Predicting Learnt Clauses Quality in Modern SAT Solvers*

Gilles Audemard
Univ. Lille-Nord de France
CRIL/CNRS UMR8188
Lens, F-62307
audemard@cril.fr

Laurent Simon
Univ. Paris-Sud
LRI/CNRS UMR 8623 / INRIA Saclay
Orsay, F-91405
simon@lri.fr

Abstract

Beside impressive progresses made by SAT solvers over the last ten years, only few works tried to understand why Conflict Directed Clause Learning algorithms (CDCL) are so strong and efficient on most industrial applications. We report in this work a key observation of CDCL solvers behavior on this family of benchmarks and explain it by an unsuspected side effect of their particular Clause Learning scheme. This new paradigm allows us to solve an important, still open, question: How to designing a fast, static, accurate, and predictive measure of new learnt clauses pertinence. Our paper is followed by empirical evidences that show how our new learning scheme improves state-of-the art results by an order of magnitude on both SAT and UNSAT industrial problems.

1 Introduction

Only fifteen years ago, the SAT problem was mainly considered as theoretical, and polynomial reduction to it was a tool to show the intractability of any new problem. Nowadays, with the introduction of lazy data-structures, efficient learning mechanisms and activity-based heuristics [Moskewicz *et al.*, 2001; Eén and Sörensson, 2003], the picture is quite different. In many combinatorial fields, the state-of-the-art schema is often to use an adequate polynomial reduction to this canonical NP-Complete problem, and then to solve it efficiently with a SAT solver. This is especially true with the help of Conflict Directed Clause Learning algorithms (CDCL), an efficient SAT algorithmic framework, where our work takes place. With the help of those algorithms, valuable problems [Prasad *et al.*, 2005] are efficiently solved every day.

However, the global picture is not so perfect. For instance, since the introduction of ZCHAFF [Moskewicz *et al.*, 2001] and MINISAT [Eén and Biere, 2005], the global architecture of solvers is stalling, despite important efforts of the whole community [Le Berre *et al.*, 2007]. Of course, improvements have been made, but they are sometimes reduced to data-structures tricks. It may be thus argued that most solvers are often described as a compact re-encoding of ideas of Chaff, a height-years old solver, with small improvements only (phase caching [Pipatsrisawat and Darwiche, 2007], luby restarts [Huang, 2007], ...).

Since the breakthrough of Chaff, most effort in the design of efficient SAT solvers has always been focused on efficient Boolean Constraint Propagation (BCP), the heart of all modern SAT solvers. The global idea is to reach conflicts as soon as possible, but with no guarantees on the new learnt clause usefulness. Following the successful idea of the Variable State Independent Decaying Sum (VSIDS) heuristics, which favours variables that were often – and recently – used in conflict analysis, future learnt clause usefulness was supposed to be related to its activity in recent conflicts analyses.

In this context, detecting what is a good learnt clause in advance is still considered as a challenge, and from first importance: deleting useful clauses can be dramatic in practice. To prevent this, solvers have to let the maximum number of learnt clauses grow exponentially. On very hard benchmarks, CDCL solvers hangs-up for memory problems and, even if they don't, their greedy learning scheme deteriorates their heart: BCP performances.

We report, in section 2, experimental evidences showing an unsuspected phenomenon of CDCL solvers on industrial problems. Based on these observations, we present, in section 3, our static measure of learnt clause usefulness and, in section 4, we discuss the introduction of our measure in CDCL solvers. In section 5, we compare our solver with state-of-the-art ones, showing order of magnitude improvements.

2 Decision levels regularly decrease

Writing an experimental study in the first section of a paper is not usual. Paradoxically, CDCL solvers lack for strong empirical studies: most papers contain experimental sections, but focus is often given to achievement illustrations only.

In this section, we emphasize an observation – decision level is decreasing – made on most CDCL solvers on most industrial benchmarks. This behavior may shed a new light on the underlying reasons of the good performances of the CDCL paradigm. The relationship between performances and decreasing is the basis of our work in the following sections.

