To appear, 2017 Internationl Joint Conference on Artificial Intelligence (IJCAI)
http://www.cds.caltech.edu/~murray/preprints/dagm17-ijcai.pdf

Learning-Based Abstractions for Nonlinear Constraint Solving

Sumanth Dathathri [*1], Nikos Arechiga[2], Sicun Gao[3], and Richard M. Murray[1]

[1]Computing and Mathematical Sciences, California Institute of Technology
[2]Toyota InfoTechnology Center USA
[3]Computer Science and Engineering, University of California, San Diego

Abstract

We propose a new abstraction refinement procedure based on machine learning to improve the performance of nonlinear constraint solving algorithms on large-scale problems. The proposed approach decomposes the original set of constraints into smaller subsets, and uses learning algorithms to propose sequences of abstractions that take the form of conjunctions of classifiers. The core procedure is a refinement loop that keeps improving the learned results based on counterexamples that are obtained from partial constraints that are easy to solve. Experiments show that the proposed techniques significantly improve the performance of state-of-the-art constraint solvers on many challenging benchmarks. The mechanism is capable of producing intermediate symbolic abstractions that are also important for many applications and for understanding the internal structures of hard constraint solving problems.

1 Introduction

Constraint satisfaction problems play a key role in diverse applications, ranging from planning and scheduling for AI systems to program analysis. However, the constraints that arise for real-world systems are often nonconvex, nonlinear, and lack structure that can be exploited, making these problems prohibitively computationally expensive. For instance, if the constraints contain only polynomial arithmetic, then the constraint-solving problem is decidable [Tarski, 1951], but the algorithms are doubly exponential [Brown and Davenport, 2007]. If the constraints contain transcendental functions, such as trigonometrics or exponentials, then the constraint solving problem is undecidable. A relaxed version of the problem that admits approximate solutions, however, is NP-complete [Gao, Kong, and Clarke, 2013].

Several approaches have been proposed to improve performance for discrete constraint satisfiability problems using decompositions, including [Darwiche, 2001; Bacchus, Dalmao, and Pitassi, 2009]. Recent work in this direction is [Friesen and Domingos, 2015] which proposes problem-decomposition approach where the original nonconvex optimization problem is decomposed into subproblems that are approximately independent. In contrast, our approach first decomposes the problem and then uses a learning procedure to find simpler abstractions of the problem that are more tractable to solve. The main problem with finding these *interpolants*—which serve as simplifications through syntactic manipulations [Albargouthi and McMillan, 2013; Drews and Albarghouthi, 2016]—is that they are typically slower and difficult to scale.

However, a concern with using learning-based approaches for constraint solving problems is finding provable relaxations or restrictions of the original problem. Our approach decomposes a set of nonlinear arithmetic constraints into subsets and learns simpler relaxations and restrictions for them. To learn these simplifications, we begin by computing a large number of satisfying and falsifying instances for each subset of constraints using a sampling procedure. For a given set of samples, we use a *semi-soft* support vector machine (SVM) to learn asymmetric classifiers as candidate *antecedents* and *consequents*. The semi-soft SVM is embedded into a refinement loop where a reduced constraint problem is solved to ensure that the candidate antecedent and consequent are indeed a provable antecedent and a provable consequent. The simpler learned antecedents can be used to find a satisfying instance, and the learned consequents can be used to demonstrate that no satisfying instance exists. Other potential applications of the abstraction learning framework proposed here could be in finding Craig interpolants and reachability analysis for dynamical systems.

Other attempts at using learning to find interpolants in domains such as program analysis and safety-analysis for hybrid systems [Sharma, Nori, and Aiken, 2012; Aréchiga et al., 2015] do not address the problem of scalability, which is the main focus of our work.

We show that our techniques significantly improve the performance of constraint solving on various challenging benchmarks, including model-predictive control of Dubins car, finding valid encoder expressions for FPGA design, and geometric reasoning in high-dimensional spaces.

The paper is structured as follows. First, we provide an overview of the problem statement and terminology used in the paper. Next, we describe our asymmetric non-strict classi-

[*]sdathath@caltech.edu

fier, which is used to learn abstractions of sets of constraints. Subsequently, we describe the heuristics that we use to decompose the constraints into subsets for abstraction. Lastly, we describe our results from experiments and conclude.

2 Background

Constraint solving over real numbers. Suppose a set of constraints $\mathbb{A}(x) = \{A_1(x) \ldots, A_m(x)\}$ is given, where x represents a vector of variables. The real-valued constraint solving problem, also called *satisfiability modulo theory of the reals*, is to compute a real-valued vector $\hat{x}$ that simultaneously satisfies all of the constraints in $\mathbb{A}$ or to prove that no such vector exists. If a satisfying instance $\hat{x}$ is found, then we say that the constraint set is *satisfiable*. If no such satisfying instance exists, then we say that the constraint set is *unsatisfiable*.

