A Machine Learning Technique for Hardness Estimation of QFBV SMT Problems (Work in progress)

Mohammad Abdul Aziz, Amr Wassal and Nevine Darwish

Computer Engineering Department, Cairo University, Cairo, Egypt
mohammad.abdulaziz8@gmail.com
a.wassal,ndarwish@ieee.org

Abstract

In this paper, we present a new approach for measuring the expected runtimes (hardness) of SMT problems. The required features, the statistical hardness model used and the machine learning technique which we used are presented. The method is applied to estimate the hardness of problems in the Quantifier Free Bit Vector (QFBV) theory and we used four of the contesting solvers in SMTCOMP2011 to demonstrate the technique. We have qualitatively expanded some propositional SAT features existing in the literature to directly work on general SMT problem instances without preprocessing. Experimental results with the standard set of benchmarks are promising and our implementation proves the concept.

1 Introduction

Recently SMT has been attracting a lot of research interest because of the wide range of applications where it can be applied. It has been extensively used for formal verification of hardware, software, security protocols as well as other applications. Interest in SMT solvers research is increasing because of the expressive power they offer compared to SAT solvers. They allow easier ways to solve more complex problems. In addition, due to the theory specific decision procedures on which they are based they give a better performance than general first order logic solvers. In [1] a very good treatment of different SMT solving techniques, applications and other issues is presented.

Accurate hardness (i.e. solution runtime) evaluation of SMT problem instances will impact building SMT solvers portfolios, where it can help in choosing the solver having the least expected runtime, portfolios between SMT solvers and solvers of other types of encodings of the same problem, where it can be used to choose an SMT solver or another solver of a different encoding of the same problem whichever has less expected runtime. The proposed approach is also characterized by efficient feature extraction time. This is achieved by devising features that work directly on general problems without any need for the problem to be in a standard or normal form. In our research, we have used problem benchmarks from the SMT-Lib benchmarks repository. This constitutes a large variety of problems; from industrial problems to randomly generated problems to hand crafted problems. [11] is a good reference about the SMT-Lib initiative, the SMT-Lib v2 language, the logics it supports, the theories it supports and the compatible solvers. We have chosen QFBV logic because the largest repository of benchmarks is for that logic. It also has wide applications in the verification of systems such as software verification, hardware verification, cryptographic protocols verification and different encodings of various classical optimization problems. Nevertheless, the method that we propose can be generalized to other logics.

In the next section we will refer to relevant work in the literature. In section 3 we will describe our main approach and our assumptions regarding the SMT problems and their hardness. In section 4 we describe the features that we used to characterize the SMT problems. In section 5 we describe our experimental setup and the experimental results that we obtained. In sections 6 and 7 we conclude and state our suggested steps for future work.

2 Related Work

The literature is rich with publications related to empirical hardness evaluation of problems in Artificial Intelligence (AI). Other relevant efforts have been addressed in research related to algorithm selection. References like [2, 3, 4, 5, 6] address these questions in a statistical machine learning setting. Based on a training set of performance data for a large number of problem instances, a model is learned that maps (problem, algorithm) pairs to expected performance. The most relevant to our work is the Estimation of hardness of propositional SAT problems. This work is well established and has started early since [7] and [8], which considered the empirical performance analysis of DPLL-type SAT solvers running on uniform random k-SAT instances. Specifically, it finds a strong correlation between the instance's hardness and the ratio of the number of clauses to the number of variables in the instance. For example, it is empirically well established that for random SAT problems the hardest clause to variable ratio. For instance, for random 3-SAT a clauses-to-variables ratio of roughly 4.26 corresponds to a phase transition in an algorithm-independent property of the instance. In other words, this transition is in the probability that a randomly generated formula having a given ratio will be satisfiable. This discovery and others such as islands of tractability in [18], search space topologies for stochastic local search algorithms in [16] and [17], backbones in [19], backdoors in [20] and random restarts in [15], prove that empirical complexity analysis of the problem is useful in discovering some inherent properties in the SAT problem instances independent of the solver's algorithm as well as deepen our understanding of the SAT problem in general. Most relevant to our approach is the work in [9], which is the approach that we have used as a starting point to our work.

