MiniSAT with Classification-based Preprocessing

Wenxiang Chen and Adele Howe and Darrell Whitley
Computer Science Dept., Colorado State University
Fort Collins, CO, USA

Abstract—We present a classification-based approach to selecting preprocessors of CNF formulas for the complete solver MiniSAT. Three different preprocessors are considered prior to running MiniSAT. To obtain training data for classification, each preprocessor is run, followed by running MiniSAT on its resulting CNF, on instances from the last competition. On each instance, the preprocessor leading to the smallest solving time is considered the “best” and used as the classification for the instance. A decision tree is trained to select the best preprocessor according to features computed from the CNF formulas. The decision tree can therefore be used to predict the best preprocessor for new problem instances.

I. INTRODUCTION

Preprocessing techniques simplify a CNF formula in order to speed up SAT solving. Preprocessing techniques are particularly useful for solving real-world SAT instances due to redundant information introduced during the conversion to CNF formula [1], producing a more compact form of the real-world problem, which is less domain-specific [2].

As has been observed by others (e.g., [3]), the value of specific preprocessing techniques is highly variable and depends on its parameterization, the problem instances and the core solver. In a pilot study, we examined the impact of specific CNF modifications on the performance of MiniSAT on application instances from the SAT 2013 competition. We found that different preprocessors (e.g., versions of SatELite) can produce very different CNF formulas. Additionally, we found that some simple truth preserving modifications (e.g., re-arranging the clauses or flipping the polarity of the literals) can significantly impact performance. To leverage the diverse biases of various preprocessors, we apply machine learning techniques to automatically select the preprocessor with the best predicted performance on a per-instance basis.

Three different types of preprocessing methods are used in our solver: 1) principaled, 2) bias adjusting and 3) speculative. Principaled encompasses those that employ well founded, truth preserving techniques for simplifying the formula, e.g., SatELite [2] and Coprocessor [4]. Bias adjusting refers to syntactic modifications to the formula that preserve truth and do not themselves reduce the number of variables or clauses, e.g., re-ordering clauses, grouping unit clauses. Speculative includes techniques that use heuristics to set variable values, remove those variables and the clauses satisfied by them and then use the reduced formula as the input to a core solver. The motivation for speculative pre-processing is that if the variables have been set correctly then the reduced search space should support solving instances more efficiently.

II. MAIN TECHNIQUES

Our solver consists of a set of preprocessors and a single solver. Given an instance, the solver computes its features, selects a preprocessor based on the features, runs the preprocessor, and provides the modified formula to MiniSAT as input. When the selected preprocessor is speculative, additional steps may be added if the heuristically set values lead to the formula being recognized as unsatisfiable; in this case, MiniSAT will be run without other pre-processing for the remaining time.

A. Constituent Preprocessors

Our solver employs three preprocessors (in addition to that built in to MiniSAT), one of each type:

a) Principled: We include the well known SatELite preprocessor. Although MiniSAT-2.2-Simp integrates SatELite, we still found that running SatELite before MiniSAT can be beneficial on some instances, i.e., this strategy solves some instances that MiniSAT by itself cannot solve in the allotted time.

b) Bias Adjusting: We considered a variety of syntactic modifications. The one that appears to best complement the other preprocessors is to invert the signs of all literals, run MiniSAT on the inverted instances and finally invert the returned assignment if a model is found on the inverted instances. MiniSAT has a polarity vector that is initialized to all false, which introduces a bias of setting new branching variables to false. Inverting the instance reverses the impact of this bias, which appears to be helpful for instances in which a majority of variables in the solutions are false.

c) Speculative: We have investigated techniques for heuristically setting variable values based on a Walsh analysis of the formula which we call "Hyperplane Reduction" [5]. We use a simplified form of this for our speculative preprocessor. The ten most frequently occurring variables in an instance are first fixed to false while running MiniSAT for at most 2600 seconds. If MiniSAT return unsatisfiable on the subspace or MiniSAT runs out of time, MiniSAT is rerun with the ten most frequent variables fixed to true. Finally, if both of the speculative assignments are proven to be unsatisfiable, MiniSAT is rerun on the search space without any assumption on assignment.

d) No Additional Preprocessor: We also include the option of not running any of the preprocessors just described; in this case, MiniSAT is run solo.
B. Classifying Problem Instances

We formulate the preprocessor selection as a classification task. In our solver, 43 base features \[6\] which are used in SATZilla \[7\] are computed on each instance. The training dataset contains 150 instances from the Core Solvers, Sequential, Application SAT track of SAT competition 2013. The Weka implementation of the C4.5 decision tree with default parameters is used for classification. In the offline training process, each of the preprocessors coupled with MiniSAT-2.2-Sim[\textsuperscript{2}] are evaluated on the training set to obtain runtimes on each training instance. The name of preprocessor resulting in the smallest solving time is used as the class label. The training process generates a decision tree, which is further converted into a script for picking the predicted best preprocessor based on features computed on new instances. Once a preprocessor is selected, it is coupled with MiniSAT to run on the new instance until it returns an answer or times out.

There are two main advantages of using a decision tree as the classifier. First, the classification with decision tree simply goes through a number of binary condition checks on computed features and this can be done quickly with little overhead in runtime. Second, one can manually inspect the built decision tree to gain an understanding of relationship between instance features and performances of preprocessors.

III. MAIN PARAMETERS

We use unmodified MiniSAT as a complete solver along with its default parameters. The parameter to our preprocessing is the number of bits to fix in speculative preprocessor. We use 10 bits for this number because it gave us the most consistent results in our empirical tests. It is possible that the number of bits is instance dependent, but we leave this determination to future work.

IV. IMPLEMENTATION DETAILS

The code to find the ten most frequent variables and to generate the instance with these variables fixed to some predefined truth values was implemented in C. GCC 4.8.2 is used to test our code using the “O3” compiler flag to create an optimized 64-bit binary. We used the J48 implementation from Weka 3.6.10 with default parameters of the C4.5 decision tree to perform classification. We wrote a Python script to convert the output decision tree from Weka to a Python library which can be imported to accomplish preprocessor selection.

The decision tree submitted to the competition was trained on data collected from HP-xw6600-Xeon5450-SAS machines with 8-core CPU at 3.0GHz and 16GB main memory. The configuration is close to what we expect for the competition and so the decision tree should approximate predicted performance.

V. SAT COMPETITION 2014 SPECIFICS

We have submitted our solver to the Sequential, Application SAT track and the Sequential, Hard-combinatorial SAT track of the competition. Although MiniSAT can determine if a speculative preprocessed instance is unsatisfiable, it does not necessarily mean that the original problem is unsatisfiable. Therefore we submitted our entry only to the SAT tracks.

We chose the Application and Hard-combinatorial tracks because we conjecture that the structure of industrial(-like) instances are more likely to have solutions that has a majority of variables set to true (or false).

VI. AVAILABILITY

Our C code and scripts can be downloaded from the following link:

http://www.cs.colostate.edu/~chenwx/doc/mcp.zip

ACKNOWLEDGMENT

This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-11-1-0088. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

REFERENCES


---

\[1\] Weka software webpage: http://www.cs.waikato.ac.nz/ml/weka/

\[2\] Source code is available at http://minisat.se/downloads/minisat-2.2.0.tar.gz