Boosting the Performance of SLS and CDCL Solvers by Preprocessor Tuning

Adrian Balint[1] and Norbert Manthey[2]

[1] Institute of Theoretical Computer Science, Universität Ulm
[2] Institute of Artificial Intelligence, Technische Universität Dresden, Germany

Abstract. Preprocessing techniques are crucial for SAT solvers when it comes to reaching state-of-the-art performance as it was shown by the results of the last SAT Competitions. The usefulness of a preprocessing technique depends highly on its own parameters, on the instances on which it is applied and on the used solver. In this paper we first give an extended analysis of the performance gain reached by using different preprocessing techniques individually in combination with CDCL solvers on application instances and SLS solvers on crafted instances. Further, we provide an analysis of combinations of preprocessing techniques by means of automated algorithm configuration, where we search for optimal preprocessor configurations for different scenarios. Our results show that the performance of CDCL and especially of SLS solvers can be further improved when using appropriate preprocessor configurations. The solvers augmented with the best found preprocessing configurations outperform the original solvers on the instances from the SAT Challenge 2012, achieving new state-of-the-art results.

1 Introduction

The propositional satisfiability problem (SAT) is one of the most studied $\mathcal{NP}$-complete problems in computer science. One reason is the wide range of SAT's practical applications ranging from hardware verification to planning and scheduling [1,2,3]. Given a propositional formula in conjunctive normal form (CNF), the SAT-problem consists in finding an assignment for the variables such that all clauses are satisfied.

The area of pragmatic SAT Solving is dominated by two types of solving techniques: Conflict Driven Clause Learning (CDCL) solvers and Stochastic Local Search (SLS). Each technique has its strength on different types of problems. While CDCL solvers are best suited for structured problems and unsatisfiable crafted problems, SLS solver exhibit their strength on random and satisfiable crafted problems.

Besides solving techniques, preprocessing techniques (PPT) (meanwhile also used during search and known as *inprocessing*) have turned out to be crucial for SAT Solvers and enabled them to further increase their performance. PPTs can be seen as transformation rules, that take a formula as input and output a transformed problem, that is satisfiability equivalent to the original one.

Since the introduction of the SATELITE preprocessor [4], which is still one of the most used preprocessor today and is prepended in several state-of-the-art SAT solvers, many new techniques for preprocessing have been proposed (e.g. [5,6,7,8,9]) and implemented. Although most of these new PPTs have found their way into SAT solvers, until the introduction of the preprocessor COPROCESSOR [10] there was no PPT framework like SATELITE which provided them as a stand-alone tool.

Most of the new PPTs are developed only for CDCL solvers or at least with CDCL solvers in mind. A reason why the parameters of these techniques are predetermined to work well in collaboration with CDCL solvers, which raises up the question whether these techniques (maybe with different parameters) could also be useful to SLS solvers. Some PPTs have been analyzed in combination with SLS solvers [11], but did not yield an improvement. Though there is evidence that some PPTs can help SLS solvers on crafted problems: the SLS solver SATTIME2012 [12] (which uses failed literal probing and unit propagation as preprocessing) showed remarkable performance on the crafted problems during the latest SAT Competitions.

In this paper we are interested in analyzing the utility of PPTs for CDCL and SLS solvers separately. A PPT P is considered to be useful (or utile) for a solver S on a set of instances I if the performance of S on I denoted by $perf(S(I))$ can be improved by first executing the preprocessor P and then running the solver S on the simplified problems $P(I)$, i.e. if $perf(S(P(I)) > perf(S(I))$, where $perf$ is the statistical measure of interest (e.g. the number of solved instances). The same run time limitations are imposed to preprocessing time and solving time. Note that the usefulness of a PPT highly depends on the solver and on the instances. We are not interested in the size of the reduction nor in the structural changes of PPTs, but in the speedup preprocessing entails for the SAT Solver. Therefore, we address the following questions related to the utility of PPTs.

1. *How useful is each PPT on its own?*
2. *Which combination and parametrization of PPTs yields the best improvement?*
3. *How far can the best PPT be improved with appropriate parametrization?*
4. *How sensitive is the performance gain when exchanging the solvers?*

To answer the above questions, we first implemented most of the currently available formula simplification techniques into the preprocessor COPROCESSOR. The new version of COPROCESSOR(CP3)[3]extends some of the functionalities by incorporating *covered clause elimination* techniques [9]. CP3 as a PPT framework in combination with the state-of-the-art SAT solvers GLUCOSE 2.1 (CDCL) and SPARROW (SLS) represents the experimental basis for our analysis. For the evaluation we use the application and crafted instances from the SAT Challenge 2012 (SC12)[4]. Question 1 is answered by evaluating each PPTs individually in combination with the two solvers. To answer question 2 and 3, we make use of an automated algorithm configuration tool, which searches for optimal combinations and parameterization of PPTs. The results obtained are validated with other solvers, answering question 4.

Although there is also a reasonable amount of work on *inprocessing* [13,14], which is performing formula simplifications during search, we do not focus on this kind of application of PPTs here, for two reasons: (i) SLS solvers do not learn clauses, and therefore triggering inprocessing rules during search is less powerful, and (ii) GLUCOSE 2.1 does not ship with a framework for performing inprocessing.

1.1 Summary of Contributions

To the best of the authors knowledge, this is the first study on the utility of individual PPTs, and the first attempt to automatically configure these for different types of SAT solvers on heterogeneous sets of SAT instances.