3 Main Approach

In this paper we try to devise a technique that can be used in estimating the hardness of SMT problems. Our technique deals directly with general SMT problems i.e. no preprocessing of the problem is required and the problem doesn't have to be in some normal form (such as 3-CNF). We will first start with our the general model of the SMT problem that we chose. Next we describe how an SMT solver solves it and how should this affect the expected hardness model. The SMT problem can be modeled by the tree as shown in Fig. 1. In this model, the problems are made of a tree whose inner nodes are functions and the leaf nodes are uninterpreted constants. In the shown tree the white circular nodes are Boolean functions which belong to the core theory (for instance $\bigwedge$, $\bigvee$, $\longrightarrow$, etc...), the grey rectangular nodes are theory T functions which have a Boolean codomain, and a theory sort domain (such as bvuge in the QFBV theory) and the grey triangular nodes are functions which have as a domain and a codomain T theory sorts (such as bvadd in the QFBV theory). The black and grey circular nodes are Boolean and theory T uninterpreted constants, respectively. An SMT solver of the lazy approach will solve this problem by dealing with the entire problem as a propositional SAT problem, where it will search for Boolean assignments of Boolean uninterpreted constants and the theory atoms

(subtrees which have the rectangular nodes in the tree representation as their heads). After the SAT solver finds a model for this SAT problem, the assignments of the theory atoms are delivered to the theory specific decision procedure, which checks whether this assignment of the theory atoms is satisfiable or not according to the structure of the theory subtrees. It passes the justification in case of an unsatisfiable assignment, or it returns the model of the theory uninterpreted constants if the model is satisfiable. We hypothesize that the hardness H can be modeled as follows

$$H = f_a(H_{SAT}, H_{TH}, H_{HYBRID}). \tag{1}$$

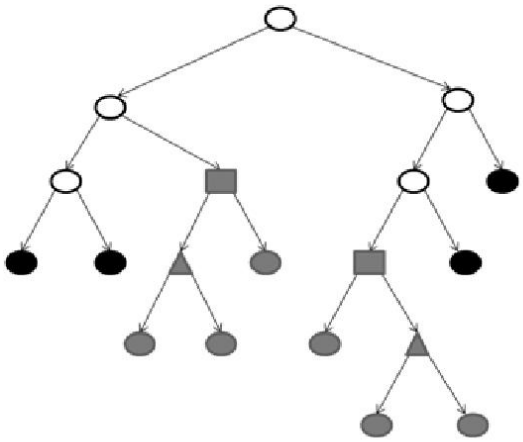

Figure 1: A graphical representation of an SMT problem. The white circular nodes are boolean functions, the black circular nodes are boolean uninterpreted constants and the grey nodes are T theory functions or uninterpreted constants.

where the variable H_{SAT} is the hardness of the propositional SAT structure of the SMT problem, H_{TH} is the hardness of the theory formula, which contains the conjunction of theory atoms in the SMT problem and the variable H_{HYBRID} is the hardness introduced by the interactions and dependencies between the propositional SAT and the theory portions of the problem and $f_a()$ is a function that increases with H_{SAT}, H_{TH} and H_{HYBRID}. According to the way we modeled the hardness of an SMT problem, a statistical hardness model should include three disjoint sets of features. Each of these sets should be estimating one part of the hardness of the problem. We assume that each of the parts of the problem hardness will be a linear combination of the set of its corresponding set of features. This means that each part of the hardness can be represented by this equation

$$H_p = k_p + \sum_{i=1}^{n_p} \alpha_i x_{p_i}, [p : p \in [SAT, TH, HYBRID]], \tag{2}$$

where x_p is a feature in the set of features of the hardness part p and n_p is the number of features to represent the part p and k_p is a constant. We put this methodology in mind to devise features for each of the parts of the problem hardness.

We follow the assumptions used by [4] to describe the model for the probability distribution of the average runtimes of a propositional SAT problems. In this model, a problem instance i

represented by a feature vector x_i has a normal probability distribution for its expected runtime with average $f_a(x_i)$ and unity variance, where $f_a()$ is the hardness model of the solver algorithm a.