Antecedents and consequents. Let $A(x)$ be a logical formula with a vector of free variables x. A logical formula $\alpha(x)$ is an *antecedent* of $A(x)$ if $\forall x \,.\, \alpha(x) \implies A(x)$. This means that the values of x that satisfy α are a subset of the values of x that satisfy A, so we will also call $\alpha(x)$ an *underapproximator* of $A(x)$. Conversely, a formula $\gamma(x)$ is a *consequent* of $A(x)$ if $\forall x \,.\, A(x) \implies \gamma(x)$. In this case, all values of x that satisfy $A(x)$ also satisfy $\gamma(x)$, so we refer to $\gamma(x)$ as an *overapproximator* of $A(x)$. If $\{A_1, \ldots, A_k\}$ is a set of constraints, we say that $\alpha(x)$ is an antecedent for the set of constraints if it is an antecedent for the conjunction, i.e.

$$\forall x \,.\, \alpha(x) \implies A_1 \wedge \cdots \wedge A_k. \tag{1}$$

The consequent of a set of constraints is similarly defined to be a logical consequence of the conjunction of the constraints.

3 Faster solving with abstractions

Our approach is to decompose the original set of constraints $A = \{A_1, \ldots, A_m\}$ into subsets $\mathbf{A}_1, \ldots, \mathbf{A}_k$, and then learn antecedents and consequents for each subset. We then use these antecedents and consequents to solve the original constraint satisfaction problem. The details of our learning procedure will be described in Section 4, but for now we simply note that it relies on obtaining satisfying and falsifying instances and learning classifiers from them that are antecedents or consequents.

Let α_j be the learned antecedent for the subset $\mathbf{A}_j$, and let γ_j be its consequent. Then, we solve the sets of constraints $\alpha = \{\alpha_1, \ldots, \alpha_j\}$ and $\gamma = \{\gamma_1, \ldots, \gamma_j\}$. If the antecedents α are satisfiable by some instance $\hat{x}$, then $\hat{x}$ satisfies the original constraint set, by definition of antecedence. Conversely, if the consequents γ are unsatisfiable, then the original set A is also unsatisfiable. For if A were satisfiable, then each $\mathbf{A}_j$ would be satisfiable, and so would each γ_j, by the definition of consequence.

Alternatively, if α is unsatisfiable and γ is satisfiable, then we cannot conclude anything about A. We must refine our antecedents and consequents and try again.

Our technique provides performance improvements under the following premises:

1. The subsets of constraints $\mathbf{A}_j$ are simple enough that they can be solved quickly and repeatedly to obtain satisfying and falsifying instances.
2. The learned antecedents and consequents are simple enough that checking the properties of antecedence and consequence is fast.
3. The learned antecedents and consequents are accurate enough to establish the existence or non-existence of a solution.

The above requirements are inherently conflicting. Greater simplification is obtained by considering larger subsets $\mathbf{A}_j$, but these larger subsets are harder to solve quickly to obtain samples. Similarly, simpler antecedents and consequents will be easier to check, but will provide poorer approximations to solve the original problem.

To illustrate the proposed approach, consider the problem of solving the constraint sets A and B shown in Figure 1. The region arising from constraint set A is underapproximated by the region A'. The constraint set A' is now solved in conjunction with B to find a solution satisfying A and B. The learned constraint set A' has a simpler form, enabling faster computation.

4 Learning abstractions

This section describes an approach for conservative learning of constraints with SVMs [Cortes and Vapnik, 1995]. First, we use a sampling procedure to generate satisfying and falsifying instances for the set of constraints. The proposed sampling procedure (Algorithm 1, described in Section 4.2) uses an existing constraint solver, and is able to generate samples while keeping the constraint queries small.

Next, we use a semi-soft SVM with boosting to generate a constraint that serves as either a candidate antecedent or consequent. This classifier is refined using an abstraction refinement loop until the SMT solver is able to verify that the candidate constraint is a true antecedent or consequent. This constraint is a conjunction of classifiers, and the procedure is described in Algorithm 2.

First we present the formulation of the semi-soft SVM followed by the approach for boosting, where the approximation is learned as a conjunction of classifiers.

4.1 Semi-soft SVM

As a supervised learning approach, SVMs have been known to perform effectively in learning classifiers. Two types of SVM are commonly employed: hard margin and soft margin. Hard-margin SVM approaches do not allow for any misclassification, but may be infeasible if the data points are not linearly separable. On the other hand, soft-margin SVM approaches allow for misclassification of some points, but they do not serve our purpose. Since in general an antecedent will need to exclude all falsifying instances and a consequent will need to include satisfying instances, soft-margin SVMs are not suitable for our purposes.

