

A Portfolio Based Parallel SAT Solver with Multiple Deletion Strategies

Long Guo, Said Jabbour, Jerry Lonlac, and Lakhdar Sais

CRIL-CNRS Université d'Artois
F-62307 Lens Cedex, France
{guo, jabbour, lonlac, sais}@cril.fr

Abstract. Conflict based clause learning is known to be an important component in Modern SAT solving. Because of the exponential blow up of the size of learnt clauses database, maintaining a relevant and polynomially bounded set of learnt clauses is crucial for the efficiency of clause learning based SAT solvers. In this paper, we first compare several criteria for selecting the most relevant learnt clauses with a simple random selection strategy. We then propose new criteria allowing us to select relevant clauses w.r.t. a given search state. Then, we use such strategies as a mean to diversify the search in a portfolio based parallel solver. An experimental evaluation comparing the classical ManySAT solver with the one augmented with multiple deletion strategies, shows the interest of such approach.

Keywords: Satisfiability, Clause Learning, Parallel SAT solving

1 Introduction

The SAT problem, i.e., the problem of checking whether a set of Boolean clauses is satisfiable or not, is central to many domains of computer science and artificial intelligence (theorem proving, planning, non-monotonic reasoning, electronic design automation or knowledge-base verification and validation). During the last two decades, SAT has gained considerable audience with the advent of a new generation of SAT solvers.

In addition to the traditional hardware and software verification fields, SAT solvers are gaining popularity in new domains. For instance, they are also used for general theorem proving and computational biology. This widespread adoption is the result of the efficiency gains made during the last decade. Indeed, industrial instances with hundred of thousand of variables and millions of clauses are now solved within a few minutes. This impressive progress can be related to both the algorithmic improvements and to the ability of SAT solvers to exploit the hidden structures¹ of such instances.

However, new applications are always more challenging with instances of increasing size and complexity, while the gains traditionally given by low level algorithmic adjustments are now stalling. As a result, a large number of industrial instances from the last competitions remain challenging for all the available SAT solvers. Fortunately,

¹ By structure, we mean the dependencies between variables, which can often appear through Boolean functions. One particular example being the well-known notion of backdoors.

the previous comes at a time where the generalization of multicore hardware gives parallel processing capabilities to standard PCs. While in general it is important for existing applications to exploit these new hardware, for SAT solvers, this becomes crucial.

Many parallel SAT solvers have been previously proposed. Most of them are based on the divide-and-conquer principle (e.g. [1]). They generally divide the search space using the well-known guiding-path concept [2]. The main problems behind these approaches rise in the difficulty to get workload balanced between the different processing units and in finding the best guiding path. Also, splitting the search tree using guiding paths leads to the exploration of unrelated parts of the search space and reduces the benefit of clause sharing. Portfolio-based parallel SAT solving has been recently introduced [3]. It avoids the previous problem by letting several differentiated DPLL engines compete and cooperate to be the first to solve a given instance. Each solver works on the original formula, and search spaces are not split or decomposed anymore. To be efficient, the portfolio has to use diversified search engines with clause sharing. The key point rises in finding such strategies while maintaining an exchange of higher quality. In ManySAT [3], one of the state-of-the-art portfolio-based parallel SAT solver, such diversification is obtained by a careful combination of different restarts policies, literals polarity assignment, and learning schemes. These differentiated search strategies enhanced with clause sharing aim to explore the search space with less possible redundancies. The first rank obtained by ManySAT on the parallel track of the 2008 SAT Race and 2009 SAT competition demonstrates that portfolio-based parallel approaches clearly outperform the divide-and-conquer based ones.

However, when clause sharing is added, diversification has to be restricted in order to maximize the impact of a foreign clause whose relevance is more important in a similar or related search effort. Despite the efficiency of ManySAT, the question of finding the best portfolio of diversified strategies while maintaining a high quality of exchange remains very challenging. Indeed, two orthogonal (respectively close) strategies might reduce (respectively increase) the impact of clause sharing. Therefore, a challenging question is to maintain a good and relevant "distance" between the parts of the search space explored by the different search efforts which is equivalent to the finding of a good diversification and intensification tradeoff. This question heavily depends on the problem instance. On hard ones, it might be more convenient to direct the search towards building the same and common proof (intensification), whereas on easy ones diversifying it might be the way towards finding a short proof.

Taking this in mind, we propose to diversify the search in portfolio based parallel SAT solvers using different deletion strategies. More precisely, we propose to study different criteria to predict the importance of a learnt clause. A good criterion can help us to generate a more relevant learnt clauses database, while maintaining the performance of unit propagation.