[3] COPROCESSOR is available at `http://tools.computational-logic.org`.
[4] http://baldur.iti.kit.edu/SAT-Challenge-2012/index.html

The results of our analysis show that the performance of SLS solvers can be drastically increased when using the appropriate PPT parameterization. The performance of the SLS solver SPARROW could be improved by $25\,\%$ on hard combinatorial instances from the SC12, making it the best performing single engine solver on this class of instances. We could also show that good configuration of PPTs for SLS solvers strongly differ from those commonly used for CDCL solvers, though the best performing single techniques is the same, namely (bounded) variable elimination. A deeper analysis of this PPT shows that for SLS solver it is beneficial to allow a growth of the formula by up to 1000 clauses, while each elimination step is allowed to introduce ten more clauses than clauses being eliminated. To our surprise, not the least frequent variable is eliminated first, but the most often occurring variable.

We show that for CDCL solvers on application instances, among the published PPTs, (bounded) variable elimination (BVE) [4] is the most powerful. However, we are also able to show that for special instance classes, techniques as equivalent literal elimination [15], or bounded variable addition (BVA) [6] are also important, having a unique contribution. State-of-the-art performance can only be achieved by combining several PPTs, more specifically BVE, BVA and unhiding [8]. We are also able to show that the performance of the award winning solver GLUCOSE 2.1 can be further improved by $3\,\%$ when using an appropriate PPT configuration.

All in all we could show that the performance of solvers can be improved by appropriate parameterization of the preprocessor and that optimal settings are very different for CDCL and SLS solvers. Further bounded variable elimination turns out to be the best PPT (independent of the solver), a result that might motivate to revisit the possible alternatives of this technique.

1.2 Related Work

In general, formula simplification is analyzed when new techniques are presented (e.g. [6,15,4,5,16,9,7]), however, the utility of these techniques has not been compared yet. This has only been done for components of CDCL SAT solvers [17], but without the analysis of complex combinations nor of PPTs. Furthermore, proposed PPTs are usually not well parameterized, although their implementation could offer many degrees of freedom.

Formula simplifications have been combined with SLS solvers already in [18], where redundant binary clauses are added to a formula to help the SLS solver simulate unit propagation. In [12], the SLS solver SATTIME2012 is combined with unit propagation and failed literal detection, to improve its performance on crafted instances. In [11], the effect of covered clause elimination(CCE) [9] techniques with respect to the performance of SLS solvers on crafted instances has been analyzed, showing that this technique family does not improve the performance, which is in line with the results we will present.

To the best of our knowledge, automated configuration of PPTs has not been considered so far – however, the power of automated configuration has been demonstrated on a wide range of optimization scenarios, including CDCL and SLS SAT solvers [19,20]. The parameters of SPARROW itself have been optimized using this techniques [20].

2 Preliminaries

We assume the reader to be familiar with propositional logic and its clausal fragment. In the following we will briefly discuss the notations used throughout this paper and give a brief description of the PPTs we are considering.

Let V be a fixed set of Boolean variables, or briefly just *variables*. A *literal* is a variable x (*positive literal*) or a negated variable $\overline{x}$ (*negative literal*). We overload the overbar notation: The *complement* $\bar{l}$ of a positive (negative, resp.) literal l is the negative (positive, resp.) literal with the same variable as l. A *clause* is a finite set of literals and a *formula* (*in conjunctive normal form*) is a finite set of clauses. A clause that contains exactly a single literal is called a *unit clause*. A clause that contains a literal and its complement is called a *tautology*. The set of all variables occurring in a formula F (in positive or negative literals) is denoted by $\mathsf{Vars}(F)$. Given two clauses C and D with $l \in C$ and $\bar{l} \in D$, we call the clause $E := (C \setminus l) \cup (D \setminus \bar{l})$ the *resolvent* of C and D, that has been created by *resolution*, which is written $E = C \otimes D$.

Since we do not discuss the properties of these techniques, we also do not introduce semantics of propositional logic here. Briefly, an *assignment* that *satisfies* a formula F is called a *model* of F.

3 Modern Preprocessing Techniques

Most modern SAT solvers still rely on standard preprocessor SATELITE published in 2005 [4], which includes the PPTs *Unit Propagation* (also called Boolean Constraint Propagation), *Subsumption*, *Strengthening* (also called *self subsuming resolution*) and (*bounded*) *variable elimination*. Since 2005 many other preprocessing techniques have been published [6,15,4,5,16,9,7]. The implementation details or possible variations (parameterizations) of these PPTs are presented in these publications only briefly. In the following we provide a short description along with possible parameterizations of all PPTs considered in this work. We define F as the input formula to a technique, and F' as the output.

Unit Propagation (**UP**): If there is a unit clause in the formula $C = \{l\} \in F$, then the interpretation mapping this literal to $\top$ is applied to the formula.

Subsumption (**SUB**): If there is a clause $C \in F$ that is a subset of another clause $D \in F$, then D is removed.

Strengthening (**STR**): Let D and E be a disjunction of literals. If there exist two clauses $C_1 = \{l, D\}$ and $C_2 = \{\bar{l}, E, D\}$, then the clause C_2 can be replaced by the resolvent $C_1 \otimes C_2 = \{D, E\}$. Note, that the clause C_2 could produce several resolvents that subsume C_2. However, to the best of the authors knowledge, C_2 is replaced immediately, so that other resolvents may not be produced anymore. We enabled COPROCESSOR to keep C_2 during strengthening, and introduce a parameter *allStrength* that enabled this extension based on the size of C_2.