For lack of space, we suppose the reader familiar with Satisfiability notions (variables x_i, literal x_i or $\neg x_i$, clause, unit clause and so on). We just recall the global schema of CDCL

*supported by ANR UNLOC project n° BLAN08-1_328904

Series	#Benchs	% Decr.	$-n/m(>0)$	Reduc.
een	8	62%	1.1×10^3	1762%
goldb	11	100%	1.4×10^6	93%
grieu	7	71%	1.3×10^6	–
hoons	5	100%	7.2×10^4	123%
ibm-2002	7	71%	4.6×10^4	28%
ibm-2004	13	92%	1.9×10^5	52%
manol-pipe	55	91%	1.9×10^5	64%
miz	13	0%	–	–
schup	5	80%	4.8×10^5	32%
simon	10	90%	1.1×10^6	50%
vange	3	66%	4.0×10^5	6%
velev	54	92%	1.5×10^5	81%
all	199	83%	3.2×10^5	68%

Table 1: $x_j = -n/m$ (fourth column) is the positive look-back "justification" of the number of conflicts needed to solve benchmarks, median value over all benchmarks that show a decreasing of their decision level (%Decr). Last column will be discussed in section 5.

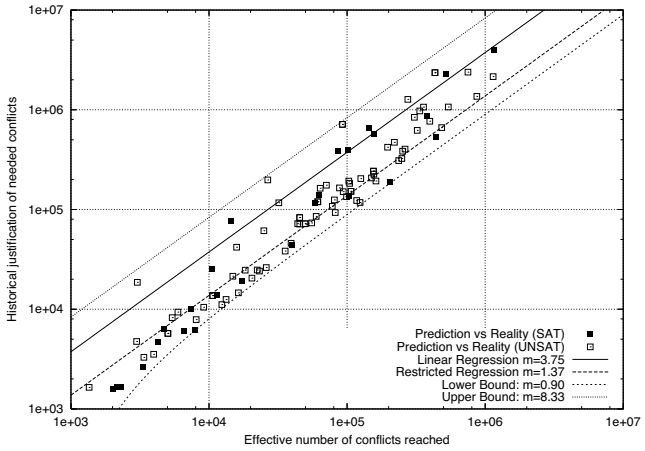

Figure 1: Relationship between look-back justification and effective number of conflicts needed by MINISAT to solve the instance.

solvers: A typical branch of a CDCL solver can be seen as a sequence of decisions followed by propagations, repeated until a conflict is reached. Each decision literal is assigned at its own level (starting from 1), shared with all propagated literals assigned at the same level. Each time a conflict is reached, a *nogood* is extracted using a particular method, usually the First UIP (Unique Implication Point) one [Zhang and Madigan, 2001]. The learnt clause is then added to the clause database and a *backjumping* level is computed from it. The interested reader can refer to [Marques-Silva *et al.*, 2009] for more details.

2.1 Observing the decreasing

The experimental study is done as follows. We run MINISAT on a selection of benchmarks from last SAT contests and races. For a given benchmark, each time a conflict x_c is reached, we store the decision level y_l where it occurs. We limit the search to 2 million of conflicts. Then, we compute the simple least-square linear regression on the line $y = m \times x + n$ that fits the set of couples (x_c, y_l). If m is negative (resp. positive) then decision levels decrease during search (resp. increase). In case of decreasing, we can trivially "*predict*" when the solver will finish the search. This should occur, if the solver follows its characteristic line, when this line intersects the x-axis. We call this point the "*look-back justification*" of the solver performance (looking-back is necessary to compute the characteristic line). The coordinates of this point are $(x_j = -n/m, 0)$. This value gives also a good intuition of how decision levels decrease during search. Note that we make very strong hypotheses here: (1) the solver follows a linear decreasing of its decision levels (this is false in most of the cases, but sufficient to compute the look-back justification), (2) finding a contradiction or a solution gives the same look-back justification and (3) the solution (or contradiction) is not found by chance at any point of the computation.

Table 1 shows the median values of x_j (fourth column) over some series of benchmarks. "#Benchs" gives the number of benchmarks in series, "%Decr." gives the percentage of benchmarks that exhibits a decreasing of decision levels. If a cut-off occurs, then the last value of the estimation is taken. In most of the cases (167 over 199 benchmarks), the regression line is decreasing, and some small values of x_0 illustrates important ones. This phenomenon was not observed on random problems and seems heavily related to the "industrial" origin of benchmarks: the "mizh" series is 100% increasing, but it encodes cryptographic problems.

