ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-1065

Timeout-Sensitive Portfolio Approach to Enumerating Minimal Correction Subsets for Satisfiability Problems

Yuri Malitsky[1] and **Barry O'Sullivan**[2] and **Alessandro Previti**[3] and **Joao Marques-Silva**[4]

1 INTRODUCTION

Constraint satisfaction and boolean satisfiability (SAT) are common approaches to modeling and solving combinatorial problems. It is often the case that a problem does not admit a solution; we refer to such problems as being inconsistent or over-constrained. In these cases we may be interested in finding the minimal subset of constraints, or clauses in SAT, that if removed from the problem would allow a solution to be found. This minimal number of constraints is referred to as a Minimal Correction Subset (MCS). More formally, an MCS can be defined as follows:

Definition 1 (Minimal Correction Subset) *Given an unsatisfiable boolean formula $\mathcal{F}$, $\mathcal{C} \subseteq \mathcal{F}$ is a* Minimal Correction Subset *(MCS) iff $\mathcal{F} \setminus \mathcal{C}$ is satisfiable and $\forall c \in \mathcal{C}, \mathcal{F} \setminus (\mathcal{C} \setminus \{c\})$ is unsatisfiable.*

The identification of MCSs has many practical applications, including software debugging, hardware debugging, and product configuration, among many others. The analysis of over-constrained problems dates back to Reiter's seminal work in diagnosis [7]. Motivated by this work, many algorithms have been developed to enumerate some or all MCSs contained in an inconsistent set of constraints. Examples of MCS extraction algorithms include (linear) search for a maximally satisfied set of clauses [1], adapting the QuickXplain algorithm [6, 2], and the iterative removal of satisfied clauses, i.e. the CLD algorithm [5].

In previous work we proposed a portfolio approach for the enumeration of MCSs [4] based on a continuous switching amongst solvers with a fixed timeout. Results confirmed that this approach is better than running a single solver. However, in [4], the switching timeout was selected arbitrarily and remained fixed. In this paper we investigate the advantages of choosing different switching timeouts, and we demonstrate how performance can be further improved.

2 PREDICTING RUNTIME

As any enumerating solver progresses the rate at which new solutions are found can vary significantly. Some solvers will exploit local solutions, returning a high number of results early on in the search. Alternatively, a solver might instead employ an exploration strategy that spends its initial time fine tuning its parameters. However, regardless of the search strategies of the solvers in a portfolio, our objective is to predict this behavior automatically.

The proposed portfolio approach, which we study in the context of SAT, will start by computing the structural features of the new instance it is to solve. Based on these features it will make a decision of which solver from its portfolio to use and with what timeout. Once the assigned solver finishes, all of the newly found solutions will be recorded, and blocking clauses added to the original instance in order to prevent future recomputation of the same MCSs. The addition of these new constraints changes the underlying structure of the problem so the features need to be recomputed. Then, using the new features, the portfolio once again makes a decision about which solver to use and its duration. The process is repeated until the overall timeout, one hour in our case, is reached.

As a first step for this methodology, an extended dataset is generated that includes the instances we are likely to observe once the portfolio has made some decisions. This is because the structure of instances is likely to be considerably different once blocking clauses are added. Therefore for each of our training instances we gather all of the MCSs found by any solver after one hour and for each original instance create three extended versions. These extended versions respectively have $\sim 25\%$, $\sim 50\%$ and $\sim 75\%$ of all the MCSs added as conflict constraints. We state approximate percentages because, in the hope of providing better coverage of possible instances, each extended instance has $\pm 5\%$ chosen uniformly at random.

For each instance in the extended dataset, each of the solvers in our portfolio is evaluated for one hour, with the number of MCSs found every two minutes being recorded. Using this information it is possible to see the average rate at which new MCSs are found for each solver. What we observe is that, in most cases, the rate of improvement starts off very high for all of our solvers and then trails off to a small constant improvement after 20 minutes. This observation helps explain the effectiveness of the previous portfolio approach that uses a constant timeout of 10 minutes. Yet because the majority of MCSs are found in the first few minutes, a portfolio that exploits this is likely to perform even better.

During the test phase, the proposed portfolio approach first computes the features of the new instances and uses this information to find a small subset of nearest neighbors. In our case we use 20 as a statistically large number. These neighbors are used to compute the average rate of improvement for each solver during a single hour. It is important to note that this rate of improvement is recorded as the percentage of the total number of MCSs found by any solver. For each solver, we fit a 4th degree polynomial function to match the observed improvement rate on neighboring instances. Here, we use a 4th degree polynomial because we want to be able to take into account a case where the rate of improvement has multiple maxima and

[1] Insight Centre of Data Analytics, University College Cork, Cork, Ireland, email: y.malitsky@4c.ucc.ie
[2] Insight Centre of Data Analytics, University College Cork, Cork, Ireland, email: barry.osullivan@insight-centre.org
[3] Complex and Adaptive Systems Laboratory, Dublin, Ireland, email: alessandro.previti@ucdconnect.ie
[4] Complex and Adaptive Systems Laboratory, Dublin, Ireland, email: jpms@ucd.ie

Table 1: Comparison of the 5 MCS enumeration solvers over 76 test instances. The table presents the total number of MCSs found over the test set in millions (M), and the average number of MCSs per instance in thousands (k).

	bfd	**bls**	**cld**	**efd**	**els**	**VBS**	**slv-predict**	**slv-time-predict**
Average (K)	138.34	217.42	213.36	148.9	210.31	267.36	326.67	**426.29**
Sum (M)	8.44	13.26	13.01	9.08	12.83	16.31	19.93	**26.0**

minima, but also not to overfit a dataset of 30 measurements (one for every two minutes). The location of the maximum is approximated for each solver, and the predicted winner is evaluated for the appropriate amount of time. The process is then repeated until the overall timeout is reached.

