A MAX-SAT Algorithm Portfolio[1]

Paulo Matos and **Jordi Planes** and **Florian Letombe** and **João Marques-Silva**[2]

Abstract.

The results of the last MaxSAT Evaluations suggest there is no universal best algorithm for solving MaxSAT, as the fastest solver often depends on the type of instance. Having an oracle able to predict the most suitable MaxSAT solver for a given instance would result in the most robust solver. Inspired by the success of SATzilla for SAT, this paper describes the first approach for a portfolio of algorithms for MaxSAT. Compared to existing solvers, the resulting portfolio can achieve significant performance improvements on a representative set of instances.

1 Introduction

In recent years, one of the optimization counterparts of the Boolean satisfiability problem (SAT) has attracted the interest of researchers: the maximum satisfiability (MaxSAT) problem. MaxSAT and its variations find a number of relevant applications, including scheduling and design automation [12, 13].

This work is the first attempt to implement and evaluate an algorithm portfolio solving MaxSAT problems. The portfolio computes several features of an instance and estimates the runtime for each solver in the portfolio. Then, it solves the instance with the estimated fastest solver. An extended number of instances have been considered, indicating that the portfolio is able to solve more instances, from the selected set of instances, than any other solver. Besides, the total run time to solve is lower for the portfolio, despite the time spent in the feature computation.

The paper is organized as follows, Section 2 gives the notions for MaxSAT solving; Section 3 introduces the portfolio learning process; and Section 4 explains the steps to execute and test the portfolio, and discusses the experimental results. The paper concludes in Section 5.

2 Preliminaries

This section provides a brief introduction to the MaxSAT problem solving. Familiarity with SAT and related topics is assumed [1].

The MaxSAT problem consists of finding an assignment which satisfies the maximum number of clauses in a CNF formula. MaxSAT algorithms have been the subject of significant improvements over the last decade (e.g., see [7, 5] for a review of past work). Despite the clear relation with the SAT problem, most modern SAT techniques cannot be applied directly to the MaxSAT problem (e.g. unit propagation or clause learning). As a result, the most successful MaxSAT algorithms, in the most recent MaxSAT Evaluations, implement branch and bound search, and integrate sophisticated lower bounding and inference techniques. However, past MaxSAT Evaluations did not consider complex problem instances from practical applications. As a result, we have also considered for the portfolio a set of practical problem instances and a recent solver focused on such instances, msu [10].

We have focused on the experience of an existent efficient portfolio, SATzilla [14], an algorithm portfolio for SAT, which has demonstrated to be a robust solver and very competitive in the SAT Competitions[3]. Before SATzilla, Gomes and Selman [3] worked with stochastic search portfolios on several $\mathcal{NP}$-Complete problems. There exists also other preliminary works on algorithm portfolios that deal with problems similar to MaxSAT [8, 6, 4].

3 Model Generation

The capacity to predict the time that a solver will spend on a given instance is one of the key aspects in the design of an algorithm portfolio. The prediction is done using a model created by a learning process over a set of instances. Once the model is created, the portfolio computes the features for a given instance and, based on the model, decides which solver to run.

Our models are linear functions $\sum_{i>0} \beta_i x_i + \beta_0$, which compute the approximate runtime of a solver on a particular instance i. For the linear function, x_i is the value for the feature i of the instance and β_i are the coefficients to be found for each feature by the model generator. After several steps of forward selection and basis function expansion, in order to fit supra-linear data, we perform ridge regression [9] to obtain the unknowns β_i. Forward selection is performed to reduce the number of interesting features. Basis function expansion of the feature set, on the other hand, allows a linear model as the one we used to model supra-linear data (which allowed us to generate the quadratic model presented in section 4). Data preprocessing also handles cases where a solver timed-out on a specific instance by removing it from the training set. The process of generating the model is executed for every solver in the portfolio. After each model is computed, it is tested over a test set.

Our model generator was tested for correctness by generating random data and finding a model for it. If the data can be fit using our model, the model output should be the same as the model used to generate the random data.

The selected solvers to be used are of three different kinds for the sake of complementarity: a Pseudo-Boolean Optimization solver, minisat+ [2]; a recent solver that efficiently deals with real problem instances, msu [10]; and the strongest solver in the MaxSAT category in the MaxSAT Evaluation 2007, maxsatz [7]. The solver maxsatz implements a branch and bound search and integrates sophisticated lower bounds and inference techniques. On the other hand, algo-

[1] This work is partially supported by EPSRC grant EP/E012973/1, and by EU grants IST/033709 and ICT/217069.

[2] School of Electronics & Computer Science, University of Southampton, UK, email: {pocm,jp3,fl,jpms}@ecs.soton.ac.uk

[3] `http://www.satcompetition.org/`

rithm msu is a process that iteratively solves several SAT problem instances, until it reaches the MaxSAT solution.

Three kind of features have been considered [11]: problem size features, balance features and local search probe features. The most important features (among the first selected by forward selection) are in the set of local search probes.

4 Experimental Results

The experimentation has been performed in a Linux Intel Xeon 3.0 GHz. A timeout of 1000 seconds was used for all MaxSAT solvers considered. The memory limit was set to 3GB. Some of the sets of instances considered are from the MaxSAT Evaluation 2007, the ones considered hard to solve and close to real problems; and instances from real problems: circuit design and planning. There are 586 instances from the following sets:

msu3.1	minisat+	maxsatz	pfquad	pflin	oracle
507	211	135	524	548	582

Table 1. Total number of solved instances for each solver

RAMSEY, SPINGLASS, MAXCUT from the MaxSAT Evaluation 2007;
DEBUG, IBM, UCLID, PIMAG from circuit design;
SATPLAN from planning problems converted to SAT instances.