Instead, we use a semi-soft margin SVM, where we allow only samples of one type to be misclassified. When

Algorithm 1: Sampling Procedure

Input : Set of constraints $A(x), k, l, m, R_1, R_2, \ldots, R_n$
Output: Sets of points X^+ satisfying $A(x)$

$X^+ = \emptyset$, $\bar{X}^+ = \emptyset$;
for *j=1:m* **do**
  Sample $p \vdash A(x)$ s.t. $\forall s \in \bar{X}^+ . d(p, s) > r_{i+1}$;
  $\bar{X}^+ = \bar{X}^+ \cup p$;
end
for *i=1:l-1* **do**
  $\underline{X}^+ = \emptyset$;
  for $q \in \bar{X}^+$ **do**
    $X^+ = \emptyset$;
    for *j=1:m* **do**
      Sample $p \vdash A(x)$ s.t. $d(p, q) < r_i$ and $d(p, s) > r_{i+1} . \forall s \in X^+$;
      $X^+ = X^+ \cup p$;
    end
    $\underline{X}^+ = \underline{X}^+ \cup X^+$;
  end
  $\bar{X}^+ = \underline{X}^+$;
end
$X^+ = \underline{X}^+$;
return X^+ ;

Algorithm 2: Algorithm to learn abstractions for sets of nonlinear arithmetic constraints

Input : Set of constraints $A(x)$, parameters m, k
Output: $g(x)$ such that $A(x) \implies g(x) > 0$

Sample set of points X^+ with $A(x)$ as input to Algorithm 1 ;
Sample set of points X^- with $\neg A(x)$ as input to Algorithm 1 ;
Initialize $g_0(x) =$ **TRUE** , iter $= 0$;
Set $X = X^+ \cup X^-$;
Map the X to a higher-dimension feature space ;
Cluster points in X^- into $\mathcal{N}$ clusters ;
Learn a classifier separating each cluster in X^- from X^+ using the semi-soft SVM;
Set $g(x) = g_0(x) \wedge$ conjunction of the $\mathcal{N}$ classifiers ;
if $A(x) \implies g(x) > 0 \ \wedge$ iter $\leq k$ *;*
then
  return $g(x)$;
else if iter $\leq k$ **then**
  iter=iter+1;
else
  $g_0 = g(x)$;
  iter=0;
Sample a set of points X^* with $\neg(A(x) \implies g(x) > 0)$ as input to Algorithm 1;
$X^+ = X^+ \cup X^*$;
go to 4 ;

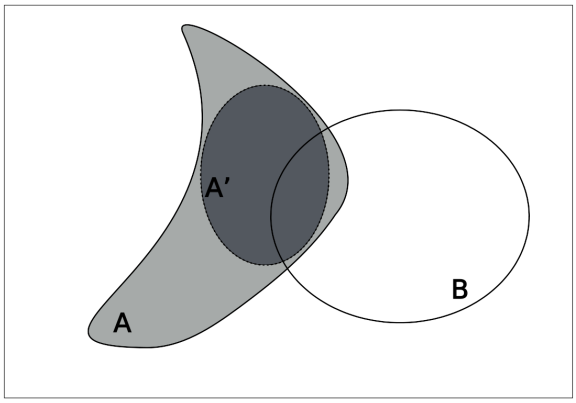


Figure 1: Illustration of the approach

searching for an antecedent (underapproximation of satisfying instances), we allow some positive examples to be misclassified. Similarly when searching for a consequent, we allow some negative examples to be misclassified, resulting in an overapproximation of the satisfying instances. Asymmetric SVMs have been of interest to minimize cases of false-negatives (or false-positives) in certain applications. Some efforts directed at asymmetric learning [Wu et al., 2008; Wu et al., 2013] aim to minimize the false-negatives but for learning consequents we seek to eliminate false-negatives.

Use of a semi-soft SVM yields a quadratic program. To abstract the constraints $\mathbf{A}_j$, we label its satisfying instances as $+1$ and its falsifying instances as -1. To compute a consequent, we set up the optimization problem to necessarily yield positive labels for the positive instances, while simply making a best effort attempt to provide negative labels for the negative instances.

Let X^+ be the set of satisfying instances of the subset of constraints $\mathbf{A}_j(x)$ and X^- the set of falsifying instances of $\mathbf{A}_j(x)$. X^+ and X^- are obtained by the sampling procedure described in Section 4.2. We will search for a consequent by the following optimization problem, which learns the classifier $g(x) = sgn(w_0 + w^T x)$:

$$\begin{aligned} \min_{w, w_0, e_b} \frac{1}{2} \left(w_0^2 + w^T w + \lambda \sum_{n=1}^{N^-} e_b \right) \\ s.t \qquad y^- \left(w_o + w^T x_b^- \right) \geq 1 - e_b, \\ y^+ \left(w_o + w^T x^+ \right) \geq 1, \\ e_b \geq 0, \\ \forall x^+ \in X^+, \\ \{x_1^-, x_2^-, \ldots, x_{N^-}^-\} = X^-, \\ b = 1, 2, \ldots, N^-, \end{aligned} \tag{2}$$

where $N^- = |X^-|$ and $N = |X^+ \cup X^-|$. Here e_b are the slack variables allowing for misclassification of the points in X^-. Notice that this deviates from the standard soft-margin SVM in the sense that only one type of data-points are assigned slack-variables. The labels $y^+ = +1$ and $y^- = -1$

correspond to points in X^+ and X^- respectively. Note that this optimization problem provides hard constraints for classifying positive points, but soft constraints for classifying negative points.

Proposition 1 *For any given sets of points X^+ and X^-, the semi-soft SVM as formulated in* (2) *finds a hyperplane $w_0 + w^T x = 0$ such that $\forall x^+ \in X^+$, $w_0 + w^T x^+ > 0$.*

Proof For $x^+ \in X^+$, by the constraint, $y^+ (w_o + w^+ x^+) \geq 1$, we have $w_0 + w^T x^+ \geq 1 > 0$.

