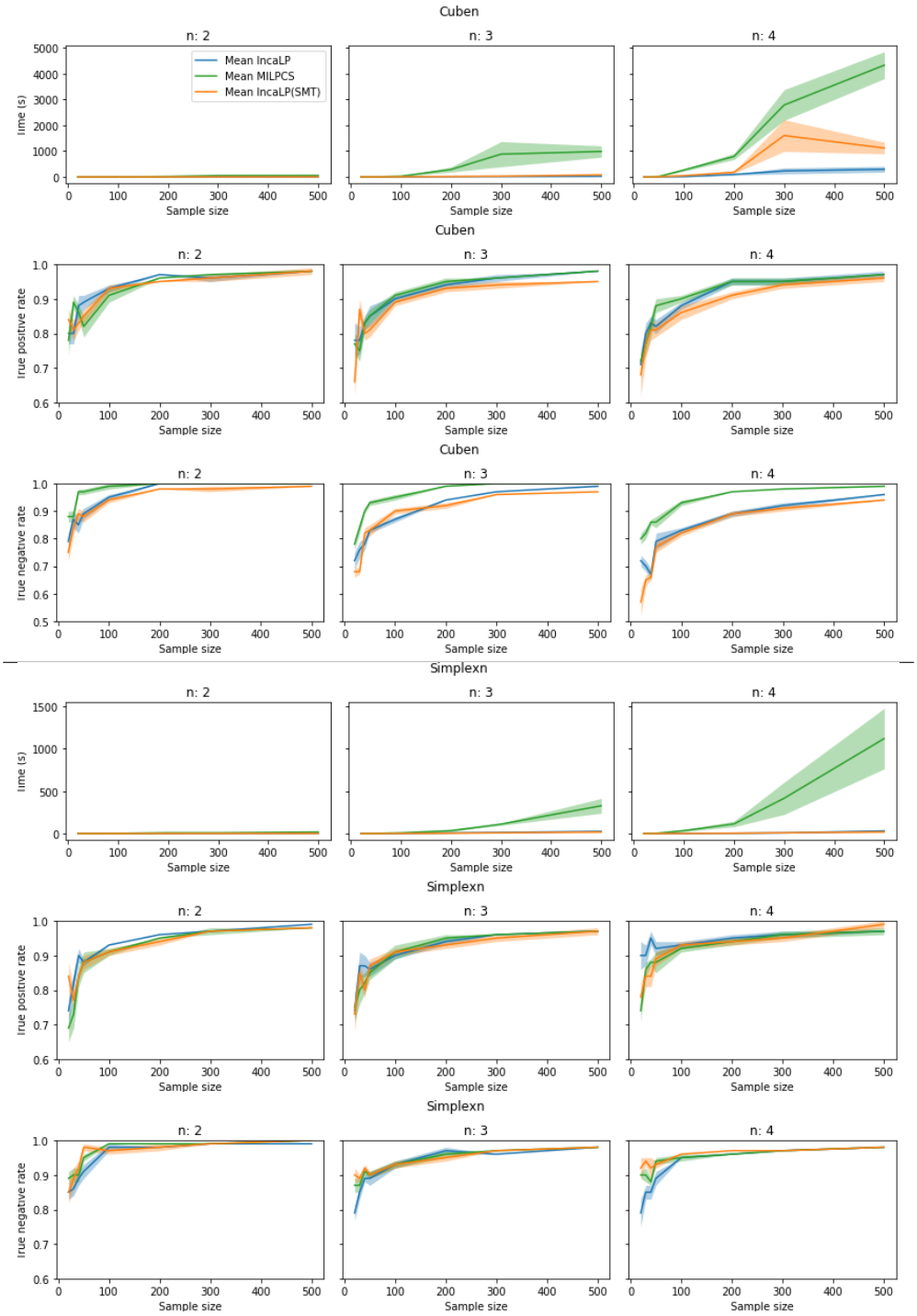

Fig. 3. Our results show that INCALP achieves good accuracy while being significantly faster than MILPCS and outperforming INCALP(SMT) on the Cube_n (top 3 rows) and $\mathrm{Simplex}_n$ (bottom 3 rows). The plots show run-time, TPR and TNR (rows) for different n (columns). For Cube_4, INCALP(SMT) timed out on 1 run (300 samples) and 1 run (500 samples), MILPCS on 1 run (300 samples) and 7 runs (500 samples).