(Bounded) Variable Elimination (**BVE**) [4]: Let $S_x = \{C \mid C \in F, x \in C\}$ and $S_{\overline{x}} = \{C \mid C \in F, \overline{x} \in C\}$. Let S be the set of non-tautological pairwise resolvents of all clauses in S_x and $S_{\overline{x}}$. Then, $F' = F \setminus (S_x \cup S_{\overline{x}}) \cup S$. As a bound, usually the number of clauses is used: if $|S| < |S_x| + |S_{\overline{x}}|$, the sets are replaced. Since it is very unlikely that $|S| < |S_x| + |S_{\overline{x}}|$, if both $|S_x|$ and $|S_{\overline{x}}|$ are large, there is a cutoff, that does not apply BVE to a variable x, if each of the sets has at least 10 clauses, or if one set has at

least 5 clauses and the other set contains more than 15 clauses. During the computation of BVE, for each clause in S_x and $S_{\overline{x}}$ the number of resolvents is counted, to calculate the bound first. If a clause does not produce any resolvent, this clause is called *blocked*, and can be removed, even if no variable elimination is performed [5]. This technique is called *blocked clause elimination* (BCE), and can be disabled with a parameter *noBce*.

(Bounded) Variable Addition (**BVA**) [6]: Let x be a fresh variable, that is $x \notin \mathsf{Vars}(F)$, and $S \in F$ be a set of clauses, such that there exists two sets S_x and $S_{\overline{x}}$, and $S_x \otimes S_{\overline{x}} = S$. Inversely to BVE, BVA replaces the set of clauses S, if $|S| > |S_x| + |S_{\overline{x}}|$. The current implementation is only able to find simple patterns, where the clauses in S are of the form $\{l, D_i\}, \{l', D_i\}$, and $i > 2$ so that the final set of clauses contains less clauses. The fresh variable will be added within the clause $x \rightarrow l \wedge l'$. If a set S is found, and $i > 4$, another clause $l \wedge l' \rightarrow x$ is added, and furthermore, all clauses C with $\overline{l}, \overline{l'} \in C$ are replaced with $(C \setminus \{\overline{l}, \overline{l'}\}) \cup \{\overline{x}\}$.

Probing (**Probe**) [21,16]: Contrary to the depth first search of CDCL solvers, probing (also called *failed literal detection*) performs a breadth first search to check whether certain literal l assignments lead to a conflict by unit propagation. If a conflict is found, then the clause $\{\overline{l}\}$ can be added to the formula. Furthermore, conflict analysis with the first UIP scheme as in the CDCL algorithm could be applied [22]. Since probing searches on the top level, also all UIPs could be collected and added to the formula. Furthermore, *double look-ahead* can be used [23]. With the immediate implications of l and $\overline{l}$, furthermore *equivalent literals*, and *necessary assignments* can be detected [23].

Covered Clause Elimination (**CCE**) [9]: There exists several techniques that allow the addition of literals to a clause C. CCE in COPROCESSOR allows *hidden literal addition* [7], *asymmetric literal addition* [7] and *covered literal addition* [9], which are described in the literature. During computing hidden literal addition, or asymmetric literal addition, the procedure might also find failed literals. The final clause C' can either be a tautological clause, or C' could be blocked. If any of the reasons match, C is removed from F. Since the computation of C' is expensive, CCE is only applied to clauses with a size larger than 40 % of the maximum clause size.

Hidden Tautology Elimination (**HTE**) [7]: When restricting the computation of the hidden literal addition to binary clauses only, a clause C can still become a tautology. These clauses are removed by HTE.

Equivalent Literal Elimination (**EE**) [15]: A set of literals, that contains equivalent literals can be computed based on the binary implication graph of the binary clauses in the formula [15]. All literals in such a set are replaced with one representative literal of that class.

Unhiding (**Unhide**) [8]: HTE and STR can be approximated based on sampling the binary implication graph of the formula [8]. During sampling the graph, failed literals can be detected. To improve the quality of the approximation, multiple randomized samplings (*iterations*) can be generated. Furthermore, one can choose whether HTE (*UHTE*) or STR(*UHLE*) should be applied based on the sampling.

Ternary Resolution (**3RES**): resolving two ternary (or binary) clauses can result in another ternary (binary) clause, or even in a binary clause. This technique can be applied before search. The newly created clause can even subsume other clauses in the formula. The subsumption can be eagerly enabled via the parameter $eagerSubsume$.

Add Binary Resolvents (**ADD2**) [18]: for SLS solvers it has been reported that adding redundant binary clauses to the formula can speed up the search of this kind of solvers. Thus, redundant binary clauses can be generated by performing probing on a literal l, and then a clause $\{\bar{l}, l'\}$ can be added, if this clause is not already present in the formula, and the literal l' is propagated after assuming l. Since the possible number of those clauses is huge, the number of added clauses is heavily restricted.

Dense : after many simplification techniques, the set of variables in the formula is not continuous any more. By removing the gaps, the total number of variables in the final formula is not decreased, however, the numeric representation is smaller, and therefore the SAT solver has a compact variable representation, which might result in a more cache friendly and thus faster execution.