The minimization problem in equation (2) is a quadratic program (QP). A solution to the constraints of the QP is $w_0 = 1, w = 0$ and $e_b = 2$, implying that the QP has a non-empty feasible set. Therefore, equation (2) always returns a hyperplane that has the above property of correctly classifying the points in X^+.

The case for computing an antecedent can be derived in a straightforward manner, by requiring that the classifier correctly provide negative labels for the negative instances, while simply making a best effort attempt to provide positive labels for the positive instances.

4.2 Sampling for learning

A key aspect of this problem is sampling points that reasonably cover both the regions satisfying the constraints and the regions that do not satisfy the constraints to learn meaningful classifiers. One way to implement such a sampling strategy follows.

We will use a distance metric $d(\cdot, \cdot)$ to enforce spacing between samples. For the experiments in this paper, the distance function is the ℓ^2-norm. We begin by choosing a large radius R_0, and sample a set of points $C = \{c_1, \ldots, c_k\}$ that are separated by a distance of at least R_0. These samples will serve as centers for circles that we will sample from at the next iteration.

Now, consider a smaller radius $R_1 (< R_0)$. Define circles with radius R_0 around each of the sampled points $\Omega_i = \{x | d(x, c_i) \leq R_0\}$. In each Ω_i, sample points that are at least R_1 apart. In this manner we can iterate through a sequence of decreasing radii $R_2, R_3 \ldots R_l$ by defining circles around the sampled points with radius R_i and sampling points from these circles that are at least R_{i+1} apart and so on.

The radii R_k are chosen so that the points sampled in a particular layer are distributed around the feasible region (the intersection of the feasible region for the problem and the circles defined by us) for that layer. Given a fixed number of samples m to be sampled in a layer, the optimal radius for that layer can be determined by a bisection search to find approximately the largest radius that gives m samples per layer. This implies that there exists no other point in the feasible region that can be sampled that is at a distance greater than r_k from the other sampled m points. Further increasing the radius would imply that we cannot find m such points and if we can find more than m points, we increase the radius until only m are found. This approach provides good coverage in our experiments. However, even if the initial samples are poorly distributed, the CEGAR loop described in Section 4.3 iteratively searches for antecedents and consequents, and only terminates once provable abstractions are found. As a result, the quality of the initial samples only has an effect on performance, but does not compromise soundness. The approach is more formally described in Algorithm 1. The term 'Sample' in Algorithm 1 refers to a query to the constraint solver to find a feasible point satisfying the constraints. Though Algorithm 1 provides no formal guarantees about m samples being found during each iteration, the learning can be performed with fewer than m samples. When the constraints have no solution and no sample can be found, the classifier learned is `false`.

Besides good coverage, an advantage of this approach is that it prevents the explosive growth of the formula used to sample using an SMT solver. Once the first set of samples are generated, the procedure is parallelized to generate samples in the balls around each of these points since the sampling in the individual balls is independent of that in the others.

4.3 Counterexample Guided Abstraction Refinement(CEGAR)

Counterexample-guided abstraction refinement (CEGAR) was first proposed in [Clarke et al., 2000] for refining abstractions of control structures in programs. We develop our algorithm in the paradigm of CEGAR.

Once the classifier $g(x)$ has been learned by the semi-soft SVM, it still remains to be validated that the classifier learned is an overapproximation (underapproximation) of the constraint-set we seek to abstract. Let A_i be the conjunction of the constraints in $\mathbf{A}_i$. To verify that the classifier learned is indeed a consequent (antecedent) of the constraint-set $\mathbf{A}_i(x)$, we need to check the consequence condition $A_i(x) \implies g(x) > 0$ (for showing antecedence $g(x) > 0 \implies A_i(x)$). If the negation of this formula is not satisfiable, i.e there is no assignment to x over which the formula interprets to $false$, we have learned a consequent $\gamma_i(x) \equiv (g(x) > 0)$ (respectively, antecedent $\alpha_i(x) \equiv (g(x) > 0)$). Otherwise, we sample points that violate the formula as proposed in Algorithm 1, add them to the training set, and learn a new classifier. This procedure is iteratively repeated until we have learned a consequent (antecedent). Note that although Algorithm 2 may fail to terminate, it will terminate only if it has found a consequent for $\mathbf{A}_i(x)$.

4.4 Boosting

In some problems, the CEGAR procedure might fail to converge after a large number of iterations or learn poor approximations of the original problem. This can occur if the constraints to be approximated represent regions in space that cannot be approximated by a single classifier. This section describes how to sequentially learn classifiers that in conjunction form a better relaxation.

As a first step, the falsifying points are partitioned into clusters using a clustering algorithm (K-means for all examples presented in this paper). A classifier is learned from each cluster separating it from all the feasible points. These classifiers in conjunction form a first approximation of the constraints. Next, a few iterations of CEGAR are run where this approximation is refined. Subsequently, collect the misclassified falsifying points and learn a new classifier using the misclassified points and the satisfying points in a similar manner.