2.2 Justifying the number of reached conflicts

At this point of the discussion, we showed that decision levels are decreasing in most of the cases. However, is it really a strong – but unsuspected – explanation of the power of CDCL solvers, or just a natural side effect of their learning schema? It is indeed possible that learning clauses trivially add more constraints, and conflicts are obviously reached at lower and lower levels in the search tree. However, if the look-back justification is a strong estimation of the effective number of conflicts needed by the solver to find the contradiction (or, amazingly, a solution), then this probably means that the decreasing reveals an important characteristic of their overall behavior and strength: CDCL solvers enforce the decision level to decrease along the computation, whatever the fitting quality of the linear regression.

Figure 1 exhibits the relationship between look-back justification and effective number of conflicts needed by MINISAT to solve the instance. It is built as follows: Each dot (x, y) represents an instance. x corresponds to the effective number of conflicts needed by MINISAT to solve the instance, y corresponds to the look-back justification. Only instances solved in less than 2 million of conflicts are represented (otherwise, we do not have the real look-back justification). Figure 1 clearly shows the strong relationship between justification and effective number of conflicts needed: in all the cases, the look-back justification is bounded between 0.90 and 8.33 times the real number of conflicts needed by MINISAT to solve the problem. In most of the cases, the justification is

around 1.37 times the effective number of conflicts. On may also notice that no distinction can be made between justification over SAT and UNSAT instances.

It is important here to notice that the aim of our study is not to predict the CPU time of the solver like in [Hutter *et al.*, 2006]. Our goal is, whatever the error obtained when computing the regression line, to show the strong relationship between the overall decreasing of decision levels and the performances of the solver. What is really striking is that the look-back justification is also strong when the solver finds a solution. When a solution exists, it is never found suddenly, by chance. This suggests that, on SAT instances, the solver does not correctly guess a value for a literal, but learns that the opposite value directly leads to a contradiction. Thus, if we find the part of the learning schema that enforces this decreasing, we may be able (1) to speed-up the decreasing, and thus the CPU time of the solver, and (2) to identify in advance the clauses that play this particular role for protection and aggressive clause database deletion.

3 Identifying good clauses in advance

Trying to enhance the decreasing of decision levels during search is not new. It was already pointed out, but from a general perspective only, in earlier papers on CDCL solvers: " *A good learning scheme should reduce the number of decisions needed to solve certain problems as much as possible*." [Zhang and Madigan, 2001]. In the previous section, it was however demonstrated for the first time how crucial this ability is for CDCL solvers.

3.1 Some known results on CDCL behavior

Let us first recall some high level principles of CDCL solvers, by firstly focusing on their restart policies. In earlier works, restarting was proposed as an efficient way to prevent heavy-tailed phenomena [Gomes *et al.*, 2000]. However, in the last years, restart policies were more and more aggressive, and, now, a typical run of a state-of-the art CDCL solver sees most of its restarts occurring before the 1000th conflict [Huang, 2007; Biere, 2008] (this strategy especially pays when it is associated to phase savings [Pipatsrisawat and Darwiche, 2007]). So, in a few years, the concept of restarting has saw his meaning moving from "restart elsewhere" (trying to reach an easier contradiction elsewhere in the search space) to "restart dependencies order" that just reorder variable dependencies, leading the solver to the same search space, by a different path ([Biere, 2008] has for instance already pointed out this phenomenon).

In addition to the above remark, it is striking to notice how CDCL solvers are particularly efficient on the so-called "industrial" benchmarks. Those benchmarks share a very particular structure: they often contain very small backdoors sets [Williams *et al.*, 2003], which, intuitively, encode inputs of formulas. Most of the remaining variables directly depend on their values. Those dependencies implicitly links sets of variables that depend on the same inputs. During search, those "linked" variables will probably be propagated together again and again, especially with the locality of the search mechanism of CDCL solvers.

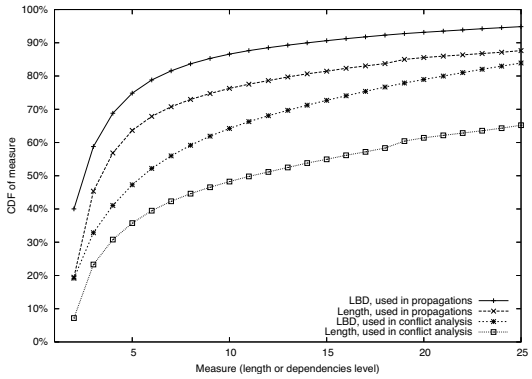

Figure 2: CDF of usefulness of clauses w.r.t. LBD and size.