According to these assumptions from [4], the distribution of the runtime t_i of the problem can be given by the equation

$$P(t_i|x_i, a) = \frac{e^{\frac{-(t_i - f_a(x_i))^2}{2}}}{\sqrt{2\pi}} \ . \tag{3}$$

Therefore for predicting the expected runtime of an instance we will have to learn the hardness model $f_a()$ given the set of (feature vector, runtimes) mappings for a set of problem instances. We chose the model to predict the logarithm of the run time instead of the runtime itself to make training the model more efficient and easier due to the large variation in the runtimes that solvers experience according to [4].

We chose the function $f_a()$ to be a linear model, where a vector w, of coefficients to be multiplied by the feature vector x_i and a constant term, is used to get the expectation of the natural logarithm of the runtime. That is the expected logarithm y_i of the runtime is given by

$$y_i = f_a(x_i) = w^T.x_i \ . \tag{4}$$

According to this linear model, the expected runtime is a weighted linear combination the features that represent the problem instance and since this set of features is the union of the hardness parts sets of features, then the function $f_a()$ which is equivalent to the total expected hardness will be

$$f_a(H_{SAT}, H_{TH}, H_{HYBRID}) = H = \alpha_1 H_{SAT} + \alpha_2 H_{TH} + \alpha_3 H_{HYBRID}. \tag{5}$$

which is the weigthed sum of the different parts of the problem's hardness where α_1,α_2 and α_3 are the corresponding weights.

To get the vector of coefficients w, we used ridge regression. Specifically to get w we will have to solve the matrix equation

$$Xw = Y \ . \tag{6}$$

where X is the matrix containing the concatenation of the feature vectors of the training set x_i, Y is the vector containing the concatenation of the runtimes y_i corresponding to each of the feature vectors in X. We use ridge regression to get the solution for w, which is

$$w = (X^T X + \alpha I)^{-1} X^T Y \ . \tag{7}$$

where α is the Tikhonov factor (Regularization Factor), I is the identity matrix.

Although it has a limited hypothesis space, we chose a linear model because of the efficiency of learning it as well as using it to calculate the predicted runtimes which can be useful in applications such as portfolios.

4 Features

Features chosen are to estimate the first two components in the assumed model of the runtime, namely, H_{SAT} and H_{TH}. We assume that H_{HYBRID} will be represented only in the constant factor in the linear model. For the features that represent the propositional SAT structure

of the SMT problem, we have used some of the features in [9] as our starting point but after adapting them to suit the richer expressive power of SMT compared to conjunctive normal form (CNF) SAT problems treated in [9] as well as the general structured problems that we want to tackle. The first set of features we used was the number of clauses, the number of variables, the ratio of clauses to variables and its reciprocal. The equivalent set of features we used for these were the following; associated with each of the modified features is a qualitative justification we used to derive it:
I)For the number of clauses we used the following features:

- N_{ANDS}= The number of arguments to all the $\bigwedge$ functions in the SMT problem.

Justification: each $\bigwedge$ function argument can be considered as a clause.

- N_A= The number of asserted Formulas

Justification: because the SMT problem is equivalent to a clause containing the conjunction of all the assertions in the problem.

- N_{EQUAL}= The summation of the number of the arguments to all the "=" functions in the SMT problem. This includes those whose arguments are boolean variables only.

Justification: we first derive the number of clauses needed to represent a single equal statement. $x_1 = x_2$ is equivalent to two clauses which are $(x_2 \bigvee \neg x_1) \bigwedge (x_1 \bigvee \neg x_2)$.

An equal statement containing n arguments ($x_1 = x_2 = x_3 = \ldots = x_n$) can be represented by a conjunction of 2 argument equalities of the form $(x_1 = x_2 \bigwedge x_2 = x_3 \ldots x_{n-2} = x_{n-1} \bigwedge x_{n-1} = x_n$), which will be a conjuntion of $(n-1)$ corresponding 2 argument equality statements. Since each 2 arguments equality is equivalent to a couple of binary clauses, then the n arguments equality statement can represented by $2(n-1)$ binary clauses.

