On the community structure of Bounded Model Checking SAT problems*

Guillaume Baud-Berthier[1,3], Jesús Giráldez-Cru[2], and Laurent Simon[1]

[1] LaBRI, UMR 5800, University of Bordeaux, France lsimon@labri.fr
[2] KTH, Royal Institute of Technology, Sweden giraldez@kth.se
[3] SafeRiver, France guillaume.baud-berthier@safe-river.com

Abstract. Following the impressive progress made in the quest for efficient SAT solving in the last years, a number of researches has focused on explaining performances observed on typical application problems. However, until now, tentative explanations were only partial, essentially because the semantic of the original problem was lost in the translation to SAT.
In this work, we study the behavior of so called "modern" SAT solvers under the prism of the first successful application of CDCL solvers, i.e., Bounded Model Checking. We trace the origin of each variable w.r.t. its unrolling depth, and show a surprising relationship between these time steps and the communities found in the CNF encoding. We also show how the VSIDS heuristic, the resolution engine, and the learning mechanism interact with the unrolling steps. Additionally, we show that the Literal Block Distance (LBD), used to identify good learnt clauses, is related to this measure.
Our work shows that communities identify strong dependencies among the variables of different time steps, revealing a structure that arises when unrolling the problem, and which seems to be caught by the LBD measure.

1 Introduction

We observed in the last years an impressive explosion of A.I. and Formal Methods tools using SAT solvers as backbones. The practical interest in SAT solver technologies really took off in the early 2000's when Conflict-Driven Clause Learning (CDCL) algorithms were introduced [24, 14]. This allowed huge SAT instances, encoding application problems, to be solved in practice, where previous *adhoc* methods had failed. One of the first successful application of SAT solvers, Bounded Model Checking (BMC), unrolls a transition system for a given number of steps, quickly producing a huge number of clauses and variables. On this kind of applications, the laziness of data structures is crucial but such an efficiency comes with a price: SAT solvers are now some kind of complex systems for which performances can be hardly explained. We know how to build an *efficient*[4] SAT solver, but the reasons of its efficiency are not clearly known.

* This work was partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013)/ERC grant agreement no. 279611, and by the French National Research Agency (ANR), with the ANR SATAS Project 2015 (ANR-15-CE40-0017-01), and SafeRiver.

[4] CDCL is nowadays the dominant technique solving this kind of problems.

Moreover, it has been shown that SAT formulas, now arising from many different fields, are highly heterogeneous: some problems can have thousands of decisions levels whereas others just have a few dozens. Some problems with millions of variables can be solved in a few seconds whereas others still resist after a few hundreds. If this is due to the diversity of applications using SAT solvers, it makes harder to analyze and explain the reasons of why SAT solvers are good: possible explanations are probably not the same on all problems.

Thus, hidden behind the success story of SAT solvers, the fact that they are complex systems working on a wide range of different problems seems to have prevented any simple explanation for their efficiency to arise. This is a crucial issue for the future of SAT solving, because additional speed-ups will certainly come from a better understanding of all the components of SAT solvers (branching heuristic, learning mechanism, clause database cleaning). This is even more crucial for parallel SAT solving, where simply identifying a good clause to share or how to split the search space between cores is still unclear. Somehow paradoxically, the reasons of most of the improvements in sequential and parallel SAT solving are hardly understood yet. Of course, a number of works proposed some explanations. For instance it was shown that CDCL solvers are stronger than DPLL, unleashing the full power of propositional resolution [23]. More recently, it was shown that other branching heuristics were possible [15]. Closer to our current work, it was also shown that most industrial instances have a *community structure* [1], which could even be used as an estimator of solver success [21]. Moreover, it has been also shown that the measure used for learnt clause usefulness (the Literal Block Distance) is highly related to this notion of clusters, where the literals of the same decision level seem to belong to the same community. In this direction, some models of random SAT formulas with community structure have been presented to better understand SAT solvers components [11–13]. In fact, the community structure has been successfully used in some SAT and MaxSAT approaches [2, 19, 17].

However, none of these works link the structure of the SAT formulas with the original problem they encode. If communities are clearly identified as a key structure in real-world instances, their origin is unknown. The paper we propose aims at answering this crucial question. We propose an experimental paper, aiming at pointing out observed correlations between the high-level description of problems and the behavior of the main components of the SAT solver, thus tracing the origin of communities to the high-level description of the problem. We chose to focus on Bounded Model Checking, because of its historical importance in the rapid development of SAT solver technologies. More precisely, we study the relationship between communities, clauses scoring, variable branching, conflict analysis w.r.t. the unrolling depth of variables.

Contributions of this paper We show that:

- communities are built on small unrolling of time steps, revealing a structure that is not present inside a single depth;
- LBD measures the proximity of literals in the clause, w.r.t. time steps;
- computation tends to produce clauses (proof) at larger and larger time steps;
- the learning mechanism implies touching literals when (1) deciding, (2) analyzing and (3) learning. We show that literals touched in (1), (2) and (3) show clearly

distinct time step extents. Typically, resolution variables (phase 2) belong to more distant time steps than learnt clause literals (phase 3), which belong to more distant than decisions literals.