3.2 Measuring learnt clause quality

During search, each decision is generally followed by a number of unit propagations. All literals from the same level are what we call "blocks" of literals in the later. At the semantic level, there is a chance that they are linked with each other by direct dependencies. Thus, a good learning schema should add explicit links between independent blocks of propagated (or decision) literals. If the solver stays in the same search space, such a clause will probably help reducing the number of next decision levels in the remaining computation.

Definition 1 (Literals Blocks Distance (LBD)) *Given a clause C, and a partition of its literals into n subsets according to the current assignment, s.t. literals are partitioned w.r.t their decision level. The LBD of C is exactly n.*

From a practical point of view, we compute and store the LBD score of each learnt clause when it is produced. This measure is static, even if it is possible (we will see in the later) to update it during search. Intuitively, it is easy to understand the importance of learnt clauses of LBD 2: they only contain one variable of the last decision level (they are FUIP), and, later, this variable will be "glued" with the block of literals propagated above, no matter the size of the clause. We suspect all those clauses to be very important during search, and we give them a special name: "Glue Clauses".

3.3 First results on the literals blocks distance

We first ran MINISAT on the set of all SAT-Race 06[1] benchmarks, with a CPU cut off fixed at 1000s. For each learnt clause, we measured the number of times it was useful in unit-propagation and in conflict analysis. Figure 2 shows the cumulative distribution function of the values over all benchmarks (statistics over unfinished jobs were also gathered). For instance, 40% of the unit propagations on learnt clauses are done on glue clauses, and only 20% are done on clauses of size 2. Half of the learnt clauses used in the resolution mechanism during all conflict analysis have LBD < 6, whereas we need to consider clauses of size smaller than 13 for the same result. Let's look at some details on table 2. What is striking is how the usefulness drops from 1827 times in conflict

[1] Used as a basis for the SAT-Race 08 too.

Measure	2	3	4	5
LBD	202 / 1827	50 / 295	26 / 136	19 / 80
Length	209 / 2452	127 / 884	46 / 305	33 / 195

Table 2: Average number of times a clauses of a given measure was used (propagation/conflict analysis).

analysis for glue clauses to 136 for clauses of LBD 4. We ran some additional experiments to ensure that clauses may have a large size (hundreds of literals) and a very small LBD.

From a theoretical point of view, it is interesting to notice that LBD of FUIP learnt clause is optimal over all other possible UIP learning schemas [Jabbour and Sais, 2008]. If our empirical study of this measure shows its accuracy, this theoretical result will cast a good explanation of the efficiency of First UIP over all other UIP mechanisms (see for instance [Marques-Silva *et al.*, 2009] for a complete survey of UIP mechanisms): FUIP efficiency would then be partly explained by its ability to produce clauses of small LBD.

Property 1 (Optimality of LBD for FUIP Clauses) *Given a conflict graph, any First UIP asserting clause has the smallest LBD value over all other UIPs.*

sketch of proof: This property was proved as an extension of [Audemard *et al.*, 2008]. When analyzing the conflict, no decision level can be deleted by resolution on a variable of the same decision level, because the resolution is done with *reasons* clauses: if a variable is propagated at level l, then the reason clause contains also another variable set at the same level l, otherwise the variable would have been set at a lower decision level $< l$, by unit propagation.

4 Agressive clauses deletion

CDCL solvers performances are tightly related to their clauses database management. Keeping too many clauses will decrease the BCP efficiency, but cleaning out to many ones will break the overall learning benefit. Most effort in the design of modern solvers is put in efficient BCP, leading to a very fast learnt clauses production. Following the success of the VSIDS heuristics, good learnt clauses are identified by their recent usefulness during conflicts analysis, despite the fact that this measure is not a guarantee of its future significance. This is why solvers often let the clauses set grow exponentially. This is a necessary evil in order to prevent good clause to be deleted. On hard instances, this greedy learning scheme deteriorates the core of CDCL solvers: their BCP performance drops down, making some problems yet much harder to solve.

Our aim in this section is to build an aggressive cleaning strategy, which will drastically reduce the learnt clauses database. This is possible especially if the LBD measure introduced above is accurate enough. No matter the size of the initial formula, we remove half of the learnt clauses (asserting clauses are kept) every $20000 + 500 \times x$ conflicts (x is the number of times this action was previously performed).

