Hybrid Solving for Quantified Boolean Formulas Based on SVM and Reinforcement Learning

Tao Li\textsuperscript{a,b,*}, Nanfeng Xiao\textsuperscript{a}

\textsuperscript{a}School of Computer Science and Engineering, South China University of Technology
Guangzhou 510640, China
\textsuperscript{b}Modern Education and Technology Center, South China Agricultural University
Guangzhou 510640, China

Abstract

In this paper, a new SVM classification algorithm is proposed, this algorithm is applied in Quantified Boolean Formulas (QBF) hybrid solving and a new QBF hybrid solver is designed. This solver apply SVM algorithm to construct inductive models and classify the formulae. At the same time, the reinforcement learning technology is applied to realize the dynamic algorithm selection. The relationship between the solving algorithm and formula is established according to test formula set and Run-time performance. When the first heuristic fails in resolving the formula, the Reinforcement Learning procedure can select another solving algorithm automatically. It can drastically improve the performance of a QBF solver and resolve more numerous formulas than sequential QBF Solver. It proved that non-linear prediction models based on SVM and the reinforcement learning are very useful in formulae classification and online algorithm switch. It is believed that machine learning technology can be helpful to a much larger degree when solving hard problems.

Keywords: Support Vector Machine; Quantified Boolean Formulas; Reinforcement Learning; Dynamic Algorithm Selection

1 Introduction

Boolean Satisfiability (SAT) solvers have become powerful enough to solve many practically relevant problems, and they are currently used in numerous industrial tools for circuit and software verification. Building upon this success, the research community has begun to consider the more general, but also more complicated Quantified Boolean Formula (QBF) domain. This allows researchers to encode problems encountered in Black Box or Partial Circuit Verification, Bounded Model Checking, and AI planning more naturally and compactly than in SAT. However, since QBF problems are generally more difficult (PSPACE-complete vs. NP-Complete), they require dedicated algorithms and increased computation power to solve relevant instances [1].
To make the QBF-based approach effective, solvers ought to be robust, i.e., able to perform well across different problem domains without the need for domain specific tuning. However, as the results of the yearly QBF solvers competitions clearly show, QBF solvers are rather brittle. The cause of this phenomenon can be traced back to the fact that all state-of-the-art QBF solvers implement some kind of heuristic algorithm. Indeed, every such algorithm will occasionally find problem instances that are exceptionally hard to solve, while the very same instances may easily be tackled by resorting to another algorithm, or by using a different heuristic. There is no single solving algorithm can handle all the QBF problems. It is necessary to multipurpose use the merit of different algorithm in order to resolve the different hard problem. It is the essence thinking in [4], the author design the self-adaptive multi-engine solver AQME for quantified Boolean formulas. In this paper, we follow the same ideas as in [4], improve the AQME and apply some other advanced technology for the same purpose. The difference is that we apply the non-linear prediction models Support Vector Machine (SVM) and reinforcement learning technology. It classifies the formulas more accurately and realizes the online dynamic algorithm selection for different QBF solving heuristics.

In this paper, a framework is developed to solving QBF that combines several QBF solvers with machine learning techniques SVM. We use statistical classification to predict optimal heuristics within a portfolio, as well as in a dynamic, online-setting. We use reinforcement learning for dynamic algorithm selection, i.e., reinforcement learning. The hardest challenge for the former will be run-time efficiency, which might rule out kernel-based and other nonparametric, methods.

The paper is structured as follows: Section 2 will start with a description of the QBF problem and how sequential and hybrid QBF solvers work (Section 2.1 and 2.2). In Section 3 we introduce a new classification algorithm of SVM. In Section 4 we introduce the process of QBF hybrid solving based on our new SVM classification, inductive models and reinforcement learning. In Section 5 we give some technical details and experimental results about our implementation, Section 6 will conclude this paper and giving some future research directions.