3.1 Implementation

The standard preprocessor SATELITE ships with a cutoff: if the number of clauses in the formula exceeds six million, the preprocessor simply returns the input formula. However, these large instances could also benefit from simplifications. Therefore, instead of applying a clause limit for the whole preprocessor, CP3 provides limits for each technique, allowing it to perform only a given number of clause dereferences. In this way, the overall simplification time can be controlled, and the simplification can be reproduced on any computing system because it is independent of time.

4 Experimental Setup

To answer the four questions posted in the introduction we set up a series of four experiments which are based on the following two evaluation scenarios. If not indicated otherwise, all statements about performance in this paper always refer to these scenarios.

1. CP3 combined with SPARROW on the satisfiable Hard Combinatorial instances from the SC12 (SHC12) with a run time limit of 900 s.
2. CP3 and GLUCOSE 2.1 without SATELITE on the Application instances from the SC12 (APP12) with a run time limit of 900 s.

Single PPT Analysis. To answer question 1 we have measured the utility of each PPT individually, keeping the parameters of each PPT close to the literature.

Combined PPTs Analysis. To answer question 2 we have parameterized all PPTs, allowing to turn each technique on and off and to set different parameters. The PPTs execution order is fixed for all experiment: UP, 3RES, SUB, STR, EE, Unhide, HTE, Probe, BVE, BVA, CCE, ADD2 and finally DENSE[5]. The parameterized version of CP3 together with the solver (which had fixed parameters) was then optimized with a parallel model based automatic algorithm configurator, which is a parallel version of the SMAC configurator presented in [24]. The configurator is implemented in the EDACC framework [25]. We have performed five configurations experiments with a configuration budget of $2 \cdot 10^6$ s and a cutoff of 450 s per job optimizing the PAR10 statistic, which is the sum of the run time of all solved instances plus the number of unsolved instances penalized with ten times the timeout. Optimizing the PAR10 statistic is almost equivalent to optimizing the number of solved instances. For the optimization of CP3 +SPARROW we

[5] Allowing the configurator to also alter the execution order of PPTs would have resulted in a much larger configuration space.

have randomly selected 150 instances from scenario 1 on which at least one of the PPTs from the analysis of the individual PPT have ever changed something on the instance. For the optimization of CP3 +GLUCOSE 2.1 we have randomly selected 300 instances from scenario 2 without further restrictions.

Extended Single PPT analysis. The results of the first two experiments raised up question 3, which we tried to answer by further extending the parameterization of the best performing single PPT, namely BVE, and optimizing it in a separate experiment with an optimization budget of $4 \cdot 10^6$ s. BVE was extended and parameterized further than proposed in the literature.

Remember BVE and notations S, S_x and $S_{\overline{x}}$. First, we loosen the limit $|S| + b_l < |S_x| + |S_{\overline{x}}|$, where b_l is a local limit that has to be met per step (default configuration $b_l = 0$). With the local limit we allow each elimination to increase the number of clauses in the formula per step. Furthermore, we introduce a global counter $c_g = 0$, which cumulates the difference of clauses in F and F' per step: $c_g := c_g + |S_x| + |S_{\overline{x}}| - |S|$. By specifying a global upper bound b_u, when can reject elimination steps that would surpass this global limit, preventing BVE from introducing more than b_u clauses. For an unlimited increase of the number of clauses, this upper bound can be disabled (b_u is enabled by default). We also allow the elimination to take place when $|S|_l < |S_x|_l + |S_{\overline{x}}|_l$, where $|\cdot|_l$ measures the number of literals (parameter (*red_lits*). Finally, we tried a number of heuristics to control how the variable for the next elimination step is chosen.

Applicability to other Solvers. Question 4 is answered by prepending the best found CP3 configurations from the previous experiments to other CDCL and SLS solvers and evaluate them on our scenarios.

Software and Hardware. All experiments were performed on the bwGrid clusters [26] (Intel Harpertown quad-core CPUs with 2.83 GHz and 8 GByte RAM). The operating system was Scientific Linux. Experiments were conducted with EDACC, a platform that distributes solver execution on clusters [25]. The Sparrow code is an improved version of the code used in [27]. GLUCOSE version 2.1 is available online[6].

5 PPTs Analysis for SLS Solvers

5.1 Single PPT Analysis

Figure 1 presents the result of the evaluation of each individual PPT as a cactus plot. Along the individual PPTs we have also evaluated Sparrow alone and prepended with the SATELITE preprocessor. Most of the PPTs are not able to boost the performance of Sparrow performing similar or even worse than Sparrow alone. BVE is the best performing PPT decreasing the runtime of Sparrow over the complete set and allowing it to solve 11 instances more (Sparrow solved 209). Independent of the used PPT, the runtime of CP3 was in most cases around five seconds and seldom above ten. When using SATELITE, the run time distribution is completely different: up to 300 seconds, this configuration can solve less instances than any other configuration, but later on, it is able to solve 240 instances. Compared to CP3, SATELITE is performing a combination of PPTs until completion which motivates the analysis of combinations of PPTs.