3 COMPUTATIONAL EVALUATION

For the experiments we used five MCS solvers contained in the MCSls tool [5]. The bls (Basic Linear Search) algorithm computes MCSs by testing one clause at a time. The bfd (Basic FastDiag) is based on a divide-and-conquer strategy. The cld (Clause D) algorithm is based on an iterative removal of the satisfied clauses from the over-approximation of the MCS under construction. The other two algorithms, els (Enhanced Linear Search) and efd (Enhanced FastDiag) respectively represent the bls and bfd algorithms extended with the new techniques introduced in [5]. One of the primary reasons for this choice of solvers was that they have been previously proven to be among the best performers. Moreover, they are the only ones able to deal with Partial MaxSAT instances, which is crucial to handle instances modified with blocking clauses. In particular, each new MCS is blocked by adding a hard clause to the original formula, whose clauses are soft. All experiments were run on a linux cluster, where each node is a dual six-core E5-2620 2GHz with 128GB. The memory limit was set to 4GB, and the overall timeout to one hour.

Following the experiments outlined in [4], this paper relied on the same dataset of instances which have been developed over the last two decades expressly for the enumeration task. Almost half of the instances came from DaimlerChrysler's Mercedes benchmarks [8], an extensively studied benchmark suite [3] from an automotive product configuration problem. The other instances are an assortment of small-sized industrial instances, like the FVP-UNSAT/SSCALAR and UCLID hardware verification instances, CMDADD and DIMACS set of benchmarks.

However, when working with this original dataset, we found that a significant proportion of the instances have become trivial for the modern day solvers, with all MCSs found in under 15 minutes. To create a more challenging dataset, we reduced the original dataset to 93 instances that could not be solved in under 20 minutes. We then mined the MaxSAT evaluation instances[5]. From this dataset we collected instances where it was possible to find a significant portion of MCSs within one hour but not so trivial as to be solvable within 20 minutes. The final dataset consisted of 249 instances, 76 of which were randomly chosen for testing and the rest for training.

Table 1 presents the results of the portfolio. The table first presents the average number of MCSs found by each of the solvers if they are allowed to run uninterrupted for one hour. These results are in thousands (K). What we observe is that, by itself, cld is the best performing solver, able to on average find 30 thousand more MCSs than its next competitor. However, we also see that an optimal portfolio (VBS), that knows beforehand which is the best solver, can improve this amount by an additional 50 thousand MCSs. The presented VBS approach, however, runs each of the solvers uninterrupted, and we clearly see that the approach presented in [4], slv-predict, finds over 100 thousand more MCSs per instance. Finally, Table 1 shows that the approach proposed in this paper, slv-time-predict, dominates all other approaches.

[5] http://www.satcompetition.org
http://www.satlib.org
http://maxsat.ia.udl.cat

4 CONCLUSION

Enumerating all Minimal Conflict Subsets (MCS) is an important problem in many practical applications. Previously, it was shown that when creating a solver to enumerate all MCSs it is not desirable to run a single solver uninterrupted for the full allotted timeout [4]. Instead, a portfolio should be trained that selects a solver to run for a short period of time, with newly found solutions added as blocking clauses before a new solver is chosen to run for the next brief period of time. Such an iterative portfolio approach was shown to significantly improve performance.

This paper extended this line of research by proposing a methodology that automatically learns the behavior of the solvers in its portfolio and is thus able to not only choose the next best solver, but also the amount of time that solver should be run. The observed results revealed that such a portfolio can further dramatically improve the state-of-the-art in MCS enumeration solvers.

ACKNOWLEDGEMENTS. This publication has emanated from research supported in part by a grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289, SFI PI grant BEACON (09/IN.1/I2618) and from EU FP7-FET Grant 284715 (ICON).

REFERENCES

[1] James Bailey and Peter J. Stuckey, 'Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization', in *Practical Aspects of Declarative Languages*, volume 3350, 174–186, Springer, (2005).

[2] Alexander Felfernig, Monika Schubert, and Christoph Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **26**, 53–62, (February 2012).

[3] Mark H. Liffiton and Karem A. Sakallah, 'Algorithms for computing minimal unsatisfiable subsets of constraints', *J. Autom. Reason.*, **40**(1), 1–33, (January 2008).

[4] Yuri Malitsky, Barry O'Sullivan, Alessandro Previti, and Joao Marques-Silva, 'A portfolio approach to enumerating minimal correction subsets for satisfiability problems', in *Integration of AI and OR Techniques in Constraint Programming*, volume 8451, 368–376, Springer, (2014).

[5] Joao Marques-Silva, Federico Heras, Mikolas Janota, Alessandro Previti, and Anton Belov, 'On computing minimal correction subsets', in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI, pp. 615–622. AAAI Press, (2013).

[6] Barry O'Callaghan, Barry O'Sullivan, and EugeneC. Freuder, 'Generating corrective explanations for interactive constraint satisfaction', in *Principles and Practice of Constraint Programming*, volume 3709, 445–459, Springer, (2005).

[7] Raymond Reiter, 'A theory of diagnosis from first principles', *Artif. Intell.*, **32**(1), 57–95, (April 1987).

[8] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin, 'Formal methods for the validation of automotive product configuration data', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 75–97, (2003).