2 Preliminaries

There are many ways to encode a QBF problem [1], but in our context, they are defined in Conjunctive Normal Form (CNF). A problem in CNF form starts with a variable definition. The variable definition quantifies each variable (either existentially or universally), and assigns each variable to a specific quantification level. Once the variable definition is complete, a set of clauses is given that defines the problem. More formally, a QBF is an expression of the form:

\[ \varphi = Q_1 z_1 Q_2 z_2 \cdots Q_n z_n \Phi \quad (n \geq 0) \]  

Here, every \( Q_i (1 \leq i \leq n) \) is a quantifier, either existential \( \exists \) or universal \( \forall \), \( z_1, \cdots, z_n \) are distinct sets of variables, and \( \Phi \) is a propositional formula. \( Q_1 z_1 Q_2 z_2 \cdots Q_n z_n \Phi \) is defined as the prefix, and \( \Phi \), the propositional formula, would contain a set \( P \) of clauses. While a variable is defined as an element of \( P \), an occurrence of that variable or its negation in a clause is referred to as a literal. In the following, the literal \( \bar{l} \) is defined as the negative occurrence of the variable \( |l| \) in \( P \), and \( l \) is the positive occurrence. In the following, we also use TRUE and FALSE as abbreviations for the empty conjunction and the empty disjunction, respectively. For example, an entire problem definition might be as follows:

\[ \exists x_1 \forall y \exists x_2 \{ (\bar{x}_1 \lor \bar{y} \lor x_2) \land \{ \bar{y} \lor \bar{x}_2 \} \land \{ x_2 \} \land \{ x_1 \lor \bar{y} \} \land \{ y \lor x_2 \} \} \]  

(2)
We say that \( (1) \) is in Conjunctive Normal Form (CNF) when \( \Phi \) is a conjunction of clauses, where each clause is a disjunction of literals as shown in (2). And that \( (1) \) is in Disjunctive Normal Form (DNF) when \( \Phi \) is a disjunction of cubes, where each cube is a conjunction of literals\(^1\). We use constraints when we refer to clauses and cubes indistinctly. We also define:

1. The level of a variable \( z_i \), to be \( 1 + \) the number of alternations \( Q_j z_j Q_{j+1} z_{j+1} \) in the prefix with \( j \geq i \) and \( Q_j \neq Q_{j+1} \).

2. The level of a literal \( l \) is the level of \( |l| \).

3. The level of the formula \( (1) \) is the level of \( z_1 \).

So, for example, in equation (2), \( x_2 \) is existential and is quantified on level 1, \( y \) is universal and is on level 2, \( x_1 \) is existential and is on level 3.

QBF solvers are interested in answering the question of whether or not equation (1) \( \varphi \) expresses a true or false assertion, i.e., whether or not \( \varphi \) is true or false. The reduction of a CNF formula \( \Phi \) by a literal \( l \) is the new CNF \( \Phi|_l \) which is \( \Phi \) with all clauses containing \( l \) removed and \( \neg l \), the negation of \( l \), removed from all remaining clauses. For example, let \( \varphi = \forall x z \exists y (\bar{y}, x, z) \land (\bar{x}, y) \), then \( \varphi|_x = \forall z \exists y (\bar{y}, z) \). The semantics of a QBF can be defined recursively in the following way:

1. If \( \Phi \) is the empty set of clauses then \( \varphi \) is true.
2. If \( \Phi \) contains an empty clause then \( \varphi \) is false.
3. \( \forall v \varphi \) is true if both \( \varphi|_v \) and \( \varphi|_{\neg v} \) are true.
4. \( \exists v \varphi \) is true if at least one of \( \varphi|_v \) and \( \varphi|_{\neg v} \) is true.

In this paper, we only study the formula with conjunctive normal form.

2.1 Sequential QBF Solver

The sequential QBF solvers usually apply one single algorithm to resolve the whole formulae set at runtime. We have no choice when encounter different hard problem, so some problems can not be resolved. But if we apply other solvers, the extreme hard formulae maybe easily resolved.

There are many sequential QBF solvers\(^1\). Most solvers like QMiraXT\(^1\), QuBE\(^5, 6\), yQuaffle\(^7\), sSolve\(^4\), are in principal based on the DPLL algorithm\(^2\). Others, like Quantor\(^8\) or Nenofex\(^9\), try to resolve and expand the formula until no universally quantified variables remain. This allows them then to send their remaining, existentially quantified problem to a SAT solver. This works well on many problems, but it can result in an explosion with respect to the size of the formula. On the other hand, solvers like sKizzo\(^10, 11\) do the opposite of Quantor\(^2\), and use symbolic skolemization to eliminate all the existentially quantified variables in the formula. Some so-called incomplete solvers are also based on stochastic search methods, and they can be very effective in solving some categories of problems, but are not able to prove the value of unsatisfiable formulas. A few alternative algorithms for QBF are emerging, e.g. And-Inverter Graphs. Their usage in QBF satisfiability algorithms have been explored at least in\(^12\).