Let us emphasize here the first item in the above list. It shows that the general idea on the existence of communities in BMC is wrong, or at least only partial. It is indeed believed that communities are simply a side effect of the unrolling mechanism, each community (or set of communities) being simply inside a single time step. Our work shows that variables connections between communities are stronger than previously believed.

2 Preliminaries

We assume the reader familiar with SAT but let us just recall here the global schema of CDCL solvers [24, 14, 7, 18, 9]: a branch is a sequence of decisions (taken accordingly to the VSIDS heuristic), followed by unit propagations, repeated until a conflict is reached. Each decision literal is assigned at a distinct, increasing decision level, with all propagated literals assigned at the same level (we call "block of literals" to the set of literals assigned at the same decision level). Each time a conflict is reached, a series of resolution steps, performed during the conflict analysis, allows the solver to extract a new clause to learn. This clause is then added to the clause database and a *back-jumping* is triggered, forcing the last learnt clause to be unit propagated. Solvers also incorporate other important components such as preprocessing [8], restarts and learnt clause database reduction policies. It was shown in [3] that the strategy based on Literal Block Distance (LBD) was a good way of scoring clauses. The LBD is computed during conflict analysis: it measures the number of distinct decisions levels occurring in the learnt clause. Restarts are commonly following the Luby series [16], but recent studies shown that LBD-based restarts is generally more efficient on real-world instances [4], especially on UNSAT instances [22].

Bounded Model Checking In this paper, we focus on finite-state transition systems with Boolean variables only, which are commonly used to model sequential circuits. A transition system is a tuple $\mathcal{M} = \langle V, I, T, P \rangle$, where V is a set of variables, $I(V)$ and $P(V)$ are formulas over V representing the initial states and safe states, respectively. We also refer to $\neg P(V)$ or $Bad(V)$ to express the set of bad states. $T(V, V')$ is a transition relation over V,V' defining the accepted transitions. $V' = \{v' \mid v \in V\}$ is the primed version of the set V, used to represent next state variables. When multiple transition relations are required, we will use V_i to represent variables in V after i steps.

Bounded Model Checking [5, 26] is an efficient bug-finding algorithm. BMC explores bounded paths of a transition system and checks if they can lead to a bad state. The main idea is to build a formula that is satisfiable if there exists a path of length k from the initial states to a bad state. To this end, BMC *unrolls* the transition relation k times, s.t.:

$$I(V_0) \wedge T(V_0, V_1) \wedge \ldots \wedge T(V_{k-1}, V_k) \wedge Bad(V_k)$$

This formula is usually translated into a CNF and then solved with a SAT solver. If the formula is unsatisfiable, P holds for all states reachable in k iterations. However, this definition does not ensure that Bad cannot be reached in less than k transitions. Hence, BMC$_k$ is usually performed incrementally for $k = 0$ to n. Another approach consists in extending $Bad(V_k)$ to $\bigvee_{i=0}^{k} Bad(V_i)$. If the formula is satisfiable, the transition system has a *counterexample*, i.e. there exists a path in the transition system leading to a state that contradicts the property P.

Community structure of SAT instances in CNF A graph is said to have community structure (or good *clustering*), if in a partition of its nodes into communities, most edges connect nodes of the same community. In order to analyze the quality of such partition, community structure is usually analyzed via scoring functions. The most popular one is *modularity* [10, 20].

Recently, it has been shown that most industrial SAT instances used in SAT competitions have a clear community structure (or high modularity) [1], when the CNF formula is represented as a graph. In this paper, we use the same approach of [1] to analyze the community structure of CNF formulas. This is: creating the Variable-Incidence Graph[5] (VIG) of the CNF, and using the Louvain method [6] to compute a lower-bound of the maximal modularity Q, which also returns its associated partition P of the graph, i.e., a partition of the Boolean variables of the CNF. Since we analyze the relation between variables in the high-level BMC encoding and the low-level CNF formula, we consider VIG as the most suitable graph representation of the formula.

We also emphasize that, although we use an approximate method to compute the community structure, our conclusions do not seem to be a consequence of computing a *wrong* partition (much different to the optimal one), as we will show in the next section.

3 Communities and Unrolling Depth in SAT encodings

The origins of community structure in industrial SAT formulas remain unknown. In previous works, the heterogeneous set of industrial benchmarks used in the competitions has been analyzed as a whole, regardless of where they come from or which problems they encode. Interestingly enough, it has been shown that (clear) community structure is a property in most of these instances. In this section, we provide an exhaustive analysis of the relations between the community structure and the high-level structure of the problem, on the case study of BMC problems.

For a given problem, we can generate different CNF formulas representing BMC$_k$, for different values of k. For these formulas, each Boolean variable belongs to a certain unrolling depth x,[6] and it is also assigned to a certain community y by the clustering algorithm. Notice that it is possible that two (or more) variables are characterized by

[5] In this model, the variables of the CNF are the nodes of the graph, and there is an edge between two variables if they appear together in a clause. In its weighted version –the one we use–, the clause size is also considered.

[6] For simplicity, we omit the very few variables which do not belong to any iteration.