In parallel solving, the objective is to simultaneously explore different parts of the search space. To this end, we use different learnt clauses based deletion criteria to achieve diversification in parallel SAT solvers.

In this paper, after introducing some necessary definitions and technical background, we present an experimental comparison of some well known state-of-the-art criteria with a simple strategy which randomly eliminate learnt clauses, except the binary ones

which are systematically kept in the learnt clauses database. We then propose two new criteria to identify the relevance of learnt clauses according to a given search state. These criteria are used with different reduction frequencies to manage differently the learnt clauses database. Finally, we use different deletion criteria as a way to introduce diversification in parallel SAT solvers. Before concluding, we present some experimental results obtained on SAT instances taken from the last SAT competitions.

1.1 Deletion Strategies in Modern SAT Solvers

The CDCL solvers [4, 5] are based on a nice combination of (i) clause learning [5], (ii) VSIDS heuristics [4], and (iii) restart policies [6, 7], enhanced with efficient data structures (e.g. *Watched literals*).

K. Pipatsrisawat and A. Darwiche [8] proved that modern SAT solvers formalized as a proof system are equivalent in strength to general resolution, even if the search is restarted at each conflict. This result shows that resolution-based clause learning is an important component of modern SAT solvers, since it pushes forward DPLL-like procedures from tree-like to general resolution, a more powerful proof system.

During resolution process, CDCL solvers learn new clauses that are stored in a database. Since the number of learnt clauses is proved to be exponential in the worse case, a removal of a subset of learnt clauses is carried out many times during resolution. Managing the learnt clauses database was the subject of several studies [9–11]. They are designed to cope with this combinatorial explosion problem. This issue is essentially addressed by dynamically deleting irrelevant clauses. Several deletion strategies have been designed. The most popular criterion CSIDS-D, considers a learnt clause as irrelevant if its activity or its involvement in recent conflict analysis is marginal [9]. In [10], Audemard and Simon use the number of different levels (LBD - Literal Block Distance) involved in a given learnt clause to quantify the quality of learnt clauses. In [10], clauses with smaller LBD are considered as more relevant. The set of different levels present in a learned clause has been used in [12] to prove the optimality of the First UIP scheme. More recently, a new dynamic management policy of the learnt clauses database is proposed in [11]. It is based on a dynamic freezing and activation principle of the learnt clauses. At a given search state, using a relevant selection function based on progress saving (PSM), it activates the most promising learnt clauses while freezing irrelevant ones.

These cleaning strategies aim to predict the importance of learnt clauses. Since these strategies differ in both the reduction frequency and in the number of clauses to be removed, it is difficult to determine which criterion (CSIDS-D or LBD) is better. Therefore, under the condition that the reduction frequency and the size of learnt clauses database after cleaning are fixed, the problem of determining the best criterion remains very challenging and computationally hard.

2 Definitions, Notations and Technical Background

In this section, after some preliminary definitions and notations, we introduce the most important computational features of modern SAT solvers.

A CNF formula Σ is a conjunction (interpreted as a set) of clauses, where a clause is a disjunction (interpreted as a set) of literals. A literal is a positive (x) or negative ($\neg x$) Boolean variable. The two literals x and $\neg x$ are called *complementary*. A unit clause is a clause with only one literal (called *unit literal*). An *empty clause*, is interpreted as false, while an *empty CNF formula*, is interpreted as true. A set of literals is *complete* if it contains one literal for each variable occurring in Σ and *fundamental* if it does not contain complementary literals. An *interpretation* $\mathcal{I}$ of a Boolean formula Σ associates a value $\mathcal{I}(x)$ to some of the variables x appearing in Σ . An interpretation can be represented by a fundamental set of literals, in the obvious way. A *model* of a formula Σ is an interpretation $\mathcal{I}$ that satisfies the formula, *i.e.*, that satisfies all clauses of the formula. Finally, SAT is the problem of deciding whether a given CNF formula Σ admits a model or not.

Let us now briefly describe the basic components of CDCL-based SAT solvers [4, 9]. To be exhaustive, these solvers incorporate unit propagation (enhanced by efficient and lazy data structures), variable activity-based heuristic, literal polarity phase, clause learning, restarts and a learnt clauses database reduction policy.