2.2 Hybrid QBF Solver

In hybrid QBF solvers, a portfolio of sequential solvers is considered, and the best one is selected using algorithm selection techniques. There are many hybrid solvers for SAT, but very little for
QBF. As far as we know, there exist two main implementations of Hybrid solvers for the problem of validity of QBF: AQME [4] and an unnamed solver [2]. They both use machine learning techniques to choose the best sequential solver for each single formula. On the other hand, those two procedures are all dedicated for CNF QBF.

Our paper builds strongly on [4], wherein the self-adaptive multi-engine approach herewith discussed was introduced. In [2] an independent contribution along the same lines of [4] is made. In contrast to [2] and [4], this paper applies a new SVM classification algorithm and reinforcement learning technology. We describe it in Section 3 and Section 4. It is the first time uses the non-linear prediction models for classification and algorithm selection. The hardest challenge for the point will be run-time efficiency, which might rule out kernel-based and other nonparametric, methods. At the same time, we use the reinforcement learning technology to carry out dynamic algorithm selection.

There are two fairly robust inductive models: a symbolic one, i.e., decision trees, and a functional one, i.e., Support Vector Machines (SVMs) [13]. They are also “orthogonal”, since the algorithms they use are based on radically different approaches to the dataset. Our new SVM classification algorithm integrate the merit of both, Shows more powerful ability of classification, inductive decision and algorithm selection. And our work herewith presented provides more flexible framework, analyses and further better experimental results than in the parts common to [2]. Moreover, we improve the process of algorithm selection, which upgrade the result in [4] and [2].

3 A New SVM Classification Algorithm

3.1 Algorithm Foundation

At the present time, Support Vector Machine (SVM) developed in recent years based on statistical learning theory is popularly regarded as one of the best Learning Machine, the principle of which is structuring the classification hyper-plane has maximum margin in order to guarantee it having the strongest generalization ability.

The decision tree learning is one of the most often used inductive reasoning algorithms, which is a method of approximation to discrete function, in addition, it has good robustness to noise data. The function having been learned in this way is represented as a decision tree; the decision tree learned also can be represented as more than one if-then rule in order to improve the readability.

We can combine these two important inductive approaches and make them complement each other, because the SVM has a strong ability of generalization but the knowledge gained from it, namely, hyper-plane was barely intelligible, the decision tree has an ordinary generalization ability but the knowledge induced from it is readily comprehensible, namely, we look on the maximum margin as a kind of heuristic information, to lead it into the construction of learning system by using the inverse problem of SVM. We design and implement the decision tree induction algorithm based on maximum margin according to this consideration.

The theory of maximum margin is the most important component element of SVM theory, which is believed big margin implicates good inductive capability [5]. Consider this relationship, we lead the maximum margin theory into the decision tree induction algorithm, using the maximum margin as the heuristic information for decision tree induction in order to improve
the generalization ability of decision tree. So we always seek the classification plane having the maximum margin. The definition of the classifier margin is showed in [13].

In this section, we study the application of SVM in decision tree induction. The general process of generating the decision tree is usually as follows: firstly, we take the entire training sample as the root node, thereafter according to one heuristics information, we divide downward the root nodes into some child nodes, if the child node sample is belong to the same kind, this child node is regarded as a leaf node, otherwise, continue the division until the leaf nodes appear. The process of constructing the tree is the process for dividing the nodes. However, the inverse problem of SVM is just studying the division of sample in order to seek the division having maximum margin, therefore, we apply the inverse problem of SVM to the inductive process of decision tree, using the maximum margin as heuristics, we can obtain the corresponding division by solving the inverse problem of SVM and construct the decision tree in this way.

3.2 Algorithm Design

According to the above description of the maximum margin theory, we can apply the thought of maximum margin to the decision tree, namely apply the inverse problem of SVM to the induction procedure of decision tree, and use the maximum margin as heuristics in the induction procedure.