[6] `https://www.lri.fr/~simon/?page=glucose`

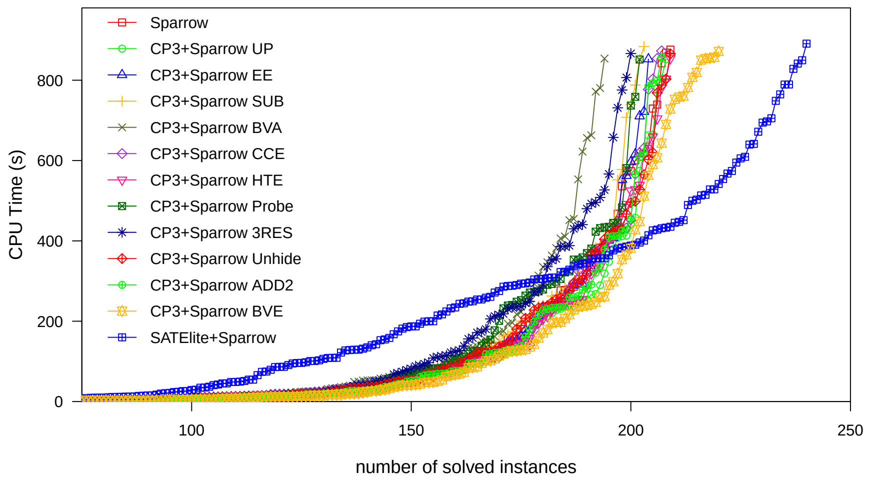

Fig. 1: Number of solved instances by Sparrow alone and when prepended with the different PPTs individually and with SatElite on the set of satisfiable Hard Combinatorial instances from the SAT Challenge 2012.

5.2 Combined PPTs Analysis

When searching for the best PPT combination and parameterization for our Sparrow scenario we have used only a small set of training instances (150 from the total of 360). The best performing configurations from our five configuration experiments described in 4 where evaluated on the complete set of instances. We report the results only for the best combined configuration, though some other configurations performed also relatively good and had similar settings. From Figure 2 we can see that the best combined PPT technique denoted in the plot wit *CP3+Sparrow combined* is able to significantly improve Sparrow, solving a total of 250 instances from the set, more than the best single engine SLS solver from the SC12.

5.3 Extended Single PPT analysis

As BVE was the best performing PPT we have performed an extra configuration experiment (see Section 4). The best found BVE configuration used the new introduced parameters in a somehow unexpected way: No Gate Detection, only SUB and not STR during the BVE loop, prefer the variable with maximum occurrences to be eliminated first. Finally, the formula is allowed to grow: per step at most $b_l = 10$ clauses can be added and in total the formula should grow by $b_u = 1000$ clauses. Consequently the preprocessed formulas can get larger in terms of clauses, but variables with large occurrences will be eliminated first. This type of BVE configuration is almost contrary to the standard configuration for CDCL solvers. In Figure 2 we can see that this configuration denoted in the plot with CP3+SPARROW EXT BVE is able to outperform the combined PPT configuration further by 11 instances solving a total of 261 instances, which is far more than any single engine solver that participated in the last years SC12. The performance increase is due to two classes of instances present in our evaluation set, the *fsf** and

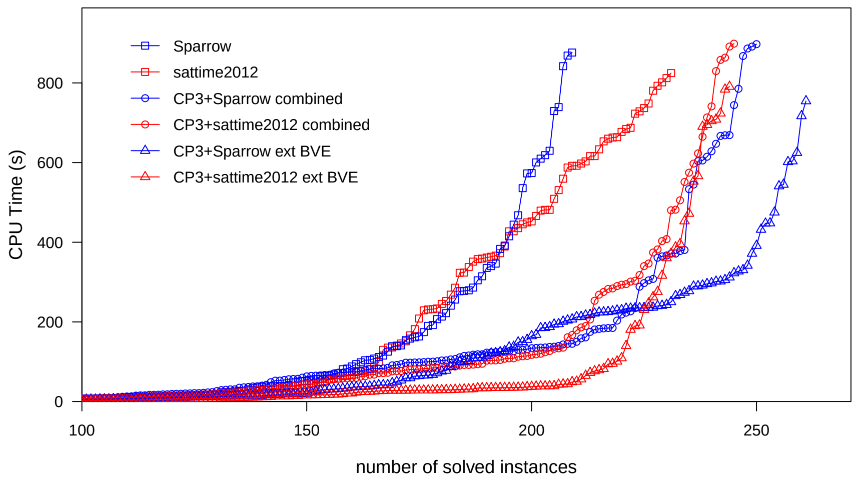

Fig. 2: Number of solved instances by Sparrow and SATTIME2012 alone and when prepended with the best combined PPTs and the best BVE parameterization on the set of satisfiable Hard Combinatorial instances from the SAT Challenge 2012.

the *prime**, which Sparrow is barely able to solve alone. When prepended with the BVE PPT configuration, SPARROW solves all these instances quite easily, having run times lower than 300 seconds. Note, that none of these instances is solved by CP3 already.

5.4 Applicability to other SLS Solvers

We have combined the best performing CP3 configuration from our combined and extended single analysis with another SLS solver, namely SATTIME2012 [12], which was the best performing single engine solver for the SHC12 instance set. SATTIME2012 already has an incorporated preprocessor, which performs failed literal probing and unit propagation. In Figure 2 we can see that the performance of SATTIME2012 can also be increased by using the combined and the extended BVE configuration, as it was the case for Sparrow. The gain in terms of number of solved instances is not that big as it is for Sparrow, but is still significant and decreases the runtime of SATTIME2012 on a large set of instances drastically (more than 200 instances can be solved in less than 50 seconds). SATTIME2012 can not benefit that much from the BVE configuration because the preprocessing is not able to make those instances solvable, which SATTIME2012 cannot solve alone, namely the *VanDerWärden** instances. Still, using another preprocessor before running the internal preprocessor of SATTIME2012 seems to pay off.