Algorithm 1: CDCL solver

Input: a CNF formula Σ
Output: SAT ou UNSAT

```

1  $\mathcal{I} = \emptyset$ ;                                     /* interpretation */
2  $\Delta = \emptyset$ ;                                /* learnt clauses database */
3  $dl = 0$ ;                                         /* decision level */
4 while (true) do
5    $\gamma = \text{unitPropagation}(\Sigma, \mathcal{I})$ ;
6   if ( $\gamma \neq \text{null}$ ) then
7      $\beta = \text{conflictAnalysis}(\Sigma, \mathcal{I}, \gamma)$ ;
8      $btl = \text{computeBackjumpLevel}(\beta, \mathcal{I})$ ;
9     if ( $btl == 0$ ) then return UNSAT;
10     $\Delta = \Delta \cup \{\beta\}$ ;
11    if (restart()) then  $btl = 0$ ;
12     $\text{backjump}(btl)$ ;
13     $dl = btl$ ;
14  else
15    if ( $\mathcal{I} \models \Sigma$ ) then
16      return SAT;
17    end
18    if (timeToReduce()) then  $\text{reduceDB}(\Delta)$ ;
19     $\ell = \text{selectDecisionVariable}(\Sigma)$ ;
20     $dl = dl + 1$ ;
21     $\mathcal{I} = \mathcal{I} \cup \{\text{selectPhase}(\ell)\}$ ;
22  end
23 end

```

Algorithm 1 depicts the general scheme of CDCL SAT solver. Typically, this solver can be assimilated to a sequence of decision and unit propagation literals. Each literal chosen as a decision variable is affected to a new decision level (dl) with given polarity (lines 19-21). If all literals are assigned, then $\mathcal{I}$ is a model of σ and the formula is answered to be satisfiable (lines 15-16). If a conflict is reached by unit propagation (γ is the conflict clause), a new asserting clause β is derived by conflict analysis (line 7), mostly following the 'First-UIP' scheme ('Unique Implication Point' [5]), and a backtrack level (btl) is computed (line 8). If btl is null, then the formula is answered unsatisfiable (line 9), otherwise β is added to the learnt clauses database (line 10). Some CDCL solvers perform a restart (various restart strategies are possible [6]) and in this case they backtrack to the root of the search tree (line 11)).

When it is time to reduce, the learnt clauses database is reduced (line 18). This component is clearly crucial to the solver's performance. In fact, keeping too many learnt clauses will slow down the unit propagation process, while deleting too much will break the overall learning benefit. Consequently, identifying good learnt clauses - relevant to the proof derivation - is clearly an important challenge.

Algorithm 2: Elimination Strategy: `reduceDB` function

Input: Δ : The learnt clauses database of size n
Output: Δ The new learnt clauses database of size $n/2$

```

1 sortLerantClauses() ;      /* by the defined criterion */
2 limit = n/2;
3 ind = 0;
4 while ind < limit do
5   clause =  $\Delta[ind]$  ;
6   if clause.size() > 2 then
7     removeClause() ;
8   else
9     saveClause() ;
10  ind ++;
11 return  $\Delta$  ;
```

Algorithm 2 depicts the general scheme of an elimination strategy of learnt clauses (`reduceDB( $\Delta$ )`). To define a good elimination strategy, the following two questions need to be answered: (1) Which clauses to remove? and when?. Determining what is a relevant clause before completing the proof is of great importance for the efficiency of the solver. However, answering such question is computationally hard since it is related to the problem of finding a proof of minimal size. The existing strategies keep systematically the binary clauses and the clauses which are reasons of propagated literals. To determine the time steps for reducing the learnt clauses database, different functions are generally used.

In order to include diversification in the parallel portfolio, in the next section, we present two new policies to assess the relevance of a clause with respect to a given search state.

3 Identification Criteria of Relevant Learnt Clauses

As mentioned above a CDCL-based SAT solver can be formulated as a resolution proof system [8]. In practice, the main problem behind resolution-based techniques arises from their exponential space complexity. Consequently, the practical incarnation of modern SAT solvers can be seen as a resolution-based procedure with a deletion strategy. As a consequence, the completeness of modern SAT solvers is heavily dependent on both the chosen deletion strategy and restart policy. For example, if we use a restart with a static cutoff value and an aggressive deletion strategy, we can not guarantee the completeness of the solver. For this reason one needs to be careful when designing a deletion strategy. Thus, defining what is a relevant clause before completing the proof itself is of great importance for the efficiency of the solver.

3.1 BackTrack Level Based Clause Activity

In this section, we present a new criteria to deal with the problem of determining relevant learnt clauses with respect to the search process in the context of SAT solving.