The design philosophy is showed as follows: For a training sample set which each given attribute value is continuous type data, we do not consider the category of samples, through solving the inverse problem of SVM, we can get a division of this sample, and this division has a maximum margin, namely divide the sample set into two subset, and these subset are used as the branches of the decision tree, the sample set labeled -1 is usually quoted as left branch and the sample set labeled +1 is usually quoted as right branch, the hyper-plane corresponding to this division is quoted as the decision function at corresponding node. When we test the new samples, insert them into the decision function, the samples having positive output value are assigned to the left branch, the samples having negative output value are assigned to the right branch.

To take two type problems for examples, the concrete algorithm is showed as the function MaximumM arg in (Examples, β) followed:

In this function, dataset Examples is the training sample set, β is a threshold value of the ratio of a sample category, the function returns a decision tree which can classify the given sample set Examples.

The algorithm works as follows:

Step 1: Creating the root node of the decision tree;
Step 2: If the ratio of positive class in Examples is greater than β, the function returns a single tree root node labeled plus sign +;
Step 3: If the ratio of negative class in Examples is greater than β, the function returns a single tree root node labeled subtraction sign −;
Step 4: Otherwise,
(1) Solving the inverse problem of SVM on Examples, then get the division has maximum margin;
(2) Let the hyper-plane we get is function \( f(x) \), if \( f(x) \leq 0 \), let \( c(x) = -1 \), if \( f(x) \geq 0 \), let
\[ c(x) = +1; \]

3. Adding two branches under the root, corresponding to \( f(x) \leq 0 \) and \( f(x) > 0 \), respectively;

4. Let \( \text{Examples}_1 \) is the subset satisfies \( f(x) \leq 0 \) in \( \text{Examples} \);
Let \( \text{Examples}_{+1} \) is the subset satisfies \( f(x) > 0 \) in \( \text{Examples} \);

5. Adding the sub tree \( \text{MaximumM} \arg\max(\text{Examples}_1, \beta) \)
And \( \text{MaximumM} \arg\max(\text{Examples}_{+1}, \beta) \) under these two branches, respectively;

Step 5: End;

Step 6: Return the root node;

Obviously, the time complexity of our algorithm is \( O(2^{k-1}n^2) \). we elide the process of Algorithm analysis.

4 Process of QBF Hybrid Solving

4.1 Feature Choice

We use the QBF features described in [4]. In order to perform inductive inference on QBFs, we transform them into vectors of numeric values, where each value represents a specific feature. In particular, we consider several basic features, such as, e.g., the number of clauses \( c \), the number of variables \( v \), and the number of quantifier alternations. Among these features we also tried to consider parameters that should be somehow related to the algorithmic properties of different engines such as, e.g., the proportions in the number of universal and existential variables/literals, and the number of positive/negative occurrences of the variables [4]. We also consider combined features, i.e., ratios and products of the basic features. These include, e.g., the product of the positive and negative occurrences of a variable, and several ratios including, e.g., the clause-to-variables ratio, the universal-to-existential ratio, and the positive-to-negative occurrences ratio.

On the other hand, we also consider some special complex feature, e.g., quantifier structure, quantifier trees and tree-width of QBFs. These features can classify the formula and construct the relationship between formula and algorithm effectively.

4.2 Classification and Inductive Models

The second design issue concerns the inductive models to implement in our solver. An inductive model is comprised of a classifier, i.e., a function that maps an unlabeled instance (a QBF) to a label (a solver), and an inducer, i.e., an algorithm that builds the classifier. In the following, we call training set the dataset on which inducers are trained, and test set the dataset on which classifiers are tested. While there is an overwhelming number of inductive models in the literature, we can somewhat limit the choice considering that our solver has to deal with numerical attributes (QBF features) and multiple heuristics class labels. Moreover, we would like to avoid formulating specific hypotheses about the features, and thus we prefer inducers that are not based on hypotheses of normality or dependence among the features. Finally, we also prefer non-linear prediction models for algorithm selection. Considering all the above, we chose to implement our new SVM classifier introduced in Section 3 to our hybrid solver.
The SVM classifier has advantage on small sample. Our SVM Inductive Models is based on a small number of problem instances, and it ban be expected to behave similarly on future unseen problem instances. We can apply the optimal classification function Eq. (3) of SVM:

$$f(x) = sgn\{\omega^* \cdot x + b^*\} = sgn\left\{\sum_{i=1}^{l} \alpha_i^* y_i (x_i \cdot x) + b^*\right\}$$

We compute the classification function for very heuristic in the sequential QBF solver. In the process of solving, we classify the features of the instance, when the value is matching to the range of value for optimal classification function, we can switch to the matching heuristic to resolving the instance.