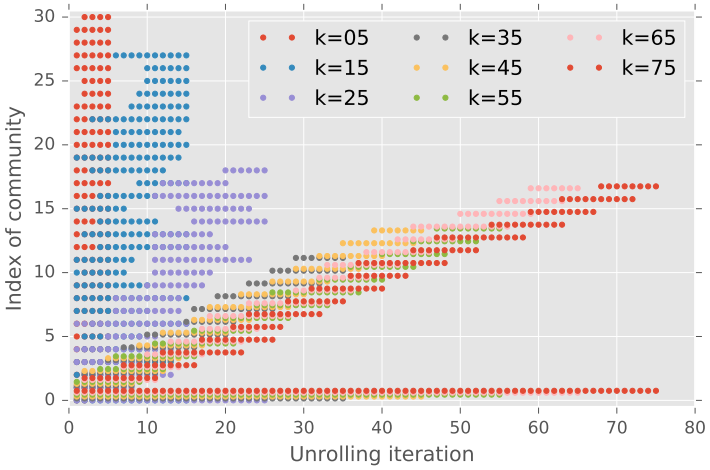

Fig. 1. Relation between unrolling iterations and community structure in the instance `6s7`, for different number of unrolling timesteps k.

the same (x, y) coordinates, when they both belong to the same unrolling iteration and they both are assigned to the same community.

In Figure 1, we represent the relation between the unrolling iterations and the community structure of the instances BMC_k encoding the problem `6s7`, for different values of the unrolling depth $k = \{5, 15, 25, 35, 45, 55, 65, 75\}$. This is, the iteration x and the community y of the variables of each CNF. The y-value for some k are slightly shifted to improve the visualization.

We can observe that when the total number of unrolling iterations is small (see $k = 5$), the community structure is not related to the unrolling depth, since most communities have variables in all depths. The number of communities is greater than k. Interestingly enough, as we increase the total number of iterations k, there is a clear pattern, suggesting that the correlation may be very strong (see cases $k \geq 45$).

However, each point represented in the previous plot can be just due to a single variable. Therefore, it is hard to say if the unrolling iterations and the community structure are indeed correlated. In order to solve this problem, we can compute the Pearson's correlation coefficient r between these two variables X and Y over all these (x, y) points. The correlation coefficient is defined as:

$$r_{X,Y} = \frac{\sigma_{XY}}{\sigma_X \, \sigma_Y} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

where n is the size of the sample with datasets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$, $\bar{x}$, σ_X, $\bar{y}$ and σ_Y their means and standard deviations, respectively, and σ_{XY} the covariance. Notice that when r is close to 1 (resp. to 0), the variables X and Y are highly (resp. almost no) correlated.

Table 1. Statistics of the correlation coefficient r between unrolling iterations and community structure over the instances of the HWMCC15, for different k.

k	mean	std	median	max	$P_{5\%}$	min
5	0.594	0.304	0.667	0.995	0.055	0.000
10	0.677	0.304	0.808	0.997	0.108	0.023
20	0.856	0.170	0.918	0.999	0.492	0.108
40	0.892	0.155	0.956	0.999	0.580	0.109
60	0.904	0.152	0.973	0.999	0.643	0.131

For the instance analyzed in Figure 1, we obtain the following results: $r = 0.423$ when $k = 5$, $r = 0.769$ when $k = 25$, and $r = 0.968$ when $k = 45$. These results confirm that the community structure is actually a good proxy to represent the unrolling depths of this problem.

Now the question is whether this occurs in all BMC problems. To answer this question, we analyze the 513 problems of the Hardware Model Checking 2015 competition (HWMCC15). For each of these benchmarks, we create different BMC$_k$, with $k = \{5, 10, 20, 40, 60\}$, compute the community structure of those, and finally calculate the correlation coefficient r as before. In this analysis, we have omitted those benchmarks for which the computation of the community structure requires more than 5000 seconds (notice that some problems become extremely large after a big number of unrolling iterations). Even though, the resulting sets of instances always contain more than 400 benchmarks; the size of the set varies for each value of k, from 400 instances when $k = 60$, to 509 instances when $k = 5$.

In Table 3, we represent the aggregated results of this analysis. In particular, for each depth k, we compute mean, standard deviation (std), median, max, min, and percentile 5% ($P_{5\%}$), over the r coefficients of all these instances.

It is clear that, in general, the correlation between the unrolling depth and the community structure is very strong (high mean and median, with small deviation). In fact, it becomes stronger for bigger values of k (as we expected from the previous experiment). We can also observe that such correlation is surprisingly small for some instances (very small min). However, it is only the case for a very reduced number of problems, as the percentile 5% indicates. *This analysis strongly suggests that the community structure is originated by the process of unrolling depth iterations to create the BMC formula.*

Additionally, an interesting observation depicted in Figure 1 is that communities computed by the clustering algorithm contain variables from different unrolling depths, rather than just aggregating all variables of the same depth. Therefore, the community structure reveals a non-trivial structure existing in the CNF encoding: *Our analysis also suggests that communities are spread over successive time steps.*

3.1 Communities are stronger when spreading over time steps

Our last observation above is important enough to be checked in detail. It goes against the general belief about the existence of communities in BMC problems. We thus pro-

Table 2. Statistics of the actual modularity Q and the modularity Q_d using as partition the variables of the same unrolling depth, over the instances of the HWMCC15, for different values of depth k. The highest values between $\bar{Q}$ and $\bar{Q}_d$ are highlighted, for each depth k.