6 PPTs Analysis for CDCL Solvers

Over the years, many PPTs have been proposed to help systematic SAT solvers to improve their performance or to simulate high level domain simplifications. From todays perspective it is not clear how each single PPT contributes to the performance of CDCL solvers, especially on heterogeneous benchmarks.

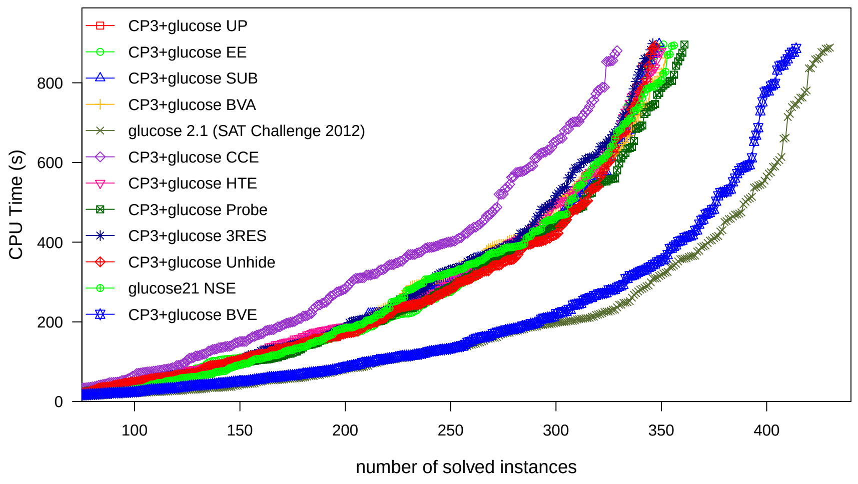

Fig. 3: Number of solved instances by GLUCOSE 2.1 when prepended with the different PPTs individually on the set of application instances from the SAT Challenge 2012. As base solver we added glucose21 NSE, which is GLUCOSE 2.1 without the SATELITE preprocessor, and as a state-of-the-art solver GLUCOSE 2.1 2.1 in the version of the SAT Challenge 2012.

6.1 Single PPT Analysis

In the following we evaluate the utility of each PPT on the APP12 benchmarks. In Figure **??** we can see the cactus plot of the different configurations – note that this plot shows only the number of solved instances, but does not tell the general utility of each technique. The plot reveals that BVE is a crucial simplification technique, which outperforms any other PPT. This fact also explains the performance of the SATELITE preprocessor, which incorporates BVE.

To identify the utility of each individual PPT, we use the *unique solver contribution* that gives the number of instances that can be solved solely by this PPT. Since we always use the same solver, this measure results in the *unique PPT contribution* (UPT):

	None	UP	3RES	SUB+STR	EE	Unhide	HTE	Probe	BVE	BVA	CCE
UPT	2	–	–	1	6	–	1	5	52	2	2
solved	356	347	346	349	351	347	350	361	414	352	329

The given data shows that the techniques UP, 3RES and Unhide do not help GLUCOSE 2.1 to improve on the used benchmark. The contribution of HTE is small, one instance can be solved by using HTE only. When using BVA, CCE or no simplification at all, each time two unique formulas can be solved. This fact motivates two conclusions: (i) there exists instances where formula simplifications change the formula so that solver heuristics do not perform as well, and (ii) there exists only a few instances in the benchmark that benefit from complex techniques like BVA or CCE. When giving more reasoning power to the preprocessor by performing look-ahead during probing, 5 instances can be solved. Furthermore, when combining equivalent literals and thus reducing the number of variables of the formula, the solver can tackle another 6 instances. Finally, the

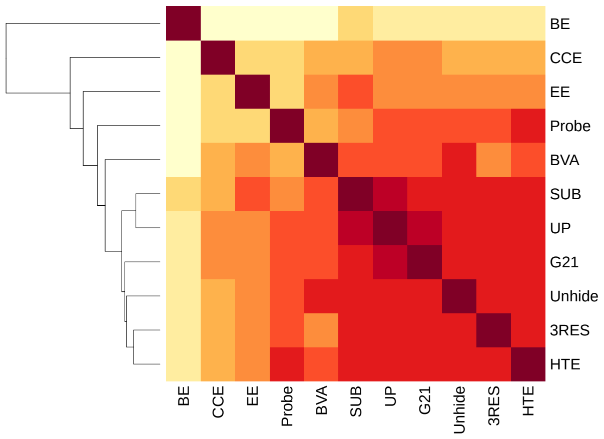

Fig. 4: Spearman correlation matrix when correlating the performance of GLUCOSE 2.1 when being combined with a single PPT. The darker each field, the more correlated is the performance of the two configurations.

most powerful technique BVE is able to solve 52 instances that cannot be solved by any other technique.

Another interesting result can be concluded from the data: a PPT is not only valuable if it helps solving the highest number of instances from a benchmark, but it is also important to have a UPT. Looking at EE, it can be seen that GLUCOSE 2.1 +EE solves less instances than using GLUCOSE 2.1 alone. Still, EE helps solving 6 instances that cannot be solved by any other configuration. For CCE, this difference is even more significant: CCE can solve 27 instances less, but still helps to solve 2 instances[7].