### 4.3 Solvers and Heuristics Choice

We apply the solvers used in AQME [4], so We include four search solvers, namely QuBE7, ssolve, quaffle, and yQuaffle, and three combinatorial ones, namely 2clsQ, quantor, and sKizzo, and another special one, StruQS. In these solvers, each one has different heuristic algorithm. So we have eight different heuristics in our hybrid solvers in all. The first four are search-based solvers and the following three are variable-elimination-based ones. And StruQS is a solvers Integrating the deduction and Search. Its main feature is a dynamic combination of search — with solution and conflict backjumping and variable elimination. The key point in this approach is to implicitly leverage graph abstractions of QBFs to yield structural features which support an effective decision between search and variable elimination [22].

### 4.4 Reinforcement Learning for Algorithm Switching

In the research areas of algorithm selection, the reinforcement learning is a sophisticated technology which has been used in many algorithm selection problems, such as [18] [19] [20]. In [4], the author use retraining mechanism to realize the different algorithm switching, its main advantages are simplicity, and independence from the specific inducer used to learn the engine prediction policy. But it needs a static initial training process. In this paper, we adopt the reinforcement learning described in [18] as the algorithm switching mechanism. Our learning mechanism is a variation of the well known Q-learning algorithm, adapted to account for multiple state transitions. The general update equation of Q-learning is Eq. (4):

$$Q^{(t+1)}(s_t, a_t) = (1 - \alpha)Q^{(t)}(s_t, a_t) + \alpha[R_{t+1} + \min_a \{Q^{(t)}(s_{t+1}, a)\}]$$

where $s_t$ is the state at time $t$, $a_t$ is the action taken at time $t$, $R_{t+1}$ is the one-step cost for that decision, and $\alpha$ is the learning rate. If $a_t$ is a non-recursive algorithm, the resulting state is terminal and has a cost of 0, so the update rules, reduces to Eq. (5):

$$Q^{(t+1)}(s_t, a_t) = (1 - \alpha)Q^{(t)}(s_t, a_t) + \alpha R(s_t, a_t)$$

In this paper, we adopt eight different solvers, there are eight different heuristics algorithm. On a macro perspective, it is not a recursive procedure. So we apply the Eq. (5) as the learning rule. The switching paradigm allocates run-time to the pure eight heuristics during the solving process.
Very heuristic has a weights. During each iteration, the run-time of each pure algorithm, \( a \in A \), is determined using a weight \( w_i(a) \), which is learned as the solving progresses. The weights for an iteration are normalized to sum to 1 and therefore \( w_i(a) \) corresponds to the fraction of time \( t_i \) allocated to algorithm \( a \). Weights are updated after each iteration by considering the current weight and the performance of each algorithm during the last iteration. Performance is measured in cost improvement per second, which allows us to compare algorithms despite the differing run times. We normalize the cost improvement per second to sum to 1 producing a performance value \( p_i(a) \). The weights are then adjusted using the reinforcement learning formula Eq. (5). According to the weights, the different heuristics algorithm can get running time allocation [20]. In Fig. 1, the execution loop of algorithms selection is described.

4.5 Process of Solving

The process of solving is similar to the method in [4], but we revise the classification algorithm and adopt the reinforcement learning as the algorithm switching mechanism. An explanation of solving process proceeds as follows:

1. Given a small quantity of test set, we conduct the classification training; construct the relationship between the features and heuristic.

2. Given a QBF \( \phi \), extracts the features and it stores them in a vector \( \Phi \).

3. The classifier predicts the engine to be run on the basis of \( \Phi \), and it launches such engine with a time limit \( \tau \).

4. The reinforcement learning is activated to selection a pure solver to solve the formula.

5. If \( \phi \) is solved, the procedure ends, else the reinforcement learning procedure is called. Reinforcement learning starts according to three key parameters: \( \tau \), the order on which alternative engines are tried, and the time \( \tau \) granted to each such trial. If the reinforcement learning finds an alternative engine which solves \( \phi \), then our hybrid solver updates its internal models accordingly, adding the features vector \( \Phi \) (labeled with the engine) to the training set on which the classifier is trained, and a new inductive model is generated as result of this operation.