	Q						Q_d					
k	mean	std	median	min	max	$P_{5\%}$	mean	std	median	min	max	$P_{5\%}$
5	**0.850**	0.053	0.857	0.681	0.970	0.747	0.616	0.083	0.607	0.480	0.826	0.387
10	**0.874**	0.047	0.877	0.701	0.977	0.786	0.687	0.085	0.676	0.549	0.901	0.450
20	**0.887**	0.043	0.894	0.731	0.981	0.813	0.728	0.086	0.717	0.588	0.945	0.488
40	**0.906**	0.038	0.907	0.759	0.986	0.837	0.751	0.086	0.739	0.609	0.968	0.509
60	**0.917**	0.036	0.917	0.776	0.988	0.856	0.761	0.094	0.767	0.614	0.976	0.516

pose here to search for additional evidences validating it. We also need to check whether this is a property shared by all the BMC problems we gathered.

For this purpose, we consider the partition of the formula in which all the variables of the same unrolling depth conform a distinct community, and compute its associated modularity Q_d. In order to check that our results are not a consequence of computing a *wrong* partition, much different to the optimal one, we compare Q_d w.r.t. the modularity Q computed by the clustering algorithm. In Table 3.1, we represent the aggregated results of Q and Q_d over the set of benchmarks of the HWMCC15. Again, we represent some statistics for different values of k.

As expected, we observe that the modularities Q and Q_d increase for bigger values of the unrolling depth k. Interestingly, the actual modularity of the formula Q is clearly greater than Q_d, independently of the number of iterations k the problem is unrolled. In fact, in many instances, the number of communities is smaller than the number of unrolling iterations k, when k is big enough (e.g., $k = 60$). Another interesting observation is that Q_d fluctuates less than the correlation coefficient r (see Table 3).

These observations support our hypothesis. Therefore, this analysis indicates that the existence of community structure in BMC problems encoded as SAT instances is due to the process of unrolling iterations in the high-level problem, and the larger the unrolling depth is, the more clear the community structure becomes. However, *this structure seems to identify strong dependencies among the Boolean variables in the CNF encoding, which is different from just the unrolling depths they belong to.*

4 Unrolling Depth of Decisions, Resolutions and Learnt Clauses

In the previous section, we have studied the static structure of SAT formulas w.r.t. the unrolling depths of propositional variables. We now propose to analyze where the solver performs the search in these formulas. In particular, we show that, although the solver is not aware about the semantics of each variable (i.e., the unrolling iteration it belongs to), it is able to exploit some features of the BMC encoding. This is probably due to the relation between the high-level BMC encoding and the CNF formula, captured by the community structure.

Our experimental investigations will be based on a set of 106 BMC instances. This set of formulas contains all the problems of the HWMCC15, unrolled until obtaining

a satisfiable answer or not solved within a timeout of 6600 seconds (for the largest possible depth). We excluded the easy instances, solved in less than a minute. We base our experimental observations on an instrumented version of the solver `Glucose`.

For our analysis, we compute three sets of variables that characterize the solver at each conflict. First, the set dec contains all the decision variables, i.e., all the decisions stored in the trail of the solver. Some decisions may not be related to the current conflict, but we think that observing the evolution of dec may be a better trace of the solver search (for instance, top decision variables may be useful in conflicts). Second, the set res of variables used in all the resolution steps, performed during conflict analysis. And finally, the set $learnt$ of literals in the learnt clause. One may notice here that resolution variables (res) are disjoint from decision and learnt variables ($dec \cup learnt$). In some sense, dec will be the witness of the VSIDS heuristics (even if dec has a longer memory of variables decisions, because a variable with a low VSIDS score can still be in the dec if it was chosen at the top of the search tree and its score has been decreased). Moreover, one may notice that the VSIDS heuristic bumps all the variables in the set $res \cup learnt$. It is thus very surprising to find different distributions of values in these 3 sets of variables, as we will show in Section 4.3.

For each of these sets, and for every conflict, we store the set size, and some statistics about the unrolling depths in the high-level problem. Namely, they are: min, max, mean, standard deviation (std), median, median absolute deviation (mad) and skewness. Clearly enough, on hard problems involving millions of conflicts, we cannot simply summarize all these data. Thus, we decided to take samples of 10,000 values and only report summaries of the above values for each sample. Every 10,000 conflicts, we aggregate the results by computing the same values (min, max, et cetera) for each measure (we thus recorded, for example, the median of min, the skewness of skewness, et cetera). After investigating all these data, we found that working on the mean of means for each set dec, res and $learnt$ is sufficient to draw some conclusions; it is a good estimator of the measured values in the sample and it is easily understood. We thus report in this paper the mean values (computed over 10,000 conflicts) of the average number of unrolling depths found in each set dec, res and $learnt$. Note that we recorded this for SAT and UNSAT instances, and also for instances that timed-out, where final statistics were printed before exiting. We found 7 SAT instances, 61 UNSAT ones and 39 time-outs. We cumulated more than 1.2 billion conflicts, and 3.8 millions samples averaging each measure over 10,000 conflicts.