- $N_{DISTINCT}$= The summation of the number of the arguments to all the distinct functions in the SMT problem (This has the same justifications of the "=" function).

- N_{IMPLY}= The number of instances of $\longrightarrow$ function in the SMT problem.

The justification:
$a \longrightarrow b$ is equivalent to $(\neg a \bigvee b) \bigwedge true$, which is equivalent to one clause.

- N_{XOR}= The number of the xor functions in the SMT problem.

The justification:
$xor(a, b)$ is equivalent to $(\neg a \bigvee \neg b) \bigwedge (b \bigvee a)$, which is equivalent to two clauses.

- N_{ITE}= The number of instances of *ite* function.

The justification:
$ite(a, b, c)$ is equivalent to $(b \bigvee \neg a) \bigwedge (a \bigvee c) \bigwedge (b \bigvee c)$, which are 3 clauses.
II)For the number of variables we used:

- N_{BOOL}= Number of Boolean uninterpreted constants(These are variables to find a substitution for.)

- $N_{THEORY-ATOMS}$= The number of theory atoms in the SMT problem(The justification for this is that; for the propositional SAT solver part of the SMT solver, the theory atoms are substituted and dealt with as if they are Boolean variables i.e. They are assigned the values either true or false).

III)To measure the equivalent number of clauses, we have weigthed the number of instances of each of the boolean functions, and accordingly for the clauses to variables ratio and its reciprocal we used:

- $C/V = (N_A + N_{ANDS} + 2(N_{EQUAL} - 1) + 2(N_{DISTINCT} - 1) + N_{IMPLY} + N_{XOR} + 3N_{ITE})/(N_{BOOL} + N_{THEORY-ATOMS})$
- V/C

The next two subsets of the features of the propositional SAT structure of SMT problems are adapted versions of the graphical representation of propositional SAT problems used in [9], namely; The Variable Clause Graph, and the Variables Graph. We have replaced the variable-clause graph with three bipartite graphs. Each of these graphs contained a party for the boolean variables and another one for the instances of one of the functions that comprised the clauses in the SMT problem. An edge exists between a node in the variables party and a node in the function instances party if this variable is an argument for this function instance. We have three of these graphs; one for the $\wedge$ function, another for the $=$ function and a last for the distinct function. We expect that most of the constraints in the propositional SAT problem are represented in those functions, due to their massive usage of them in the benchmarks. We also generalized the Variables Graph to a graph which has a node per Boolean uninterpreted constant, where an edge exists between any two nodes if they are common arguments for one or more Boolean function (given that the function is one of the functions which are considered in the number of clauses features set).

For the features which we used to estimate the theory part of the SMT problem, which in our case was QFBV, we used the total number of bits in the problem (i.e. the summation of the lengths of the bitvector uninterpreted constants in the problem)$V_{BIT-VECTORS}$, and the number of instances of QFBV functions used in the SMT problem. Fig. 2 shows a list of the 71 used features classified according to the category of the feature.

5 Experimental Setup and Results

We use benchmarks on the QFBV benchmark repository of SMT-Lib. We chose this benchmark repository because it contains a wide variety of problems; industrial, randomly generated problems and handcrafted problems. We have used all the 33000 of these bechmarks for training and validation and to obtain the average of 10-fold validation root mean square error (RMSE). We used a feature extraction timeout of 20 seconds, which resulted in using 23433 problem instances in obtaining our results, with an average feature extraction time 2.453 seconds. Of these problems there was 10353 satisfiable instances and 13080 unsatisfiable instances. The only preprocessing step which we applied to the SMT problems was to remove the let bindings by substituting them with their corresponding bindings. For the runtimes, we have used the runtime data from the difficulty-computation-2012 on the SMTEXEC cluster where a timeout of 1800 seconds limited the runtimes of the solvers. Interested readers can refer to [12] for more details regarding the benchmarks and the SMTEXEC cluster used for collecting the runtime data. We have also added 0.5 to all of the runtimes, because limits on the resolution of the runtimes reports by SMTEXEC have led to problem instances which have runtimes of zero seconds, which could have led to a logarithm of $-\infty$. After extracting the features we normalized each of them to be between 0 and 1. For learning the linear model, we used ridge regression with a Tikhonov factor of 0.05 (as recommended in the work of [9]).