Let us recall that during search, at each encountered conflict, a linear resolution derivation is performed to obtain a learnt clause c . Such clause is derived from the conflict thanks to successive resolution steps obtained by a bottom up traversal of the implication graph. Let $\langle c_1, c_2, \dots, c_n \rangle$ be the sequence of intermediate resolvents obtained by such resolution process, where $c_n = c$. The learnt clause c is then used by the solver to backtrack to a given node in the search tree and to propagate the asserting literal. Recently several works [6] have shown experimentally the correlation between the restart strategy and the efficiency of clause learning especially on application instances. These research works showed that a rapid restart allows the solver to learn relevant clauses leading to a shorter proofs. Starting from this observation, we suppose that rapid restarts allow us to prune the branches early in the search tree in order to obtain short unsatisfiability proofs. From this hypothesis, one can expect that the depth of the search tree should be small due to the relevant learnt clauses and to the rapid restarts. From all these observations, we assume that a clause allowing to backtrack higher in the search tree is more relevant. Based on this assumption, we define a new criterion to quantify the relevance of a clause using its backtrack level called BTL for Backtrack Level Based Clause Quality and defined as follows: Let c be a learnt clause and ℓ the backtrack level associated to the clause c , we define $\text{BTL}(c) = \ell$. For example, if we learn the clause $c = (x_1^1 \vee x_2^8 \vee x_2^9 \vee x_3^{15})$, then $\text{BTL}(c) = 9$, since the conflict occurs at level 15. Note that x_i^j indicates that the literal x_i is assigned at level j .

To analyze and to validate the assumption about the relationship between clause relevance and their backtracking level, experiments are conducted on the following SAT instances taken from different families: `manol-pipe-f7nidw`, `vmpe-32.renamed-as.sat05-1919`, `velev-pipe-o-uns-1.0-7` and `q-query-3-L70-coli.sat`.

Figure 1 shows, for a sample of instances, the total number of times clauses with a given BTL value are used during unit propagation. In this experiment, a counter $\text{cnt}[n]$ is associated to each BTL value n and increased each time a clause of BTL equal to n is used during propagation.

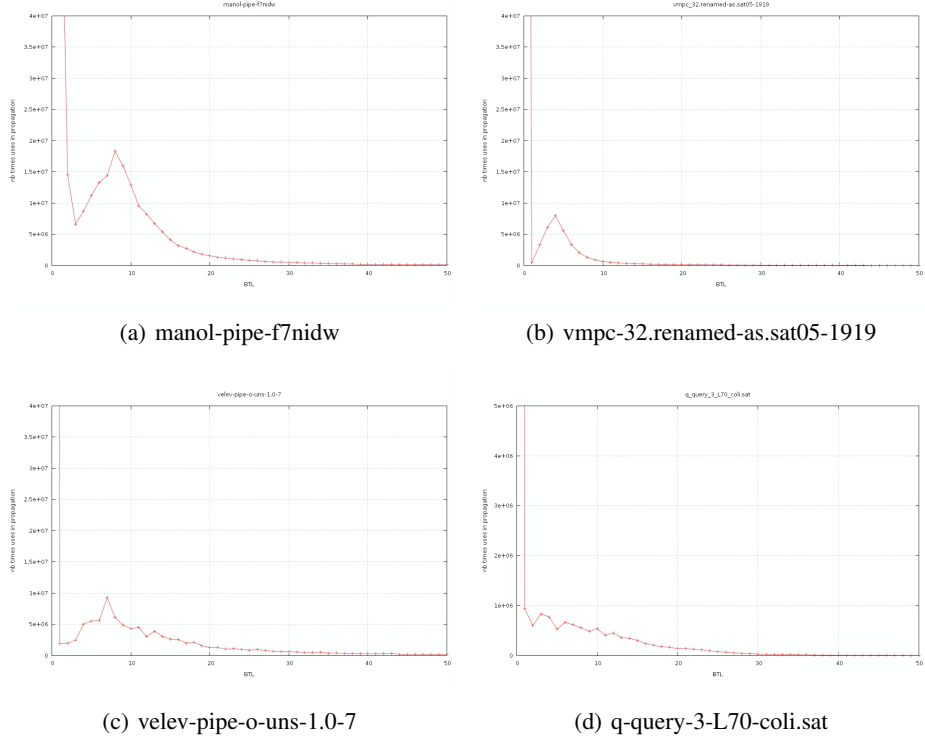


Fig. 1. BackTrack Level measure (BTL) /relevance with respect to UP

Each dot (x, y) in Figure 1 presents the total number of times y the clause with a given BTL value x is involved in unit propagation.