4.1 Relation between solver progress and unrolling depths

In our first experiment, we want to investigate whether the solver is first working on the variables of certain unrolling depths before moving to variables of other depths. For this, we compare the values of dec, res and $learnt$ during the first conflicts w.r.t. to these values measured during the last conflicts. In Figure 2, we represent this comparison for the set dec. The "first conflicts" are computed between the first 10,000 and the 20,000 conflicts (thus, avoiding any possible initialization phase in the very early conflicts, that could introduce a bias), and the value of the last conflicts is computed between the last 10,000+X conflicts, where X is the total number of conflicts modulo 10,000. Results are normalized w.r.t. the total number of unrolling iterations of each SAT instances. A

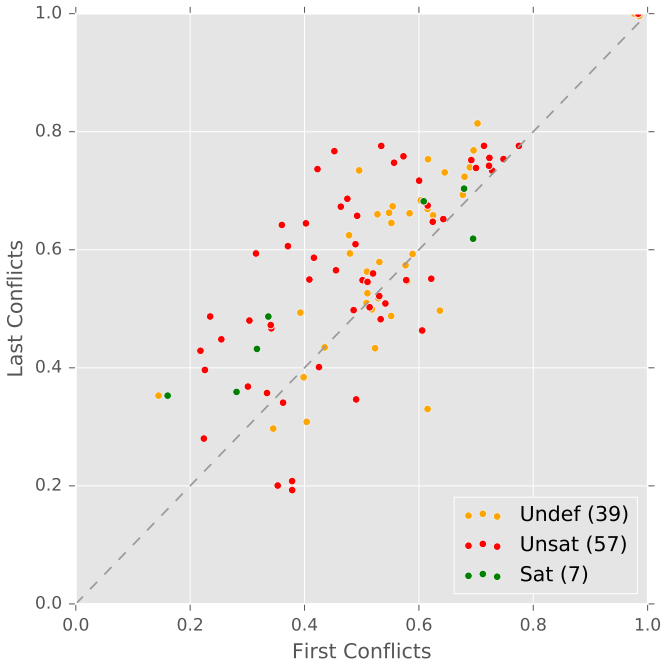

Fig. 2. Unrolling depth of the decision variables during the first conflict versus the last conflicts. Values are normalized w.r.t. the total depth of each BMC SAT formula.

value close to 1 means that the set dec contains variables of the *last* unrolling iterations, whilst a value close to 0 means that it contains the ones of the first iterations.

As we can see, the unrolling iterations involved in the early conflicts are smaller than the ones at the latest stages of the search. This is the case in most of the formulas analyzed, regardless of the answer of the solver. Interestingly enough, this also happens in non-solved instances, indicating a common tendency which suggests that the solver tends to start the search by the first unrolling depths, and continues it by exploring variables of higher depths. We have only reported the unrolling depths of the decision variables (dec). However, the same kind of increasing tendency also occurs for the resolving variables (res), and the variables in the learnt clause ($learnt$), with very similar scatter plots.

As a conclusion, we show here a general behavior of CDCL solvers over BMC benchmarks: all the efforts of the solver (variable decisions, clause analysis / resolution, clause learning) is moving to larger unrolling dephts along the computation. This results cast a new light on previous observations [25] made on the evolution of the centrality of variables in different parts of the solver engine in the general case.

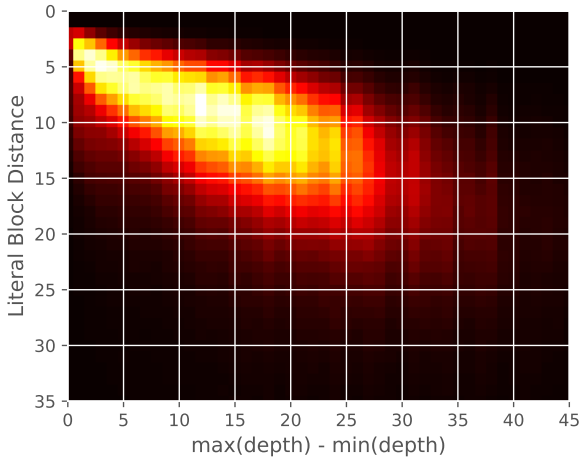

Fig. 3. Heatmap showing the correlation between LBD values of clauses and the $(max - min)$ value of the unrolling depths of its variables. The Pearson Correlation is only 0.315 on this example, showing that even small values are already good indicators for a good correlation on our set of problems.