I. Propositional SAT features

1. Boolean variables and Clauses features:

N_A

N_{ANDS}

N_{EQUAL}

$N_{DISTINCT}$

N_{IMPLY}

N_{XOR}

N_{ITE}

N_{BOOL}

$N_{THEORY-ATOMS}$

V/C

C/V

2. Boolean Variables graph features:

-Boolean variables nodes degree average, entropy, maximum, minimum and variation coefficient.

3. Boolean Variables-$\bigwedge$ function graph features:

-Boolean variables nodes degree average, entropy, maximum, minimum and variation coefficient.

-$\bigwedge$function node degree average, entropy, maximum, minimum and variation coefficient.

4. Boolean Variables-distinct function graph features:

-Boolean variables nodes degree average, entropy, maximum, minimum and variation coefficient.

-distinct function node degree average, entropy, maximum, minimum and variation coefficient.

5. Boolean Variables-= function graph features:

-Boolean variables nodes degree average, entropy, maximum, minimum and variation coefficient.

-=function node degree average, entropy, maximum, minimum and variation coefficient.

II. QFBV theory specific features

1. Number of theory variables:

$V_{BIT-VECTORS}$

2. Instances of QFBV theory functions:

bvand, bvsub, bvuge,bvugt, bvxor, bvnot, bvneg, bvor, bvadd, bvmul, bvudiv, bvxor, bvsle, shlok, addok, bvashr, bvsdiv, bvurem, bvshl, bvlshr, bvult, bvslt, bvule.

Figure 2: A list of the features used for estimating the hardness of SMT problems, classified according to their category.

Solver/Model	SAT	UNSAT	SAT/UNSAT
BOOLECTOR	0.1494	0.3082	0.2672
MATHSAT	0.1353	0.5851	0.2155
SONOLAR	0.1920	0.3877	0.2320
STP2	0.1346	0.3442	0.3166

Figure 3: The RMSE of the prediction of the logarithms of the runtimes for different models for the different solvers.

The solvers which were used to get these runtimes were Boolector, MATHSAT, STP2 and SONOLAR. These solvers are based on different approaches for solving the QFBV problems. Boolector, SONOLAR and STP2 use bit-blasting to solve QFBV SMT problems ([13], [14] and [22]). Mathsat, on the other hand, uses the lazy approach to solve the QFBV SMT problems ([21]).

We have used directly all the features demonstrated in figure 2, except for the features that were representing unused functions in all the problem instances (e.g. the XOR function). Using Ridge regression, we learnt a model for each of the previously stated solvers that predicted the logarithms of the runtimes. Figure 4 shows the correlation between the predicted logarithm

of runtimes and the actual logarithms of runtimes. The 10-fold cross validation RMSE of the prediction of the runtimes logarithm for each of the solvers' model are in the third column of the table in figure 3.

According to [9], the prediction performance of a propositional SAT problem instance hardness model will significantly improve if a model was devised for satisfiable problems and another for unsatisfiable problems. We tried this for QFBV SMT problems, and the results were an improvement in the RMSE of the satisfiable problems model. On the other hand, a detorioration has happened to the RMSE of the models of the unsatisfiable problems. Figure 5 shows the correlation of the predicted and the actual runtimes for the satisfiable and the unsatisfiable problem instances for the different solver models. The first and second columns in the table in figure 3 show the RMSE of the satisfiable and unsatisfiable instances respectively for the different solvers. However, these results for the SMT problems at hand contradict with the results reached in [9], where here the satisfiable problems are more correlated with the prediction unlike the results reached in [9], where the unsatisfiable problems were the more correlated. One possible explanation of the uncertainty of the predictions of the unsatisfiable problem instances, is that we did not use any of the DPLL probing features, which were the most important in predicting the runtimes of unsatisfiable problem instances according to [9]. This seems reasonable, because the runtimes of the unsatisfiable problem instances will most probably be discriminated with reasonable certainty by having a low number of assigned variables by applying a probing technique for a fixed time.