As we can observe from Figure 1, learnt clauses with small BTL values are used more often in the unit propagation process than those with higher BTL values. More precisely, the learnt clauses with BTL value less than 3 are always used much more than the remaining clauses. Indeed, a learnt clause with smaller BTL contains more literals from the top of the search tree.

From these observations, we can deduce that relevant clauses are those allowing a higher backtracking in the search tree or small BTL value. In practice, to use the BTL criterion as a mean to reduce the learnt clauses database, we attribute to each clause c an activity equal to the inverse of its backtracking level value. Such definition of the activity allows us to keep the clauses that prune early the search tree. More formally, we attribute to the asserting clause c an activity $act(c) = \frac{1}{BTL(c)}$. Furthermore, to get a more dynamic criterion, the activity of learnt clauses are updated during the search as follows. Suppose that a learnt clause c is used at level ℓ to propagate a literal x , if ℓ is smaller than the current $BTL(c)$, the new value of $BTL(c)$ is set to ℓ i.e., $act(c) = \min(act(c), \frac{1}{\ell})$.

4 Integration in Portfolio Based Parallel SAT solvers

The key idea of portfolio based parallel SAT solving is to simultaneously explore different parts of the search space using different solvers. In ManySAT, the solvers used by each core differ mainly in their restarts strategies, clauses learning schemas, variables ordering heuristics and polarity phase. In this section, we introduce another way to diversify the search using different deletion strategies presented previously. Indeed, the use of different policies for reducing the learnt clauses database, induces different resolution strategies. More precisely, in ManySAT portfolio based parallel SAT solver we adopted the following diversification scheme : we use CSIDS-D in the cores 1 and 2, LBD in the cores 3 and 4, BTL in the cores 5 and 6, and CSIDS-S for the cores 7 and 8 (see Figure 2). CSIDS-D (D for dynamic) corresponds to the reduction strategy used in MiniSAT 2.2. At each conflict, the activity of each encountered learnt clause during the conflict analysis is increased with a special increment. This special increment is dynamically increased at each conflict to favor clauses used in the most recent conflicts. The CSIDS-S (S for static) strategy can be considered as a variant of CSIDS-D where the special increment is set to 1 and not increased at all during search. The other strategies LBD and BTL are explained in the previous sections.

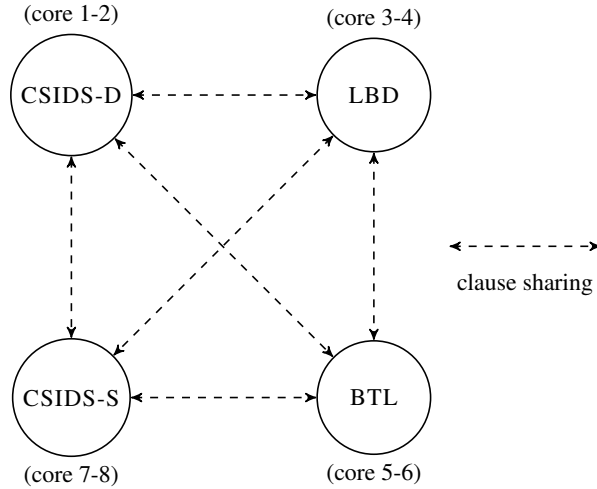


Fig. 2. Diversification using different deletion strategies in ManySAT (8 Cores).

5 Experiments

In this section, we evaluate the performance of our ManySAT solver with diversified deletion strategies. The experiments were conducted on a large panel of industrial SAT

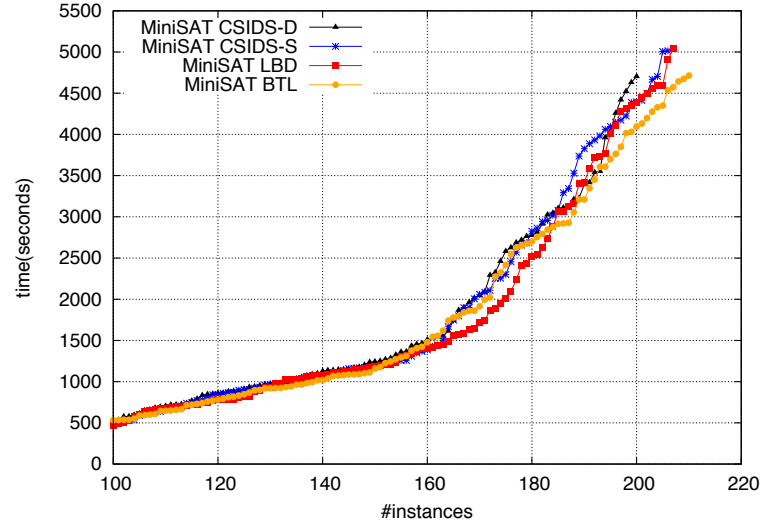


Fig. 3. 2013 SAT Competition, Industrial: cumulated time

instances from the competition SAT’2013 (300 instances in application category). All the instances were simplified by the *SatElite* preprocessor [13].