4.2 Unrolling depth and Literal Block Distance

In [21], it was shown that the LBD measure correlates with the number of communities in SAT instances, in a majority of the cases. Here, we want to know how the LBD measure correlates with the unrolling depth of the variables of BMC formulas. The Figure 1 shown in Section 3 suggests that communities are defined on a small number of successive unrolling iterations. We have tested a large number of hypotheses about the correlation between the LBD of a clause and the information of the high-level problem its variables encode (i.e, the unrolling depth they belong to). Interestingly, the highest correlation we found relates LBD with the $(max-min)$ measure. This is, the maximum depth minus the minimum depth of the variables of each learnt clause. We can represent this relation using a *heatmap* for every problem. In Figure 3, we represent one of the most striking ones. There is clearly a relationship between these two values.

However, such a strong visual relationship does not occur in all formulas, even if we observed it in the immense majority of the cases. In order to summarize all the results, we decided to compute, for each problem, the Pearson correlation coefficient r between the LBD and the value of $(max - min)$ of all learnt clauses. Of course this correlation will not be perfect: the Pearson coefficient measures the correlation between two lines, and we immediately see in Figure 3 that, even if we observe a clear tendency, the cloud of points are dispersed around a line. The Pearson coefficient here will just indicates the general tendency of the cloud, and we typically cannot expect values greater than 0.5 on this kind of clouds. In Figure 4, we represent the cumulative distribution function of r. We also report on the same Figure the CDF of the Pearson

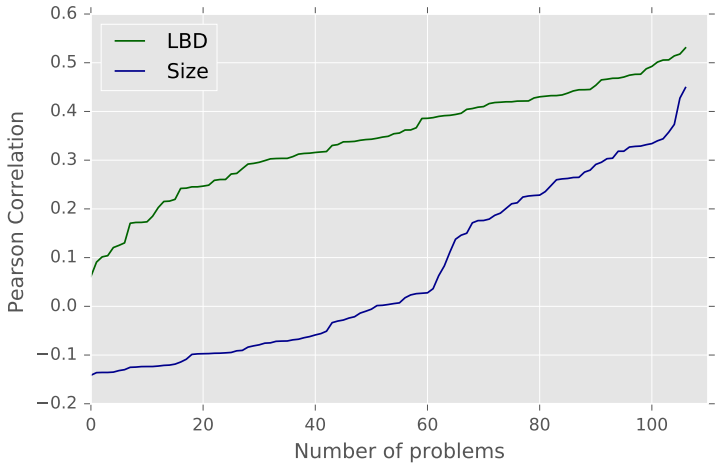

Fig. 4. CDF of Pearson Correlation Coefficient of LBD against $(max - min)$ of unrolling depth. We also show the Pearson Coefficient for the sizes of clauses instead of LBD.

coefficients using sizes of clauses instead of LBD. Our results show that, in most of the cases, the Pearson coefficient is sufficiently high to indicates that these two values are related in most of the cases. However, this correlation could be a simple artifact due to the length of the learnt clause. In particular, a larger clause can have, in general, a larger number of different decision levels, and as a consquence, a larger number of different unrolling iterations, thus increasing the $(max - min)$ measure as a side effect. We test this hypothesis, and show the results on the same Figure 4. It can be observed that the CDF of the Pearson correlation coefficient for the clause size (instead of the LBD) shows a much weaker correlation, suggesting that the LBD correlation is not due to a simple syntactical artifact. These results suggest that LBD is indeed a good metric, which is able to capture an existing structure of the high-level problem which possibly makes the solver to exploit it.

As a conclusion, we show that, in the majority of the cases, the LBD measure is related to the max-min unrolling depth of clauses.

4.3 On the Relation between decisions, resolutions and learning during solver search

In this section, we report an interesting and surprising phenomenon we observed, for which, unfortunately, we do not currently have a final explanation.

Let us now focus on the $(max - min)$ value of unrolling iterations for the set of decision variables (dec), resolutions variables (res), and variables in the learnt clause ($learnt$). Based on our observations, we conjecture now that dec has the smallest $(max - min)$ value, whilst res has the highest one. This is:

$$(max - min)_{dec} \leq (max - min)_{learnt} \leq (max - min)_{res}$$

Fig. 5. CDF of the percentage of samples s.t. $(max - min)_{dec} \leq (max - min)_{learnt} \leq (max - min)_{res}$ (left Y-axis). We also report the CDF of the number of samples per problem (right Y-axis).

In Figure 5, we represent the percentage of samples for which this relation holds. Notice that for each formula, we have to analyze as many samples as learnt clauses. Thus, for reducing the computational effort, we aggregated some data to reduce the number of samples to treat, by taking their average every 10,000 conflicts.

It can be observed that the previous relation occurs in an immense majority of the analyzed samples. *Decisions variables links fewer unrolling iterations than the variables in the learnt clauses, which relate less unrolling iterations than resolution variables.* The CDF plot is clearly showing that this hypothesis holds for almost all the samples we measured. This result is also surprising because variables from dec are chosen thanks to VSIDS, which bumps variables from res and $learnt$. Measuring a significant difference between these sets indicates that a strong mechanism is at work.

This phenomenon is possibly due to variable dependencies w.r.t unit propagations, which tends to imply variables of smaller/larger unrolling iterations as the solver goes deeper in the search tree. However, we have not been able to simply link this to the high-level BMC problem. We also make the hypothesis that if the analysis is done on a greater number of unrolling depths, only a restricted set of iterations is bumped more often, allowing the solver to focus on a restricted set of iterations only. This means that, even if unit propagation tends to propagate variables in the next unrolling iterations (which is a reasonable hypothesis), only a few variables are more often bumped. These variables tends to be localized near the top decision variables of the current search tree.