In order to check our portfolio, we have created the *oracle*, a virtual portfolio which always selects the best possible result. The entries *pflin* and *pfquad* correspond to our portolios using a linear model of the features and a quadratic model of the features respectively. A preprocessing time per instance has been added to its total time. In Table 1, we can notice the portfolio is the most robust MaxSAT solver, since it solves the largest number of instances.

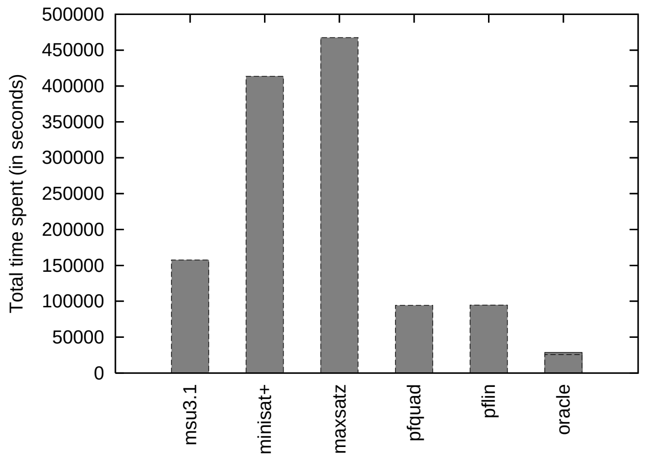

Figure 1. Total spent time in seconds for each solver in MaxSAT

Figure 1 shows the total time taken by each of the solvers in the portfolio, our two portfolio models and the oracle. The results obtained by our models are close to the oracle, and spend less time than the rest of solvers. We are aware, however, that this can still be improved. As mentioned earlier, our learning method does not handle solver timeouts, which means that our portfolio is biased regarding solvers which timeout often and solve a few instances in short time. Still, having a portfolio capable of achieving these initial results motivates additional research in algorithm portfolios for MaxSAT.

5 Conclusions

This paper presents a method to develop an algorithm portfolio for the MaxSAT problem. Given that no benchmark repository exists for MaxSAT, problem instances from real world problems and from the MaxSAT evaluation have been used. To the best of our knowledge, this is the first algorithm portfolio for MaxSAT problem.

From the experimental results we conclude that our MaxSAT algorithm portfolio is the most robust solver among the MaxSAT problem instances we have considered. Future research work includes adapting the model generator to handle timeouts, and also adapting the solver portfolio solver to deal with Partial MaxSAT and Weighted MaxSAT. Additional research on identifying suitable features will be required for further improving the model used.

REFERENCES

[1] Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang, 'Propositional satisfiability and constraint programming: A comparative survey', *ACM Computing Surveys*, **38**(4), (2006). Electronic Edition, 54 pages.
[2] Niklas Een and Niklas Sörensson, 'Translating pseudo-boolean constraints into SAT', *Journal on Satisfiability, Boolean Modeling and Computation*, **2**, 1–26, (2006).
[3] Carla P. Gomes and Bart Selman, 'Algorithm portfolios', *Artificial Intelligence*, **126**(1-2), 43–62, (2001).
[4] Kevin Keyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham, 'A portfolio approach to algorithm selection', in *International Joint Conference on Artificial Intelligence - IJCAI'03*, pp. 1542–1543, (2003).
[5] Javier Larrosa, Federico Heras, and Simon de Givry, 'A logical approach to efficient max-SAT solving', *Artificial Intelligence*, **172**(2–3), 204–233, (2008).
[6] Kevin Leyton-Brown, Eugene Nudelman, and Yoah Shoham, 'Learning the empirical hardness of optimization problems', in *Principles and Practice of Constraint Programming CP'02*, volume 2470 of *LNCS*, pp. 556–572, (2002).
[7] Chu Min Li, Felip Manya, and Jordi Planes, 'New inference rules for max-SAT', *Journal of Artificial Intelligence Research*, **30**, 321–359, (2007).
[8] Lionel Lobjois and Michel Lemaître, 'Branch and bound algorithm selection by performance prediction', in *National Conference on Artificial Intelligence - AAAI'98*, pp. 353–358, (1998).
[9] Donald W. Marquardt and Ronald D. Snee, 'Ridge regression in practice', *The American Statistician*, **29**(1), 3–20, (1975).
[10] Joao Marques-Silva and Jordi Planes, 'Algorithms for maximum satisfiability using unsatisfiable cores', in *Design, Automation and Test in Europe - DATE'08*, (2008).
[11] Eugene Nudelman, Alex Devkar, Yoav Shoham, and Kevin Leyton-Brown, 'Understanding random SAT: Beyond the clauses-to-variables ratio', in *Principles and Practice of Constraint Programming CP'04*, volume 3258 of *LNCS*, pp. 438–452, (2004).
[12] Sean Safarpour, Hratch Mangassarian, Andreas Veneris, Mark H. Liffiton, and Karem A. Sakallah, 'Improved design debugging using maximum satisfiability', in *Formal Methods in Computer Aided Design - FMCAD'07*, pp. 13–19, (2007).
[13] Hui Xu, R. A. Rutenbar, and Karem A. Sakallah, 'sub-SAT: a formulation for relaxed boolean satisfiability with applications in routing', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **22**(6), 814–820, (2003).
[14] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown, 'SATzilla-07: The design and analysis of an algorithm portfolio for SAT', in *Principles and Practice of Constraint Programming CP'07*, volume 4741 of *LNCS*, pp. 712–727, (2007).