To get an overview on the different state-of-the-art criteria used to reduce the learnt clauses database and the proposed one, we conducted a first experiment to compare the four well-known criteria CSIDS-D, CSIDS-S, LBD, and BTL. These criteria are integrated in MINISAT 2.2. All the experimental results presented in this paper have been obtained with a Quad-core Intel Xeon X5550 (2.66GHz, 32 GB RAM) cluster. Results are reported in seconds. The time limit was fixed to 5000 seconds.

The results obtained using the four criteria CSIDS-D, CSIDS-S, LBD, and BTL integrated in MINISAT 2.2 are depicted in Figure 3. The x-axis shows the number of solved instances under a given time limit.

As we can observe, the behavior of the different solvers is roughly similar. Minisat enhanced with the BTL criterion is slightly better and solve 210 instances while with LBD policy the number of solved instances is equal to 207. Classical Minisat with CSIDS-D solves 200 instances, while Minisat with CSIDS-S solves 206 instances.

Let us recall that our goal in this paper is to build a portfolio based parallel SAT solver using several deletion strategies. As each deletion strategy induces a different resolution strategy, integrating several different learned clauses deletion policies leads us to a powerful diversification mechanism. Tables 1 and 2 show the solved SAT instances where at least one solver fails to solve the instance. The first column indicates the name of the instance, while the remaining columns give the wall-clock time (in seconds) for each solver. From the two Table 1 and 2, we found 21 instances solved by only one solver. Interestingly, BTL criterion allows to solve 15 instances not solved with other criteria. This clearly shows the potential of our new BTL measure.

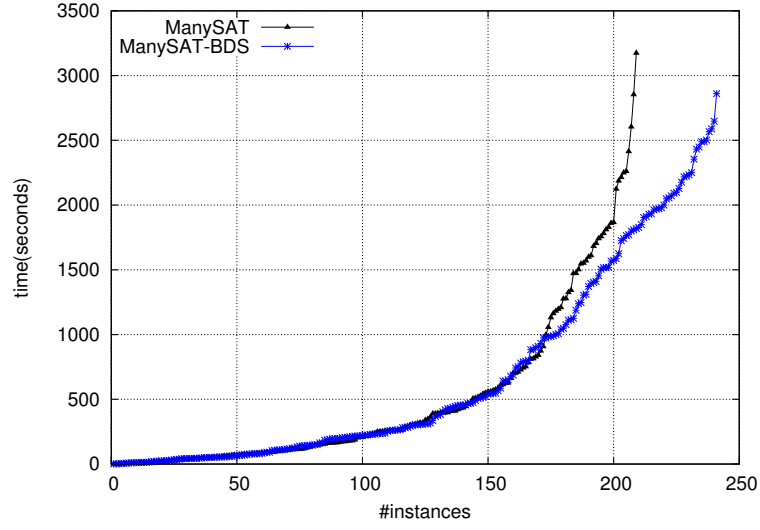


Fig. 4. A Comparative Evaluation of ManySAT 2.0 and ManySAT-BDS

We implemented the different learnt clauses deletion strategies in ManySAT 2.0 leading to a new parallel solver ManySAT-BDS. In our second experiment, we compare classical ManySAT 2.0. with our new solver ManySAT-BDS.

Figure 4 shows the solving time obtained by ManySAT and ManySAT-BDS on the set of 300 SAT instances taken from the SAT 2013 competition (application category). The cactus plot represents the number of instances solved under a given CPU time limit. As we can observe, ManySAT-BDS outperforms considerably ManySAT. Indeed, the new solver is faster than the classical one. Furthermore, ManySAT-BDS solves 241 instances where classical ManySAT does not exceed 209 instances. To better illustrate the behavior of the new solver ManySAT-BDS, Table 3 summarizes the number of satisfiable (respectively unsatisfiable) solved instances (#solved) and the average CPU time (avg.). This table confirms that enhancing ManySAT with multiple deletion strategies allows to increase the performances of the solver on both satisfiable and unsatisfiable instances.

Finally, Table 4 highlights the results through some families of instances. For each instance, we mention in the second column if it is satisfiable (respectively unsatisfiable). Our experiment shows that diversifying the search with multiple deletion strategies allows the new solver to outperform the classical one on some families of instances e.g. sha-*, 6s*, hitag2-*.