The given unique PPT contributions motivate a further analysis: namely the correlation of the solver performance after applying a certain PPT. Figure 4 provides a heatmap visualization of the Spearman correlation matrix between the different PPTs, where a darker field represents a higher correlation.

Two major correlations can be seen: SUB+STR simulate UP, and removing redundant clauses does not improve the performance of the CDCL solver much. Furthermore, the failed literals that are found during Probe seem to correlate to the failed literals that can be extracted during HTE. Finally, the only technique that seems to correlate with BVE is SUB+STR, which is natural, since BVE performs SUB+STR before executing the actual variable elimination.

6.2 Multiple PPTs Analysis

After seeing the unique solver contribution of each single PPT, naturally the question arises whether these technique can be combined, resulting in more powerful simplification procedures. Since most techniques offer parameters like the number of iterations, step limits, or the complexity of the applied algorithm, we have used an automatic algorithm configurator to find good configurations. The following combination of techniques has been found to suite GLUCOSE 2.1 best (PPTs are run in the specified order): UP, SUB+STR with *allStrength=3*, Unhide without UHLE and 5 iterations, BVE without

[7] We propose to study the effects of PPTs as EE and CCE and the CDCL algorithm in more details, since this is not part of the presented work

on the fly BCE, BVA with steps set to 120000, and DENSE. With this configuration, GLUCOSE 2.1 can solve 443 instances of the 600 benchmark instances, whereas when combined with SATELITE only 430 instances can be solved. For solving the provided benchmark, it seems to pay off to create additional clauses during strengthening, and to create all clauses that can follow from strengthening ternary or smaller clauses. Instead of performing the expensive HTE, the cheap approximation with Unhide is calculated. To achieve a good approximation, the number of iterations has been increased to 5 – also detecting more failed literals. By this setup, Unhide seems to cover most benefits from Probe, HTE and CCE. Next, BVE is executed in the default setup, but without removing blocked clauses. Finally, the formula is partially rewritten by using BVA, however, in a very limited way. The few number of steps indicates that the first steps of BVA are important, but running this technique until completion does not improve the performance of CDCL solvers further.

6.3 Extended Single PPT analysis

When tuning BVE further for GLUCOSE 2.1, the following configuration of BVE has been found to perform best on the given benchmark: no on the fly BCE, enabled global upper bound $b_g = 0$, enabled local elimination limit loosened $b_l = 2$ and DENSE. When using this configuration, the performance of GLUCOSE 2.1 + CP3 matches the performance of GLUCOSE 2.1 + SATELITE. It can be concluded that, except using the global and local bounds during the elimination, SATELITE has a well performing default parameter setting.

6.4 Applicability to other CDCL Solvers

Finally, we check whether the utility of the improved preprocessor configuration can be transferred to other SAT solvers as well. We chose the widely used MINISAT 2.2 as another solving engine. When using MINISAT 2.2 + SATELITE, 376 instances of the benchmark can be solved, whereas when switching to the best configuration found for CDCL solvers with CP3 (see Sect. 6.2), another 6 instances can be solved. The improvement is not as high as for GLUCOSE 2.1, however, the configuration process has been carried out for the latter solver. We therefore conclude, that the results obtained by our experiments can still be transferred to other SAT solvers.

7 Conclusion and Future Work

In this paper we have analyzed the utility of preprocessing techniques (PPTs) for CDCL and SLS solvers on application respectively satisfiable hard combinatorial problems by means of single and combined PPT analysis using automatic algorithm configuration procedures. We showed that the performance of SLS solvers can be drastically improved by using appropriate PPTs (which in our case was BVE) and that the configuration of PPTs strongly differs from configurations used in the literature for CDCL solvers. The utility of the best found PPT configuration keeps its validity even when exchanging the solver. For example, when applying preprocessing to another SLS solver like SATTIME2012, the performance also increased. Overall the PPT and solver combinations found in this paper achieve new state-of-the-art performance for SLS solvers on hard combinatorial problems.

For CDCL solvers we showed, that their performance can be further improved (which is known to be a hard task) by configuring the parameters of their PPTs. When combining variable elimination with variable addition and the approximation of HTE and failed literal detection based on the binary implication graph, a PPT configuration has been found that, combined with GLUCOSE 2.1, yields a better performance, than when using SATELITE. Our results also show that each individual PPT can be useful, allowing to solve instances not solved by any other PPT.

We propose two different directions of research motivated by our work. The first one is the improvement of the overall solving performance by extending the parameterization analysis to the solver parameters, execution order of PPT and even including the PPTs as inprocessing steps.

The second direction would be to analyze the structural changes performed by the PPTs on the instances and find out why they have a positive effect on the solving time of the solvers. In this way new PPTs could be developed that try to further improve these beneficial changes.

Acknowledgments The authors would like to thank Armin Biere for fruitful discussion about preprocessing techniques. We would like to thank the BWGrid [26] project for providing the computational resources. This project was funded by the Deutsche Forschungsgemeinschaft (DFG) under the number SCHO 302/9-1.