Table 3 summarizes the performances of four different versions of MINISAT: the original one, the original one with aggressive clause deletion (MINISAT +ag) based on the original clause activity, the classical one based on LBD (MINISAT +lbd) and the classical one with aggressive clause deletion based on LBD scoring (MINISAT +ag+lbd). This comparison is made on the set of 200 benchmarks from the SAT-Race 2006, with a time out of 1000s. We report the number of solved problems and the average time needed to solve them. It is firstly surprising to see that aggressive clause deletion can pay with the original MINISAT. However, less UNSAT benchmarks are solved, despite the overall speed-up dues to fast BCP rates. This may be explained by the deletion of too much *good* learnt clauses, that are essential for efficient UNSAT proofs. The version with LBD solves yet more instances, illustrating its efficiency. What is especially encouraging is the speedup observed between the original MINISAT and the one with our static LBD measure, despite the fact that three less UNSAT benchmarks are solved. Using our very simple and static measure in place of the dynamic one already pays a lot. Finally, the combination of both aggressive clause deletion and LBD is very strong.

	#N (sat-unsat)	#avg time
MINISAT	70 (35 – 35)	209
MINISAT +ag	74 (41 – 33)	194
MINISAT +lbd	79 (**47** – 32)	**145**
MINISAT +ag+lbd	**82** (45 – **37**)	175

Table 3: Comparison of four versions of MINISAT.

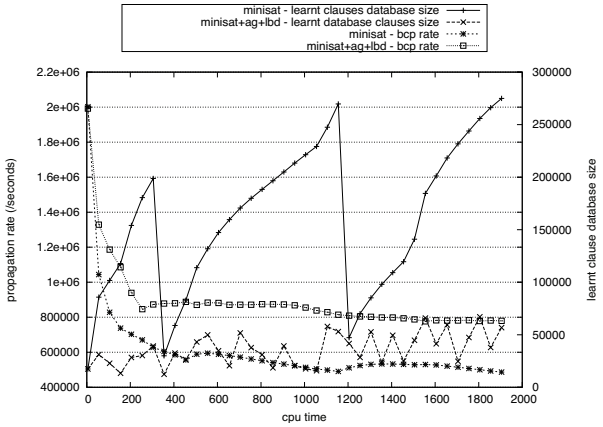

Figure 3: Comparison of BCP rate (left y scale) and learnt clauses database size (right y scale) between MINISAT and MINISAT +ag+lbd.

In order to justify these facts, figure 3 shows, for a selected hard UNSAT formula (with approximatively 50000 variables and 180000 clauses), the evolution of BCP rates and the size of the learnt clauses database. Values are gathered at regular clock time. This figure shows the huge explosion of learnt clauses in the original MINISAT. Only two calls to the reduction of the database are performed in 2000 seconds. In MINISAT +ag+lbd, an important number of database reductions are performed, keeping the BCP rate of MINISAT +ag+lbd much faster than MINISAT.

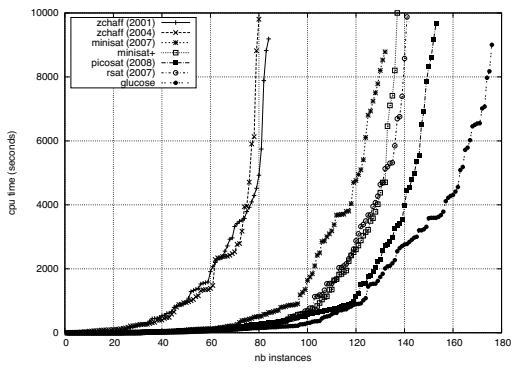

Figure 4: Cactus plot on the 234 industrial instances of the SAT'07 competition.

solver	#N	(SAT-UNSAT)	#U	#B	#S
ZCHAFF 01	84	(47 – 37)	0	13	2.9
ZCHAFF 04	80	(39 – 41)	0	5	3.9
MINISAT+	136	(66 – 74)	0	15	1.5
MINISAT	132	(53 – 79)	1	16	2.1
PICOSAT	153	(**75** – 78)	1	26	1.2
RSAT	139	(63 – 75)	1	14	1.7
GLUCOSE	**176**	(**75** – **101**)	**22**	**68**	-

Table 4: Relative performances of solvers. Column #N reports the number of solved benchmarks with, in parenthesis, the number of SAT and UNSAT instances. Column #U shows the number of times where the solver is the only one to solve an instance, column #B gives the number of times it is the fastest solver, and column #S gives the relative speed-up of GLUCOSE when considering only the subset of common solved instances between GLUCOSE and each solvers (for instance GLUCOSE is 1.7 times faster than RSAT on the subset of benchmarks they both solve).