6 Conclusion

In this paper, we showed how diversification can be performed in portfolio based parallel solvers using learned clauses deletion strategies. First, a new criterion called BTL

allowing us to quantify the relevance of a learned clause according to its backjumping level. Secondly, we integrate four different deletion strategies on the top of ManySAT solver. The experimental evaluation shows that ManySAT is significantly improved thanks to the integration of several learned clauses deletion strategies.

References

1. Lewis, M.D.T., Schubert, T., Becker, B.: Multithreaded sat solving. In: ASP-DAC. (2007) 926–931
2. Zhang, H., Bonacina, M.P., Hsiang, J.: Psato: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation* **21** (1996) 543–560
3. Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: a parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation* **6** (2009) 245–262
4. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC. (2001) 530–535
5. Zhang, L., Madigan, C., Moskewicz, M., Malik, S.: Efficient conflict driven learning in boolean satisfiability solver. In: ICCAD. (2001) 279–285
6. Huang, J.: The effect of restarts on the efficiency of clause learning. In: proceedings of IJCAI. (2007) 2318–2323
7. Gomes, C.P., Selman, B., Kautz, H.A.: Boosting combinatorial search through randomization. In: AAAI/IAAI. (1998) 431–437
8. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning sat solvers with restarts. In: CP. (2009) 654–668
9. Eén, N., Sörensson, N.: An extensible sat-solver. In: SAT. (2003) 502–518
10. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern sat solvers. In: IJCAI. (2009) 399–404
11. Audemard, G., Lagniez, J.M., Mazure, B., Sais, L.: On freezing and reactivating learnt clauses. In: SAT. (2011) 188–200
12. Audemard, G., Bordeaux, L., Hamadi, Y., Jabbour, S., Sais, L.: A generalized framework for conflict analysis. Technical Report MSR-TR-2008-34, Microsoft Research (Feb 2008)
13. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: SAT. (2005) 61–75

Instance	CSIDS-D	CSIDS-S	LBD	BTL
minandmaxor128	2815.11	3018.63	4588.50	—
bivium-40-200-0s0-0x66b*-50	—	—	—	1858.61
hitag2-8-60-0-0x880*-25-SAT	3118.98	—	—	782.48
hitag2-8-60-0-0xa3b8*-42	—	4224.00	—	—
hitag2-8-60-0-0xcdcf*-37	—	3737.32	—	—
bivium-39-200-0s0-0x53*-63	—	—	—	2624.62
bivium-39-200-0s0-0x5fa*-30	—	—	—	4132.15
bivium-39-200-0s0-0x28d*-98	—	—	—	1778.67
bivium-39-200-0s0-0x1b77*-34	—	—	—	4572.68
bivium-39-200-0s0-0xcdcf*-43	—	—	—	4030.15
hitag2-10-60-0-0x8ed*-65	—	4173.94	4443.00	4095.38
hitag2-10-60-0-0xdf7*-17	—	3933.46	4108.01	2916.57
hitag2-10-60-0-0xe14*a-99	—	—	—	4642.22
hitag2-10-60-0-0xfe*a2-78	—	—	5048.35	4328.96
hitag2-10-60-0-0xbc*13-10	—	4395.29	4313.57	3851.71
bob12m06	—	4703.50	4595.17	—
bob12s06	4258.01	3826.43	3773.58	—
6s9	—	2863.95	—	4674.12
beempgsol2b1	4630.74	3982.13	385.77	—
beempgsol5b1	3047.66	4146.59	328.52	—
bob12m06	4523.41	4098.11	4284.14	—
6s133	—	3530.87	—	2414.67
6s19	—	2658.00	—	—
6s165	124.31	—	674.25	607.10
6s166	—	—	—	4349.47
b04_s_unknown	4419.50	4495.89	—	—
ctl_4291_567_11_unsat	—	—	4494.46	3345.95
ctl_4291_567_8_unsat_pre	—	4060.79	3590.71	2875.94
ctl_4291_567_8_unsat	—	4388.35	3720.36	2790.84
ctl_4291_567_11_unsat_pre	—	5008.00	4552.49	3454.45
nossum-sha1-80-8-005	—	2253.18	415.17	—
nossum-sha1-80-8-006	594.74	383.15	1559.71	—
nossum-sha1-22-144-009	981.60	1430.18	414.94	—
nossum-sha1-22-144-002	—	1205.52	2430.90	4276.14
nossum-sha1-22-144-003	2718.56	—	653.94	2322.75
nossum-sha1-22-144-007	—	—	—	2651.99
nossum-sha1-22-144-008	—	1159.54	3061.56	—
nossum-sha1-22-160-009	—	—	—	642.04
nossum-sha1-22-160-002	1136.65	—	353.42	—
nossum-sha1-22-160-003	766.75	2299.46	—	2844.82
nossum-sha1-22-160-001	—	897.75	929.57	1477.72
nossum-sha1-22-160-008	—	—	—	3698.16
nossum-sha1-22-128-005	—	1797.33	—	1293.32
nossum-sha1-22-128-010	1150.12	1746.61	1178.81	—
nossum-sha1-22-128-002	2625.69	65.79	—	1617.26
nossum-sha1-22-128-006	143.40	—	—	1074.18
nossum-sha1-22-128-003	2460.28	—	4348.75	—
nossum-sha1-22-128-007	401.06	—	—	2701.82
nossum-sha1-23-96-003	2775.39	—	1123.98	—
nossum-sha1-23-80-001	—	—	—	965.15
nossum-sha1-23-64-003	1203.25	—	—	—