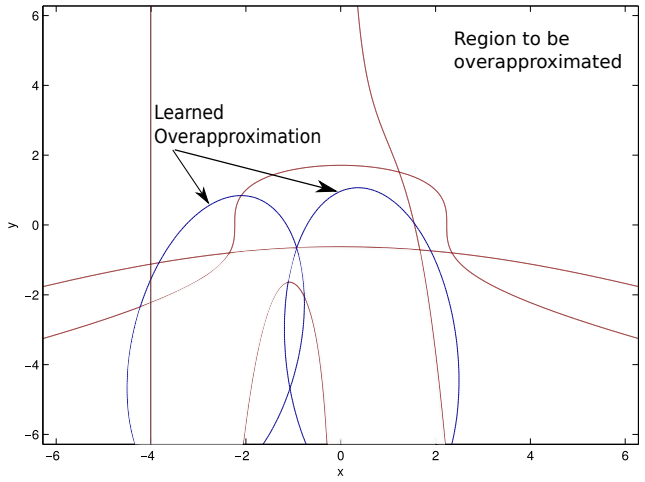


Figure 2: Quadratic Overapproximation of the infeasible region.

Note that none of the feasible points are misclassified by the construction of the semi-soft SVM.

During the CEGAR step that learns the new classifier, the condition $A_i(x) \implies \gamma_i(x)$ is checked where now $\gamma_i(x)$ is the conjunction of the most recent classifier learned and the previous classifiers. The classifier learned at the end of these CEGAR iterations is the conjunction of the two classifiers (which in turn are a conjunction of classifiers).

This differs from conventional boosting [Ting and Zhu, 2009], where the classifier learned is a weighted sum of an ensemble of classifiers, and cascade SVMs [Graf et al., 2004], where the classifiers are learned in parallel and then combined into a single one.

4.5 Example

To demonstrate the approach described above consider the following constraint

$$\begin{aligned} A = & x^2 + y^3 \leq 4 \wedge x \geq -4 \\ & \wedge\, x^2 + y^3 + 30y \leq -20 \\ & \wedge\, x^5 + 3xy - 2x \leq 5. \end{aligned} \tag{3}$$

Suppose we want to find a point that lies outside A but satisfies a constraint C i.e we are trying to solve $\neg A \wedge C$.

We first seek to learn an underapproximation of the feasible region ($\neg A$) or an overapproximation of the infeasible region (A) with quadratic classifiers, since a collection of quadratic constraints is easier to solve than a collection of higher order polynomial constraints. Recall that by negating the overapproximation of A, we can obtain an underapproximation of $\neg A$.

For constraint A we obtain satisfying and falsifying instances using Algorithm 1. Next, the sampled instances are transformed to a higher dimensional space by a feature map [Boser, Guyon, and Vapnik, 1992], which lets us learn quadratic classifiers in the higher dimensional space by the semi-soft SVM procedure described in Section 4.1. An antecedent for $\neg A$ is learned as described in Algorithm 2. Recall that the antecedence is verified using an SMT solver in Algorithm 2.

Here, the number of clusters is set at 2 and we repeat CEGAR until it converges. We learn an abstraction that is a conjunction of two quadratic classifiers. The abstraction is $A' = 0.109xy - 0.788x + 0.915x^2 + 0.935y + 0.135y^2 \geq 1.03 \wedge 5.077 - 0.338xy + 6.001x + 1.36x^2 + 0.887y + 0.257y^2 \geq 0$. A' underapproximates $\neg A$. To solve $\neg A \wedge C$, we solve $A' \wedge C$. Figure 2 shows the original set of constraints (red). The infeasible region (A) is shaded. The classifiers learned are shown in the graph (blue). The feasible area is underapproximated by the intersection of the regions outside each of the classifiers. The solution space for the problem is restricted to $x \in [-2\pi, 2\pi]$ and $y \in [-2\pi, 2\pi]$.

5 Decomposition Methods

This section describes the heuristics that we use to select which constraints should be considered together to learn an abstraction. The first level of decomposition is based on the Hamming distance.

Hamming decomposition This heuristic groups together constraints that have many common variables—i.e., those constraints whose sets of variables, treated as vectors, have a small Hamming distance. Let $FV(\cdot)$ be the function that takes a clause and returns the set of free variables of that clause. Then, the Hamming distance between constraints c_m and c_n can be computed as

$$H(c_m, c_n) = |\,(FV\,(c_m) \setminus FV\,(c_n)) \cup (FV\,(c_n) \setminus FV\,(c_m))\,|.$$

We assume that a maximum distance bound θ is given, such that any two constraints that have a Hamming distance less than or equal to θ will be grouped together for abstraction. Our implementation loops over the constraints; starting with the first clause c_0, it chooses all constraints c_k such that $H(c_0, c_k) \leq \theta$. After the first pass, it chooses the next constraint that was not grouped and loops over the remaining constraints.

If we group the constraints into a few classes with many constraints each, then sampling each class to generate abstractions will be computationally difficult. In the limit case, if all constraints are grouped into a single class, then choosing a single satisfying instance corresponds to solving the original satisfiability problem.