Obviously, the exact values of the above parameters may depend on the formulas to be solved, on the run-time distributions of the basic engines and on the inductive models used to learn the selection policy. The reinforcement learning procedure may not start if the formula is solved in the initial state. The whole process is illustrated in Fig. 2.
Fig. 2: Execution loop of solving

5 Experimental Evaluations

To evaluate our algorithm architecture, we ran few preliminary tests on some benchmark of QBFLIB (www.qbflib.org). All the experiments that we present hereafter ran on a single PC, running the environment is Ubuntu-11.10-desktop/GNU Linux. On all test runs the CPU time limit was set to 600 seconds.

In order to compare the solution efficiency, we choose three state-of-the-art solvers, namely three sequential solvers Quantor, QuBE7 and sKizzo. Quantor is based on variable elimination and expansion, plus a number of features, such as equivalence reasoning, subsumption checking, pure literal detection, unit propagation, and also a scheduler for the elimination step. QuBE7 is an efficient search-based solver for Quantified Boolean Formulas (QBFs). Maybe it is the best search-based solver. sKizzo is a powerful solver based on a new technique, called symbolic skolemization, and on a related form of symbolic reasoning. This approach makes it differ from all the previous QBF solvers. This approach makes it differ from all the previous QBF solvers. In fact, QUANTOR and sKizzo are both based on variable elimination, but the elimination method is different. All the solvers run the same benchmarks on the same single machine.

We choose the QBFs Family Counter as the test set, it is in the Formal Verification domain. The QBFs made up by encoding of formal verification problems represent a reasonably difficult test set. Family Counter has 88 instances; we only describe the experiments result of 10 instances because of the space restraint of this paper.

Table 1 is the cumulative CPU time to solve the family Counter QBFs (unit is second (s)). We can see from the table that our solver SVM-solver spends more time on some benchmarks, but spend less time on other benchmarks. Three sequential solvers QuBE7 and sKizzo spend no time from cnt01 to cnt05, because these instances are very easy formula, the sequential can resolve them directly. But our solver SVM-solver is a hybrid solver, it must spend a little time to run the classification algorithm in order to accomplish the best algorithm selection. For the more hard formula instances cnt09 and cnt10, four solvers all spend much time, the time gap is not very obvious, but our solver is obvious dominant to other three solvers. It is an important result that our solver is better suited to hard problems.

In the second test, to prove the effectiveness of our methods, the number of benchmarks must be big, so we choose the QBFs Family Counter, C432, Debug, s3271, term1 as the test set, they are all in the Formal Verification domain. In each family, we choose ten formulae as the test set.
We can see the similar result from Table 2. From the above benchmarks, we can see the SMQBF is almost a little stronger than QuBE7. And it is stronger than other two solvers. It is obviously the best solver in this test.

The third experiment detailed in this section is carried out on the same computing platforms above, but here we focus on the following QBF encodings [14]:

Katz: QBFs resulting from the encoding of symbolic reachability for industrially relevant circuits (20 instances) [15].

Tipdiam: QBFs resulting from the encoding of symbolic diameter calculation for a variety of circuits (40 instances) [16].

These two families are the formulas whose Gaifman graphs have typical small world structure [21]. Some relevant features of the above encodings are summarized in [14], the author analyzes these two families and gets this conclusion. The density distribution of these families’ graph structure can reach relatively high values. These two set match the small world topology more closely than the above families. It means that these two QBF encodings are very difficult to solve.

Table 3 is the cumulative CPU time to solve the benchmarks family Katz and Tipdiam, it reports the number of instance solved by four solvers (Number) and the total CPU time spent on such instances (Time). A dash on both columns means that the solver does not solve any formula. We can see that all the solvers fell hard to these two families. They all can’t resolve all the formulas. But our hybrid solver SVM-solver is the best in other three solvers, it can resolve more numerous formulas than others, even it spends less time. On this experimental data, our hybrid solvers SVM-solver can resolve more numerous formulas than sequential solvers QuBE7, sKizzo and Quantor. We are surprised at its high efficiency. It proved that non-linear prediction models SVM and reinforcement learning technology is very effective in algorithm selection for QBF solving.