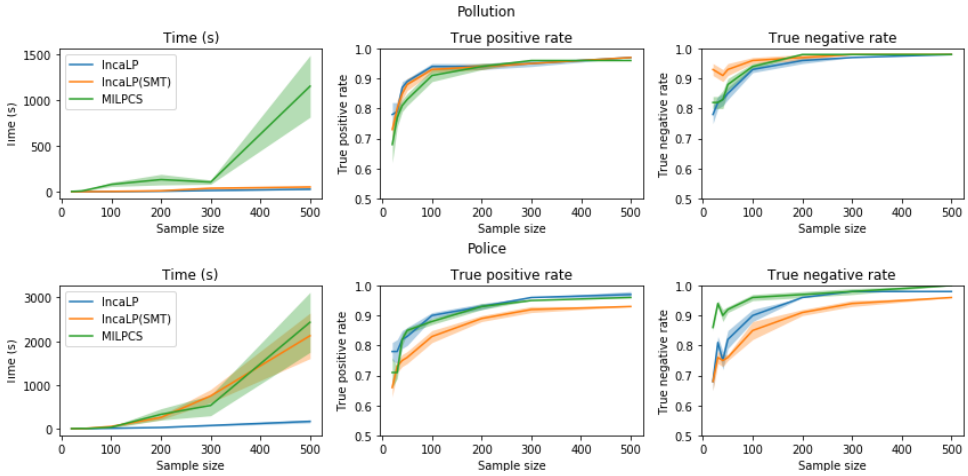

Fig. 4. INCALP is also able to outperform competing approaches on textbook problems, obtaining state-of-the-art TPR and TNR in significantly less time than MILPCS for pollution and both MILPCS and INCALP(SMT) for the sparser police problem. For police, 500 samples, INCALP(SMT) timed out on 1 run, MILPCS on 3.

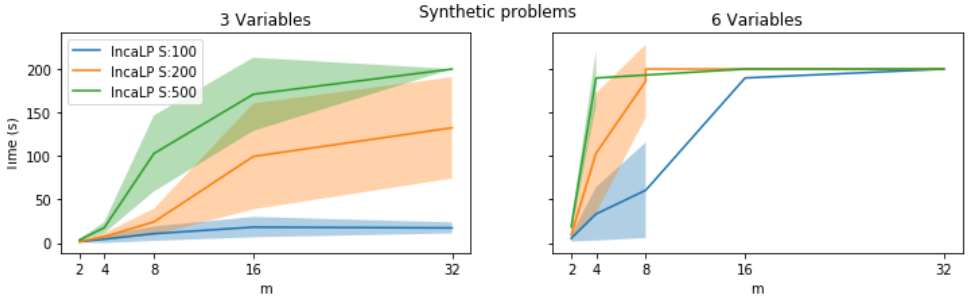

Fig. 5. Results for the synthetic instances: INCALP scales with the number of constraints m, variables n, and examples s. The timeout is $200s$.

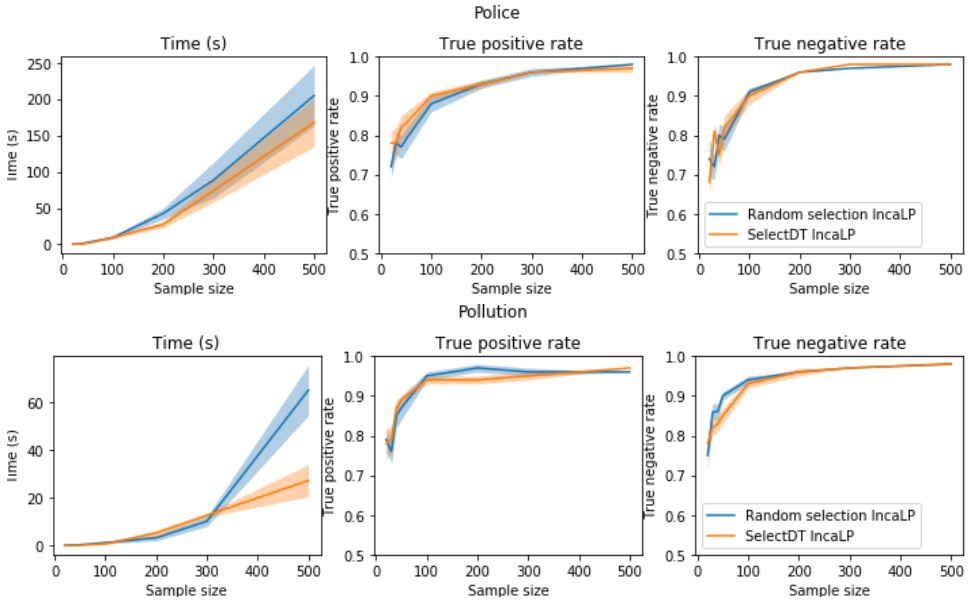

Fig. 6. Results for the SELECTDT heuristic: it achieves faster learning while obtaining high quality constraints for both police (top) and pollution (bottom).