Bounded-size decomposition In some cases, it may be the case that the earlier decomposition did not produce classes that are sufficiently small to ensure that sampling can be performed quickly. In this case, we set a bound n, and divide the large classes into smaller subsets with $\leq n$ constraints each.

6 Experiments

This section details the results of experiments on various benchmark sets on a 32-core AMD Opteron machine at 2.3 GHz, with 96 GB of RAM [1]. A timeout of 200 seconds is set

[1]Benchmark problem instances can be found at `https://github.com/dathath/IJCAI_2017_SD`

for abstracting each subset, if the abstraction procedure fails to finish in this period the constraints are used as is.

Model-predictive control for Dubins car. The model predictive control problem is to find a sequence of control inputs so that a car can reach a specific position. We use a Dubins car dynamics model with additional nonlinear terms:

$$\begin{aligned} a_t + c_1(a_t + x_t/y_t) &\geq a_{t+1}, \\ y_t + c_2\left(\cos\left(a_t\right) - y_t^2\right) &\geq y_{t+1}, \\ x_t + c_3\left(\sin\left(a_t\right) - x_t^2\right) &\geq x_{t+1}. \end{aligned} \tag{4}$$

The constraints encode the problem of starting from an initial point and the objective is to determine if a target polytope can be reached i.e $(x_{final}, y_{final}) \in ([a, b] \times [c, d])$.

The benchmark comes from practical problems in mobile robotics, and its scale can be varied by changing the number of time steps. All the problem instances are satisfiable. Figure 3a shows a plot of the run-times for dReal, our abstraction approach, and the time required to check abstractions alone. We cannot compare with un-abstracted performance of z3, because z3 does not support solving over sines and cosines. For this same reason, the sampling and abstraction checking are carried out with dReal, though we solve the simplified problem with z3, because in this example the resulting abstractions are conjunctions of linear constraints, and z3 typically performs better than dReal at linear constraints. The classifier learned for this example is a conjunction of several linear classifiers, one linear classifier learned during each step of CEGAR. The Hamming distance threshold here was set to 4, which results in subsets with 3 constraints each. Since sampling these constraints is fast, further decomposition is not required. The abstracted constraints are always solved faster that solving the constraints directly, which could be useful in an application scenario in which abstractions can be cached and re-used. For large problems, the abstraction-based approach (including the time for abstraction) performs considerably better than solving directly with dReal.

Energy-efficient FPGA design. The constraint sets here correspond to the search for valid probabilistic encoder expressions in a value-deviation-bounded serial encoding technique [Stanley-Marbell, Francese, and Rinard, 2016]. The search problem is to find assignments of probabilities for a set of Bernoulli random variables such that they satisfy the constraints for valid encoder family formulations. The hamming distance threshold is set to 3 and n is set at 5. The number of clusters was set to 2. The classifiers learned were of the form:

$$g(x) = \left(\sum_i \left(a_i x_i^2 + b_i x_i\right) + c_0 > 0\right) \wedge \left(\sum_i \left(d_i x_i^2 + e_i x_i\right) + f_0 > 0\right).$$

The abstraction procedure converged for all subsets of constraints in 20 of the benchmarks (out of a total of 29). Table 1 lists the average runtimes for the benchmarks on the different solvers. We see that even though the abstraction procedure does not terminate for all constraint sets, in the case when the procedure does terminate, the constraints are solved significantly faster. dReal times out directly on one of the benchmarks but solves it after abstraction – this benchmark is not included in computing the average times in Table 1. For all three solvers, the average time and the number of constraints are mean values reported over the 20 problem instances. The constraints in the problem instances are polynomials and the mean degree reported is over the set of constraints occurring in the 20 problem instances.

Solver	No. Of Benchmarks Solved	Average Time for Benchmarks Fully Abstracted (ms)	Avg. Number of Constraints Input Per Instance	Avg. Degree of Constraints Input to Solver
z3	29	1793.4	∼32	∼5.93
dReal	28	1788.8	∼32	∼ 5.93
Post-Abstraction (with dReal)	20	133.3	∼11.5	∼ 2.06

Table 1: Comparison on benchmarks for which the abstraction procedure completes

Sphere packing in high-dimensions In this set of benchmarks, we solve for a point in the intersection of the exterior of a set of randomized high-dimensional spheres of the form