Table 1. Deletion Strategies: A Comparative Evaluation

Instance	CSIDS-D	CSIDS-S	LBD	BTL
pb_400.09.lb.04	4703.33	5014.46	4903.82	—
pb_300.05.lb.11	—	—	—	—
pb_200.10.lb.15	2014.78	2060.41	1582.71	—
grieu-vmc-31	—	—	1738.84	1406.05
vmc_34	—	—	94.16	404.12
slp-synthesis-aes-bottom14	—	3887.65	—	—
smtlib-qfbv-*vc149789-tseitin	1961.58	2454.16	1637.89	—
hwmcc10-*k45-pdtvissoap1-tseitin	3110.39	3078.91	3728.47	—
aes_32_3_keyfind_1	2092.95	—	—	1556.19
ndhf_xits_19_UNKNOWN	863.09	—	—	—
gss-22-s100	3024.96	3292.95	—	—
maxxor032	1035.60	—	1107.27	2754.45
vmc_32.renamed-as.sat05-1919	576.61	—	2009.86	421.37
dated-5-13-u	—	—	—	3607.80
gss-23-s100	2043.03	—	—	4715.43
gss-24-s100	—	—	1649.17	3055.19
7pipe_k	3961.01	2826.15	2524.59	—
aes_64_1_keyfind_1	—	2255.65	1335.90	194.59
vmc_36.renamed-as.sat05-1922	—	—	4383.14	—
dated-5-19-u	—	4668.10	—	—

Table 2. Deletion Strategies: A Comparative Evaluation

ManySAT		ManySAT-BDS	
SAT	UNSAT	SAT	UNSAT
(#solved, avg.)	(#solved, avg.)	(#solved, avg.)	(#solved, avg.)
(127 – 427.00)	(82 – 615.95)	(136 – 528.31)	(105 – 987.72)

Table 3. Results on SAT Challenge 2013 (Application Category)

Instance	SAT	ManySAT	ManySAT-BDS
hitag2-8-60-0-0x880693399044612-25-SAT	U	1682.38	882.59
hitag2-8-60-0-0xa3b8497b8aad6d7-42	U	–	1910.92
hitag2-8-60-0-0xfba1a41b5dfd7f7-52	U	–	1968.75
hitag2-8-60-0-0xdcdbc8bf368ee73-37	U	–	1979.40
hitag2-7-60-0-0xc048b9ebae66e9d-32-SAT	U	2853.25	542.24
hitag2-7-60-0-0x39ff85d4ef127de-52	U	–	1849.61
hitag2-7-60-0-0xe97b5f1bee04d70-47	U	–	1724.81
6s9	–	–	1808.71
beempgsol2b1	U	828.36	475.52
beempgsol5b1	U	1551.81	317.48
bob12m06	U	2260.15	1504.73
6s133	U	–	985.44
6s19	U	–	2223.21
6s137	U	3173.36	1403.65
nossum-sha1-22-144-005	S	1780.93	429.48
nossum-sha1-22-144-009	S	1569.79	–
nossum-sha1-22-144-002	S	2186.13	2000.02
nossum-sha1-22-144-003	S	–	2098.25
nossum-sha1-22-144-007	S	–	469.15
nossum-sha1-22-144-001	S	–	2354.56
nossum-sha1-22-144-008	S	–	2070.83
nossum-sha1-22-160-009	S	–	990.36
nossum-sha1-22-160-002	S	–	1188.20

Table 4. ManySAT and ManySAT-BDS On Some Families of Instances