5 Comparison with the best CDCL solvers

In this final section, we show how our ideas can be embedded in an efficient SAT solver. We used as a basis for it the well-known core of MINISAT using a Luby restarts strategy (starting at 32) with phase savings. We call this solver GLUCOSE for its hability to detect and keep "Glue Clauses". We added two tricks to it. Firstly, each time a learnt clause is used in unit-propagation (it is a reason of a propagated literal), we compute a new LBD score and update it if necessary. Secondly, we explicitly increase the score of variables of the learnt clause that were propagated by a glue clause.

We propose here to compare GLUCOSE with the three winners of the last SAT competition: MINISAT (version 2-070721) [Eén and Sörensson, 2003], RSAT (version 2.02) [Pipatsrisawat and Darwiche, 2007] and PICOSAT (version 846) [Biere, 2008]. We also add in this comparison ZCHAFF [Moskewicz *et al.*, 2001], the first CDCL solver (versions of years 2001 and 2004) and a version of MINISAT including luby restarts (starting at 100) and phase polarity (called MINISAT+ in the rest of the section, as the last – but unreleased – version of it, as it was described). We use the same (234) industrial benchmarks and running characteristics (10000s and 1Gb memory limit) than the SAT'07 competition [Le Berre *et al.*, 2007] in the second stage. According to well admitted idea, we pre-processed instances with SATELITE [Eén and Biere, 2005] and ran all solvers on these pre-processed instances. We used a farm of Xeon 3.2 Ghz with 2 Go RAM for this experiment.

Figure 4 shows the classical "*cactus*" plot used in SAT competitions. X-axis represents the number of instances and Y-axis represents the time needed to solve them if they were ran in parallel. First of all, comparing ZCHAFF and the last winners of the 2007 SAT competition, we can see the gap made during the last seven years. We can also remark that MINISAT, RSAT and MINISAT+ have approximatively the same performances. Before GLUCOSE, PICOSAT was the better solver. We see how GLUCOSE outperforms all of these solvers. It is able to solve 140 problems in 2500 seconds, when currently state-of-the-art solvers need between 4000 and 10000 seconds for this. One of the advantages of this kind of plots is its ability to emphasize when each solver reaches some kind of CPU time limit, when adding more CPU time doesn't add much chance to solve additional problems. For instance, RSAT solves approximatively 100 instances in 1000 seconds but only 40 more in 10000. It is not the case for GLUCOSE, which shows a better scaling up (it solves around 120 instances in 1000 seconds and 60 more in 10000). This is clearly due to our aggressive learnt clause database reduction combined with our clause usefulness measure (BCP performance is maintained and *good* clauses are kept).

Table 4 shows some detailed results. GLUCOSE solves 176 instances, for approximately 140 for RSAT, MINISAT and MINISAT+ (note that GLUCOSE needs only 2500 seconds to solve 140 instances) and 153 for PICOSAT. GLUCOSE especially shows impressive results on UNSAT instances, which really matches our initial motivation, i.e. enhancing the decision levels decreasing. However, we can assume that this strategy is also interesting on SAT instances, since GLUCOSE obtains, with PICOSAT, the best result on these instances. An interesting point is the number of times GLUCOSE is the only solver to solve a given problem (column #U). This shows the power of our method. Finally, the last two columns of table 4 show that GLUCOSE is much faster than all other solvers. It is the fastest solver in 68 cases and performs a speed-up on common instances of at least 1.2.

Before concluding this section, Figure 5 allows us to focus on the relative performances of GLUCOSE and PICOSAT which behaves very well (as shown table 4). The x-axis (resp. y-axis) corresponds to the cpu time tx (resp. ty) obtained by PICOSAT (resp. GLUCOSE). Each dot (tx, ty) matches the same benchmark. Thus, dots below (resp. above) the diagonal indicate that GLUCOSE is faster (resp. slower) than PICOSAT.