$$\sum_{j \in \mathrm{I}} (x_j - a_j)^2 \leq 30 \tag{5}$$

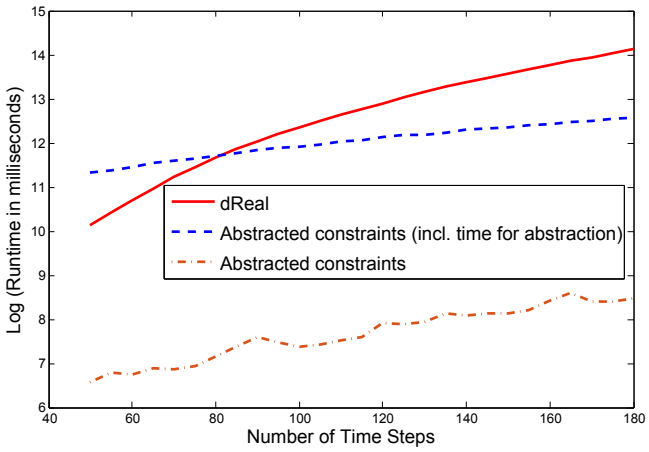
and cubes bounding the search region ($|x_k| \leq 30$). For each sphere, the parameters $a_j : \forall j \in I$ are randomly chosen in the range $[-50, 50]$ and the set of variables $x_j : j \in I$ in each constraint are chosen at random uniformly from $\{1, 2, \ldots, 15\}$. Figure 3b shows a plot of the different runtimes for varying number of constraints. The time for solving the abstracted constraints is significantly smaller than the time for solving directly with dReal and z3 and the time for abstraction scales almost linearly since the time for abstracting each subset is independent of the total number of constraints. The parameters for the learning procedure are the same as those from the FPGA design problem.

Hong family. These sets of benchmarks correspond to the Hong family of benchmarks [Jovanović and de Moura, 2013]. We restrict the solution space to the positive quadrant. The problem instances are always `unsat` by construction.

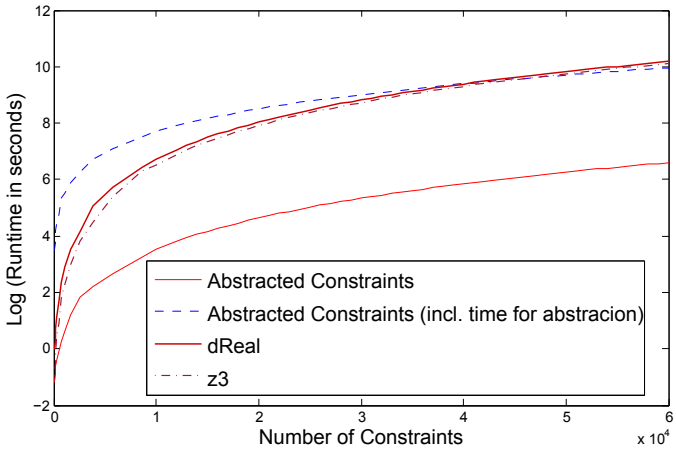
A parameterized generalization of the problem is:

$$\prod_{i=1}^{n} x_i > 1, \sum_{i=1}^{n} x_i^2 < 1. \tag{6}$$

To facilitate learning abstractions, we rewrite the problem by replacing the quadratic constraint with constraints of the form $x_i^2 + x_{i\oplus 1}^2 \leq z_j$ and $\sum_j z_j \leq 2$. This makes the sampling process easier because each of these constraints have to be sampled only in three dimensions. The Hamming threshold



(a) Car Reachability



(b) Randomized spheres

Figure 3: Performance on various benchmarks

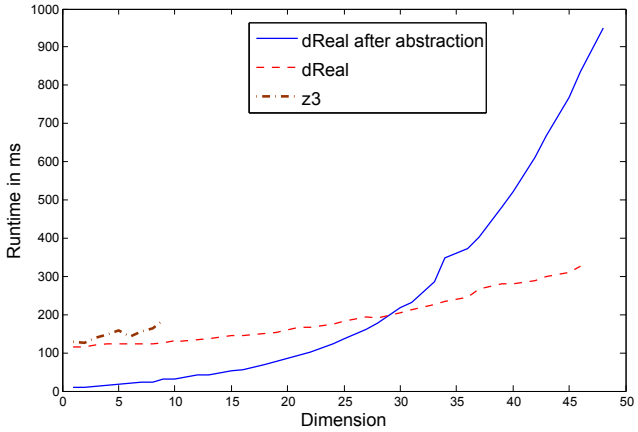


Figure 4: Performance – Hong Family

is set to 3 and bounded-size decomposition is not used here. The classifiers learned are a conjunction of linear classifiers as in the car benchmark. Figure 4 shows the runtimes. z3 does not scale beyond low dimensions, while dReal almost scales linearly. The abstraction-based scheme does reasonably well at low dimensions but at higher dimensions the relaxations computed slow down dReal.

7 Complexity and Discussion

Since the benchmarks contain trigonometric and highly non-linear functions, even the relaxed versions of the problem that admit solutions are NP-complete [Gao, Kong, and Clarke, 2013]. Existing solvers run into scalability issues when dealing with large constraint solving instances as in [Gorcitz et al., 2015].

In our approach, the number of samples grows exponentially with the number of variables in the constraints. But by restricting to small subsets of the original set of constraints, this exponential growth can be capped. Lloyd's algorithm [Lloyd, 2006] for K-means clustering is a linear-time algorithm and the semi-soft SVM results in a QP that is solvable in polynomial time. Verifying that the classifier learned from a small subset of constraints is a consequent (or an antecedent) is a reduced problem whose complexity is much smaller than solving the entire large set of constraints, since the complexity of constraint solving scales exponentially. We use a sequence of polynomial time algorithms to learn an approximate simpler instance of the constraint satisfaction problem.

From the experiments, we observe that though the abstraction procedure is not complete it can improve the handling of large sets of complex constraints abstracting certain subsets of the constraints. The experiments demonstrate the potential of the learning based abstraction approach in improving the scalability of constraint solvers.