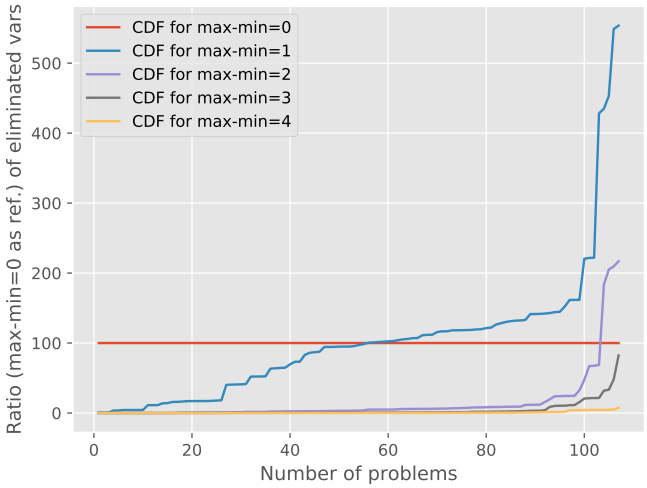

Fig. 6. CDF of the percentage of eliminated variables within the same depth, at distance 1, 2, ... The number of eliminated variable within the same depth is taken as a reference for scaling all the problems.

4.4 On the relation between variable unrolling depths and variable eliminations

The classical preprocessing used in Minisat, and hence in Glucose as well, is essentially built on top of the Variable Elimination (VE) principle [8]. Variable Elimination is crucial for many SAT problems, and particularly for BMC ones. On a typical BMC problem, hundreds of thousands variables can be eliminated. More precisely, the preprocessor orders the variables according to the (current) product of their positive and negative occurrences in the formula, thus trying to eliminate variables that will limit the combinatorial explosion first. Then, a variable is eliminated only if it does not increase the formula size too much (i.e., no more clause after the elimination, and no clause larger than a constant in the resolvents). We want to check here where the preprocessing is working. Our initial hypothesis is that in most of the cases, variables are eliminated inside a single unrolling depth. We thus build the following experimentation. We measure, for each eliminated variable, the maximal depth of the variables occurring in the set of all produced clauses (by the cross product) minus their minimal depth. Thus, a $(max - min)$ value 0 means that eliminating a variable does not add any link between two unrolled depths.

In Figure 6, we represent the results. This plot can be read as follows. On approximately 50 problems (where the blue curve representing $(max - min) = 1$ crosses the red curve representing $(max - min) = 0$) there are more eliminated variables inside a single unrolling iteration than connecting two of them. We can see that on almost all the problems, there are only very few variable eliminations that involve a large

$(max - min)$ value of depths. It is however surprising to see that some problems involve the elimination of a larger set of variables not strictly within the same community (i.e., when the blue curve is above the red line). We think that an interesting mechanism may be at work here and further investigations on the role of these variables may be important. However, *in general, the elimination of the variable chosen by the classical preprocessing heuristic only involves clauses of very limited* $(max - min)$ *values*, as expected.

5 Discussions about improving SAT solvers

We strongly believe that a better understanding of what CDCL solvers are doing, and how they are doing that is needed by the community for improving SAT solving. Given the observations we made in this paper, we naturally tried a number of Glucose hacks in order to try to exploit the unrolling iterations of variables. However, until now, our attempts were at least partially unsuccessful.

5.1 Shifting variables scores

We tried, at each restart triggered by the solver, to force it to branch more deeper (or shallower) in the formula. For this, during the first descent right after a restart, we modified the variable picking heuristics as follows: each time a variable was picked, we took the same variable (1) one time step after, (2) on the last time step, (3) one time step before, (4) on the first time step. What we observed is that only the hack (2) was competitive with the original Glucose (it solved the same amount of problems). All the other versions were degrading the performances.

5.2 Scoring Clauses w.r.t. max-min of unrolling iterations

Of course, because of the clear relationship of LBD and the $(max - min)$ of unrolling iterations of variables, we tried to change Glucose clause scoring scheme. Instead of the LBD, we simply used this $(max - min)$ metric. We however observed a small degradation of its performance. Further experiments are needed here, essentially because the LBD mechanism is important in Glucose for scoring the clauses, but also for triggering restarts.

5.3 Forcing variable elimination according to their unrolling iterations

We also tried to change the order of variable eliminations during the preprocessing step [8]. In our attempt, we first eliminate all the variables from within a single time step: when the cross product does not produce any clause with variable from different unrolling iterations. Then, variable elimination is allowed to produce clauses containing variables from two successive time steps. Then 3, 4, ... time steps. We observed, as expected, that almost all the variable eliminations are done during the first round (all clauses are generated within the same time step), even if other variables are eliminated in the other rounds. From a performance point of view, these versions show a slightly degradation of performance.