In many cases, GLUCOSE outperforms PICOSAT. When PICOSAT wins, GLUCOSE is still close to its performances. On UNSAT instances, and except a few cases, even when GLUCOSE looses, it is close to PICOSAT. Most bad performances are observed on SAT instances only. After a deeper

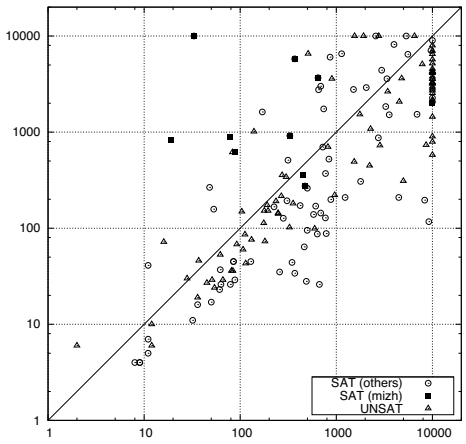

Figure 5: Comparison between GLUCOSE and PICOSAT.

look on those instances, we noticed that seven of those difficult instances are from the mizh family. This result is paradoxically yet again encouraging because, as pointed out in table 1, the evolution of decision levels of CDCL solvers on those instances is increasing, and thus does not match our initial aim: accelerating the decreasing of decision levels (let us recall here that mizh benchmarks are not "industrial" ones, in the classical sense).

In order to come full circle, let us explain now the last column of table 1. This percentage is the relative value of the look-back justification of GLUCOSE by comparison with MINISAT. For instance, on the ibm-2002 instances, the look-back justification of GLUCOSE is 28% smaller than the one of MINISAT. Thus, our overall strategy pays: we accelerated the decreasing of decision levels in most of the cases.

6 Conclusion

In this paper, we introduced a new static measure over learnt clauses quality that enhances the decreasing of decision levels along the computation on both SAT and UNSAT instances. This measure seems to be so accurate that very aggressive learnt clause database management is possible. We expect this new clause measure to have a number of great impacts. We may try to use it to propose preprocessing techniques (adding short LBD clauses), but also more exotic ones, like trying to reorder the formula in order to get a good starting of the decreasing. More classically, we think that our measure will also be very useful in the context of parallel SAT solvers. Indeed, such solver needs to share *good* clauses, which can be now identified with confidence by our measure.

Finally, we can now put efforts in designing an efficient framework for incomplete algorithm for Unsatisfiability. We know what efficient solvers should look for, and the future of our work is clearly on that path. We think that this work opens a new a promising path to answer one of the strongest challenges proposed more than ten years ago, in [Selman *et al.*, 1997], and still wide open.

References

[Audemard *et al.*, 2008] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais. A generalized framework for conflict analysis. In *proceedins of SAT*, pages 21–27, 2008.

[Biere, 2008] A. Biere. PicoSAT essentials. *Journal on Satisfiability*, 4:75–97, 2008.

[Eén and Biere, 2005] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.

[Eén and Sörensson, 2003] N. Eén and N. Sörensson. An extensible SAT-solver. In *proceedings of SAT*, pages 502–518, 2003.

[Gomes *et al.*, 2000] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.

[Huang, 2007] J. Huang. The effect of restarts on the efficiency of clause learning. In *proceedings of IJCAI*, pages 2318–2323, 2007.

[Hutter *et al.*, 2006] F. Hutter, Y. Hamadi, H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *proceedings of CP*, pages 213–228, 2006.

[Jabbour and Sais, 2008] S. Jabbour and L. Sais. personnal communication, February 2008.

[Le Berre *et al.*, 2007] D. Le Berre, O. Roussel, and L. Simon. SAT competition, 2007. http://www.satcompetition.org/.

[Marques-Silva *et al.*, 2009] J. Marques-Silva, I. Lynce, and S. Malik. *Handbook of Satisfiability*, chapter 4. 2009.

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *proceedings of DAC*, pages 530–535, 2001.

[Pipatsrisawat and Darwiche, 2007] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *proceedings of SAT*, pages 294–299, 2007.

[Prasad *et al.*, 2005] M. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *journal on Software Tools for Technology Transfer*, 7(2):156–173, 2005.

[Selman *et al.*, 1997] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *proceedings of IJCAI*, pages 50–54, 1997.

[Williams *et al.*, 2003] R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI'03*, 2003.

[Zhang and Madigan, 2001] L. Zhang and C. Madigan. Efficient conflict driven learning in a boolean satisfiability solver. In *proceedings of ICCAD*, pages 279–285, 2001.