6 Conclusion

To conclude, we have devised a technique for the hardness estimation of SMT problems, which up to our knowledge, is the first trial of its kind. We achieved relatively good results with cheeply computed features on a large variety of problems and solvers. Although our approach was based on an approach targeting random SAT problems and was based on a linear model, it worked on problems whose majority are industrial. The main advantages of our work are both its simplicity and generality. It can be used for general SMT problems. It provides a general technique which is based on the structure of the SMT problem independent of the underlying logic(s). The features we used spared us from any preprocessing of the problems, such as the conversion of the problem to k-CNF or any other standard form, which helps saving time making them more usable in applications such as portfolios. Although our model of the problem hardness assumed that the solver will use the lazy approach in solving the SMT problem, it worked with significant accuracy for both types of solvers. This means that the features suggested by us extract problem inherent properties that represent the hardness of the problem and are solver independent.

7 Future Work

This research is a promising prototype and a proof of concept. We recommend trying it on different theories and to try it on problems made of theory combinations and expect it will perform well.

For the features, we think that incorporating a probing technique (for instance an SMT solver can be run for a fixed period, and statistics such as number of restarts, conflicts, binary propagations, etc...) would significantly improve the preformance of the prediction, escpecially for the unsatisfiable instances. We also recommend further research to find some adaptation

of the Clauses Graph ([9]) and check how this would affect the prediction efficiency. We also recommend to explore more features and check the effect of adding features related to the interaction part such as theory variables- theory functions graphs, and theory variables- clauses graphs. Another approach for dealing with the let bindings can lead to great savings in the features extraction time (Currently we are experimenting with the DAG representation of SMT problems instead of the tree representation which we used).

References

[1] Biere, A., Huele, M., Maaren, V., Walsh, T.: The Handbook Of Satisfiability, chapter 25, Satisfiability Modulo Theory (2009)

[2] Rice, J.R.: The algorithm selection problem. Advances in computers. 15, 65–-118. Academic Press, New York (1976)

[3] Gogiolo, M., Schmidthuber, J.: Learning Dynamic Algorithm Portfolios. Annals of Mathematics and Artificial Intelligence, Volume 47 Issue 3-4, August 2006

[4] Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla Portfolio-based Algorithm Selection for SAT. J. of Artificial Intelligence Research 32, 565-606 (2008)

[5] The portfolio based first order logic theorem prover E-SETHEO, `http://www4.informatik.tu-muenchen.de/schulz/WORK/e-setheo.html`

[6] Gomes, C., Selman, S.: Algorithm Portfolios. Artificial Intelligence Journal, Vol. 126, 43-62 (2001)

[7] Selman, B., Mitchell, D., Levesque, H.J.: Generating hard satisfiability problems. Artificial Intelligence, 81(1-2):17.29 (1996)

[8] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In: Proc. IJCAI-1991, pages 331.337 (1991)

[9] Nudelman, E., Leyton-Brown, K., Hoos, H.H., Devkar, A., Shoham, Y.:Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In: Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings, Vol. 3258, pp. 438-452 (2004)

[10] Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L.: Feature Extraction, Foundations and Applications. Springer. (2006)

[11] Barrett, C., Stump, A., Tinelli, C.: The SMT-LIBv2 Language and Tools: A Tutorial (2010)

[12] The SMTEXEC website, `http://www.smtexec.org/`

[13] Brummayer,R., and Biere, A.: Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: Proc. 15th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09), Lecture Notes in Computer Science (LNCS), vol. 5505, pages 174-177, Springer (2009)

[14] Biere, A.: PicoSAT Essentials. Journal on Satisfiability, Boolean Modeling and Computation 4 75-97 (2008)

[15] Gomes, C., Selman, B., Crato, N., and Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. of Automated Reasoning, 24(1):67–100 (2000)