References

1. Kautz, H., Selman, B.: Planning as satisfiability. In: Proceedings of the 10th European conference on Artificial intelligence. ECAI '92, New York, NY, USA, John Wiley & Sons, Inc. (1992) 359–363
2. Kaiss, D., Skaba, M., Hanna, Z., Khasidashvili, Z.: Industrial strength sat-based alignability algorithm for hardware equivalence verification. In: Proceedings of the Formal Methods in Computer Aided Design. FMCAD '07, Washington, DC, USA, IEEE Computer Society (2007) 20–26
3. Großmann, P., Hölldobler, S., Manthey, N., Nachtigall, K., Opitz, J., Steinke, P.: Solving periodic event scheduling problems with sat. In Jiang, H., Ding, W., Ali, M., Wu, X., eds.: IEA/AIE. Volume 7345 of Lecture Notes in Computer Science., Springer (2012) 166–175
4. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proc. 8th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '05). Volume 3569 of LNCS., Springer (2005) 61–75
5. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Proceedings of the 16th international conference on Tools and Algorithms for the Construction and Analysis of Systems. TACAS'10, Berlin, Heidelberg, Springer-Verlag (2010) 129–144
6. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of Boolean formulas. In: Hardware and Software: Verification and Testing – 8th Int. Haifa Verification Conf. (HVC 2012). LNCS, Springer (2013) to appear.
7. Heule, M., Järvisalo, M., Biere, A.: Clause elimination procedures for cnf formulas. In: Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning. LPAR'10, Berlin, Heidelberg, Springer-Verlag (2010) 357–371
8. Heule, M.J.H., Järvisalo, M., Biere, A.: Efficient cnf simplification based on binary implication graphs. In: Proceedings of the 14th international conference on Theory and application of satisfiability testing. SAT'11, Berlin, Heidelberg, Springer-Verlag (2011) 201–215
9. Heule, M., Järvisalo, M., Biere, A.: Covered clause elimination. CoRR **abs/1011.5202** (2010)
10. Manthey, N.: Coprocessor 2.0: a flexible CNF simplifier. In: Proc. 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '12). Volume 7317 of LNCS., Springer (2012) 436–441
11. Gableske, O.: The Effect of Clause Elimination on SLS for SAT. In: Pragmatics of SAT(POS'12). (2012)

12. Li, C.M., Li, Y.: Satisfying versus falsifying in local search for satisfiability. In: Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing. SAT'12, Berlin, Heidelberg, Springer-Verlag (2012) 477–478
13. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Proceedings of the 6th international joint conference on Automated Reasoning. IJCAR'12, Berlin, Heidelberg, Springer-Verlag (2012) 355–370
14. Manthey, N., Philipp, T., Wernhard, C.: Soundness of inprocessing in clause sharing sat solvers. In: Proceedings of the 16th international conference on Theory and Applications of Satisfiability Testing. SAT'13 (2013, accepted.)
15. Gelder, A.V.: Toward leaner binary-clause reasoning in a satisfiability solver. Annals of Mathematics and Artificial Intelligence **43**(1) (2005) 239–253
16. Lynce, I., Marques-Silva, J.P.: Probing-based preprocessing techniques for propositional satisfiability. In: Proc. 15th IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI '03), IEEE (2003) 105–110
17. Katebi, H., Sakallah, K.A., Marques-Silva, J.P.: Empirical study of the anatomy of modern SAT solvers. In: Proc. 14th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT '11). Volume 6695 of LNCS., Springer (2011) 343–356
18. Wei, W., Selman, B.: Accelerating random walks. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming. CP '02, London, UK, UK, Springer-Verlag (2002) 216–232
19. Hutter, F., Babic, D., Hoos, H.H., Hu, A.J.: Boosting verification by automatic tuning of decision procedures. In: Proceedings of the Formal Methods in Computer Aided Design. FMCAD '07, Washington, DC, USA, IEEE Computer Society (2007) 27–34
20. Tompkins, D.A.D., Balint, A., Hoos, H.H.: Captain jack: new variable selection heuristics in local search for sat. In: Proceedings of the 14th international conference on Theory and application of satisfiability testing. SAT'11, Berlin, Heidelberg, Springer-Verlag (2011) 302–316
21. Li, C.M., Anbulagan, A.: Heuristics based on unit propagation for satisfiability problems. In: Proceedings of the 15th international joint conference on Artifical intelligence - Volume 1. IJCAI'97, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1997) 366–371
22. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. 38th Annual ACM/IEEE Design Automation Conf. (DAC), ACM (2001) 530–535
23. Heule, M., van Maaren, H.: Look-ahead based sat solvers. In Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (2009) 155–184
24. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the 5th international conference on Learning and Intelligent Optimization. LION'05, Berlin, Heidelberg, Springer-Verlag (2011) 507–523
25. Balint, A., Diepold, D., Gall, D., Gerber, S., Kapler, G., Retz, R.: Edacc - an advanced platform for the experiment design, administration and analysis of empirical algorithms. In: Proceedings of the 5th international conference on Learning and Intelligent Optimization. LION'05, Berlin, Heidelberg, Springer-Verlag (2011) 586–599
26. bwGRiD (http://www.bw grid.de/): Member of the german d-grid initiative, funded by the ministry of education and research (bundesministerium für bildung und forschung) and the ministry for science, research and arts baden-wuerttemberg (ministerium für wissenschaft, forschung und kunst baden-württemberg). Technical report, Universities of Baden-Württemberg (2007-2010)
27. Balint, A., Fröhlich, A. In: Improving Stochastic Local Search for SAT with a New Probability Distribution. Volume 6175. Springer (2010) 10–15