8 Future work

A future direction of work is to develop more sophisticated heuristics for deciding which constraints are to be abstracted together. Certain decompositions could result in regions that are easier to abstract with a low-complexity, high-fidelity abstraction. Additionally, our current implementation samples points that satisfy and falsify a logical formula in the learning process, without considering the other logical constraints. In future work, we would like to address the scenario of proving a sequent, and leveraging knowledge about the consequent to guide the search of abstractions for portions of the antecedent.

Another immediate direction is to restrict the variables in the classifiers for a subset to be abstracted to those that the subset shares with the remaining constraints. A subset imposes restrictions on the other constraints only through the shared variables and learning constraints on the shared variables should suffice.

Acknowledgments This work was partially supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA and in part by Toyota InfoTechnology Center and NSF CPS1446725. The authors would also like to thank Joel W. Burdick for helpful input.

References

[Albargouthi and McMillan, 2013] Albargouthi, A., and McMillan, K. L. 2013. Beautiful interpolants. In *Computer Aided Verification*.

[Aréchiga et al., 2015] Aréchiga, N.; Kapinski, J.; Deshmukh, J. V.; Platzer, A.; and Krogh, B. 2015. Forward invariant cuts to simplify proofs of safety. In *Embedded Software*.

[Bacchus, Dalmao, and Pitassi, 2009] Bacchus, F.; Dalmao, S.; and Pitassi, T. 2009. Solving #sat and bayesian inference with backtracking search. *J. Artif. Int. Res.* 34(1):391–442.

[Boser, Guyon, and Vapnik, 1992] Boser, B. E.; Guyon, I. M.; and Vapnik, V. N. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, 144–152. New York, NY, USA: ACM.

[Brown and Davenport, 2007] Brown, C. W., and Davenport, J. H. 2007. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*.

[Clarke et al., 2000] Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In *Computer Aided Verification: 12th International Conference*.

[Cortes and Vapnik, 1995] Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20(3):273–297.

[Darwiche, 2001] Darwiche, A. 2001. Recursive conditioning. *Artif. Intell.* 126(1-2):5–41.

[Drews and Albarghouthi, 2016] Drews, S., and Albarghouthi, A. 2016. Effectively propositional interpolants. In *Computer Aided Verification*.

[Friesen and Domingos, 2015] Friesen, A. L., and Domingos, P. 2015. Recursive decomposition for nonconvex optimization. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 253–259. AAAI Press.

[Gao, Kong, and Clarke, 2013] Gao, S.; Kong, S.; and Clarke, E. M. 2013. *dReal: An SMT Solver for Nonlinear Theories over the Reals*. Berlin, Heidelberg: Springer Berlin Heidelberg. 208–214.

[Gorcitz et al., 2015] Gorcitz, R.; Kofman, E.; Carle, T.; Potop-Butucaru, D.; and de Simone, R. 2015. *On the Scalability of Constraint Solving for Static/Off-Line Real-Time Scheduling*. Cham: Springer International Publishing. 108–123.

[Graf et al., 2004] Graf, H. P.; Cosatto, E.; Bottou, L.; Dourdanovic, I.; and Vapnik, V. 2004. Parallel support vector machines: The cascade svm. In Saul, L. K.; Weiss, Y.; and Bottou, L., eds., *Advances in Neural Information Processing Systems 17*. Cambridge, MA: MIT Press. 521–528.

[Jovanović and de Moura, 2013] Jovanović, D., and de Moura, L. 2013. Solving non-linear arithmetic. *ACM Commun. Comput. Algebra* 46(3/4):104–105.

[Lloyd, 2006] Lloyd, S. 2006. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.* 28(2):129–137.

[Sharma, Nori, and Aiken, 2012] Sharma, R.; Nori, A. V.; and Aiken, A. 2012. Interpolants as classifiers. In *Computer Aided Verification*.

[Stanley-Marbell, Francese, and Rinard, 2016] Stanley-Marbell, P.; Francese, P. A.; and Rinard, M. 2016. Encoder logic for reducing serial i/o power in sensors and sensor hubs. In *28th Annual IEEE Symposium on High-Performance Chips (Hot Chips'16)*.

[Tarski, 1951] Tarski, A. 1951. *A Decision Method for Elementary Algebra and Geometry*. Berkeley: University of California Press, 2nd edition.

[Ting and Zhu, 2009] Ting, K. M., and Zhu, L. 2009. *Boosting Support Vector Machines Successfully*. Berlin, Heidelberg: Springer Berlin Heidelberg. 509–518.

[Wu et al., 2008] Wu, S.-H.; Lin, K.-P.; Chen, C.-M.; and Chen, M.-S. 2008. Asymmetric support vector machines: Low false-positive learning under the user tolerance. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, 749–757. New York, NY, USA: ACM.

[Wu et al., 2013] Wu, S.-H.; Lin, K.-P.; Chien, H.-H.; Chen, C.-M.; and Chen, M.-S. 2013. On generalizable low false-positive learning using asymmetric support vector machines. *IEEE Transactions on Knowledge and Data Engineering* 25(5):1083–1096.