In the last experiment, we use a large test set having 350 formulas, including Katz, Tipdiam and Formal Verification domain problem. We run the above four solvers on this special test set. We record the number of instance solved at some time node, i.e. how many formulas are solved when time pasts 10 s, 100 s and 200 s. As shown in Fig. 3, the SVM-solver is in fact able to solve...
Table 2: Solving time for benchmarks of domain formal verification

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>SVM-solver</th>
<th>QuBE7</th>
<th>sKizzo</th>
<th>Quantor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter (10)</td>
<td>7.56</td>
<td>7.47</td>
<td>7.5</td>
<td>7.56</td>
</tr>
<tr>
<td>C432 (10)</td>
<td>6.32</td>
<td>7.34</td>
<td>7.42</td>
<td>7.68</td>
</tr>
<tr>
<td>Debug (10)</td>
<td>8.37</td>
<td>8.53</td>
<td>8.56</td>
<td>8.67</td>
</tr>
<tr>
<td>s3271 (10)</td>
<td>7.93</td>
<td>8.12</td>
<td>8.09</td>
<td>9.32</td>
</tr>
<tr>
<td>term1 (10)</td>
<td>7.72</td>
<td>7.96</td>
<td>7.99</td>
<td>8.12</td>
</tr>
<tr>
<td>Total</td>
<td>37.9</td>
<td>39.42</td>
<td>39.56</td>
<td>41.35</td>
</tr>
</tbody>
</table>

Table 3: Solving time for benchmarks of hard formula family

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>QuBE7</th>
<th>sKizzo</th>
<th>SVM-solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Time</td>
<td>Number Time</td>
<td>Number Time</td>
<td>Number Time</td>
</tr>
<tr>
<td>Katz (20)</td>
<td>–</td>
<td>–</td>
<td>9 80.76</td>
</tr>
<tr>
<td>Tipdiam (100)</td>
<td>36</td>
<td>312.73</td>
<td>75 340.45</td>
</tr>
</tbody>
</table>

Fig. 3: Time scale comparison of several solvers on big benchmarks

more instances than other solvers. At the beginning, SVM-solver does not show the advantage, however, as time grows, it can solve more formula than other solvers. When the time pasts 200 s, Performance of Quantor and QuBE are declining, performance of SVM-solver is stable growth trend. This shows that it has a better robustness. Hence, by simply employing dynamic algorithm selection technology, it is possible to gain an advantage over such sophisticated QBF solvers as QuBE in some respects.

From the above result, it is fair to say that SVM-solver alone is able to solve a fairly large number of instances, and most of these instances are those that cannot be solved by other solvers in their original formulation. It shows how SVM-solver can be beneficial independently of the algorithm featured by the solver. The improvement is more substantial than pure sequential QBF solver.
6 Conclusions

This paper builds on and extends [2, 4], but focus different aspects of the study and improve the method to some extent. In this paper, we propose a new SVM classification algorithm, apply this algorithm in QBF hybrid solving and design a new QBF hybrid solver. This solver apply SVM algorithm to construct inductive models. We also apply reinforcement learning as the dynamic algorithm selection technology to realize the algorithm switching. The relationship between the solving algorithm and formula is established according to test formula set. When the first heuristic fails in resolving the formula, the dynamic algorithm selection procedure can update the inductive models and select another solving algorithm automatically. It can drastically improve the performance of a QBF solver and resolve more numerous formulas than sequential QBF Solver.

It proved that non-linear prediction models SVM and reinforcement learning are very useful in dynamic algorithm selection for QBF hybrid solving. We believe that machine learning technology can be helpful to a much larger degree when solving hard problems. Further directions for future work include optimal feature selection, and the use of optimal dynamic algorithm selection technology, e.g., applies reinforcement learning in SVM inductive models. The hardest challenge for the latter will be run-time efficiency and apply stronger inference or partitioning in classification.

Acknowledgments

The authors would like to thank Luca Pulina in University of Genoa for very fruitful help.

References

[8] A. Biere, Resolve and expand, Lecture Notes in Computer Science, 3542 (2005), 59-70
[10] Marco Benedetti, sKizzo: A suite to evaluate and certify QBFs, Lecture Notes in Computer Science, 3632 (2005), 369-376
[14] Luca Pulina, Armando Tacchella, An empirical study of QBF encodings: From treewidth estimation to useful preprocessing, Fundamenta Informaticae, 102 (2010), 391-427