[16] Hoos, H., and Stutzle, T.: Stochastic Local Search—Foundations and Applications. Morgan Kaufmann (2004)

[17] Hoos, H.: SAT-encodings, search space structure, and local search performance. In: Proc. IJCAI-99, pages 296–302. Morgan Kaufmann (1999)

[18] Phokion Kolaitis: Constraint satisfaction, databases and logic. In: Proc. IJCAI-2003, pages 1587–1595 (2003)

[19] Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L.: Determining computational complexity from characteristic 'phase transitions'. Nature, 400:133–137 (1999)

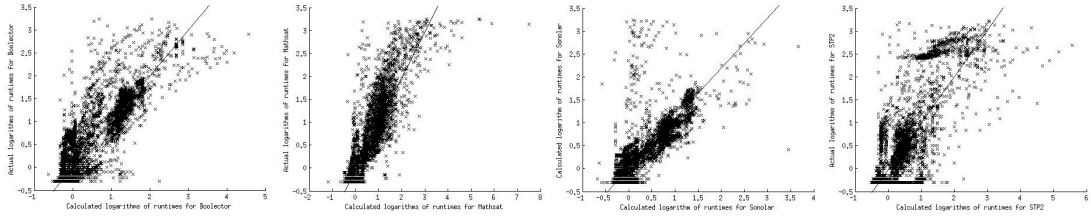

Figure 4: The results for a single model for both satisfiable and unsatisfiable problems show noisy correlation between actual and predicted logarithms of runtimes. The shown figures from left to right are for Boolector, MATHSAT, SONOLAR and STP2.

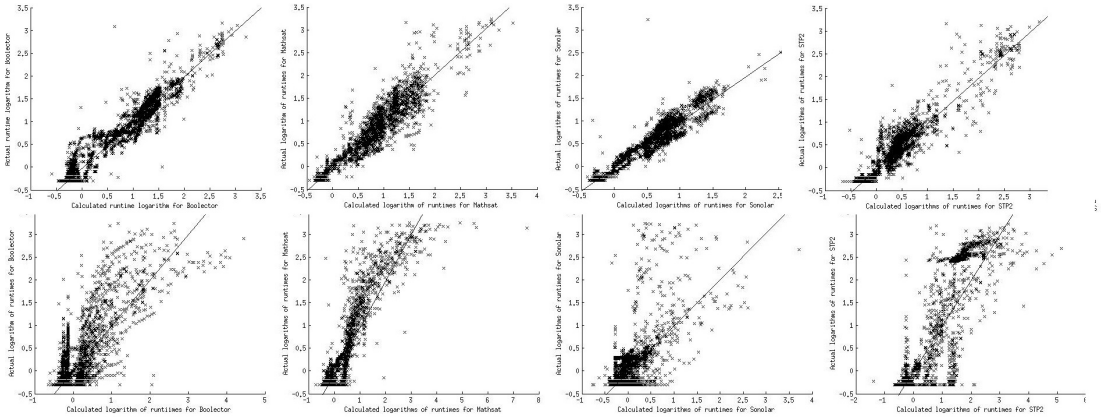

Figure 5: The results for a separate model for satisifiable and unsatisfiable problems. The upper four graphs are for the models of the satisfiable instances, and the lower are for the unsatisfiable instances. The shown figures from left to right are for Boolector, MATHSAT, SONOLAR and STP2. The models for the satisfiable models show better correlation results than the unsatisfiable models.

[20] Williams, R., Gomes, C., and Selman, B.: Backdoors to typical case complexity. In: Proc. IJCAI-2003, pages 1173–1178 (2003)

[21] Bruttomesso, R. ,Cimatti, A., Franz, A.,Griggio , A., Hanna, Z., Nadel, A., Palti, A. , and Sebastiani, R.: A Lazy and Layered SMT(BV) Solver for Hard Industrial Verification Problems. In proc. International Conference on Computer-Aided Verification, CAV 2007 (2007)

[22] Ganesh, V., and Dill, D.: A Decision Procedure for Bit-Vectors and Arrays. In proc. International Conference on Computer-Aided Verification, CAV 2007 (2007)