5.4 Further possible ways of improvements

Despite the disappointing experimental improvements we reported above, we strongly believe that our work can lead to a significant improvement of SAT solvers over BMC problems. Our work advocates for a better specialization of SAT solvers. For instance, it may probably be possible to use the notion of variables time steps to estimate the progress of the solver along its computation. We are also thinking of using this knowledge for a better parallelization of SAT solvers, for instance, by splitting the search on deeper variables only.

6 Conclusions

Despite the empirical success of SAT solvers, they lack from a simple explanation supporting the observed performances. One intriguing question is why they are more efficient, at the CNF level, than *adhoc* approaches that can access to higher level semantic information. In our study, we focus on the historical success application of SAT solvers, i.e., Bounded Model Checking. We show that community structures, identified a few years ago in most of the SAT formulas encoding real-world problems, are not a trivial artifact due to the circuit unrolling when encoding BMC problems. Our work is an important effort for a SAT solver experimental study. We report evidences that the community structure identified in previous works, and well captured by current SAT solvers, is unveiling a structure that arise when unrolling the circuit only (on our set of BMC problems). This is an interesting finding, that answers open questions about the origin of communities. We also show that the proof built by the SAT solver, approximated in our study by the set of all learnt clauses is evolving along the computation, producing clauses of higher unrolling depths at the latest stages of the search, rather than at the beginning of the execution.

We plan to extend our work in many ways. First, we would like to study the semantic of SAT formulas in other domains, like CSP benchmarks encoding planning or scheduling problems, or cryptographic formulas. In all these cases, we think that working on benchmark structures could help SAT solver designers to specialize their solvers to some important applications. We could also expect that CSP solvers could benefit from an expertise of how SAT solvers are working on the underlying SAT formula representing such CSP problems. In particular, this may be useful to refine the heuristic used to solve such instances. Second, we would like to test whether SAT solvers could benefit from the unrolling depth information for BMC problems. We particularly think of studying the unsatisfiable proof in order to work on a better parallelization of SAT solvers *specialized* for BMC problems.

References

1. Ansótegui, C., Giráldez-Cru, J., Levy, J.: The community structure of SAT formulas. In: Proc. of SAT'12. pp. 410–423 (2012)
2. Ansótegui, C., Giráldez-Cru, J., Levy, J., Simon, L.: Using community structure to detect relevant learnt clauses. In: Proc. of SAT'15. pp. 238–254 (2015)

3. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern sat solvers. In: Proc. of IJCAI'99. pp. 399–404 (2009)
4. Biere, A.: Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016. In: Proc. of the SAT Competition 2016. Department of Computer Science Series of Publications B, vol. B-2016-1, pp. 44–45. University of Helsinki (2016)
5. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Proc. of TACAS'99. pp. 193–207 (1999)
6. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10), P10008 (2008)
7. Darwiche, A., Pipatsrisawat, K.: Complete Algorithms, chap. 3, pp. 99–130. IOS Press (2009)
8. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proc. of SAT'05. pp. 61–75 (2005)
9. Een, N., Sörensson, N.: An extensible SAT-solver. In: Proc. of SAT'03. pp. 502–518 (2003)
10. Fortunato, S.: Community detection in graphs. Physics Reports 486(3-5), 75 – 174 (2010)
11. Giráldez-Cru, J., Levy, J.: A modularity-based random SAT instances generator. In: Proc. of IJCAI'15. pp. 1952–1958 (2015)
12. Giráldez-Cru, J., Levy, J.: Generating SAT instances with community structure. Artificial Intelligence 238, 119–134 (2016)
13. Giráldez-Cru, J., Levy, J.: Locality in random SAT instances. In: Proc. of IJCAI'17 (2017)
14. Gomes, C.P., Selman, B., Crato, N.: Heavy-tailed distributions in combinatorial search. In: Proc. of CP'97. pp. 121–135 (1997)
15. Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: Proc. of SAT'16. pp. 123–140 (2016)
16. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Inf. Process. Lett. 47(4), 173–180 (1993)
17. Martins, R., Manquinho, V.M., Lynce, I.: Community-based partitioning for MaxSAT solving. In: Proc. of SAT'13. pp. 182–191 (2013)
18. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. of DAC'01. pp. 530–535 (2001)
19. Neves, M., Martins, R., Janota, M., Lynce, I., Manquinho, V.M.: Exploiting resolution-based representations for MaxSAT solving. In: Proc. of SAT'15. pp. 272–286 (2015)
20. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69(2), 026113 (2004)
21. Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G., Simon, L.: Impact of community structure on sat solver performance. In: Proc. of SAT'14. pp. 252–268 (2014)
22. Oh, C.: Between SAT and UNSAT: The fundamental difference in CDCL SAT. In: Proc. of SAT'15. pp. 307–323 (2015)
23. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artificial Intelligence 175(2), 512–525 (2011)
24. Silva, J.P.M., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Trans. Computers 48(5), 506–521 (1999)
25. Simon, L., Katsirelos, G.: Eigenvector centrality in industrial sat instances. In: Proc. of CP'12. pp. 348–356 (2012)
26. Strichman, O.: Accelerating bounded model checking of safety properties. Formal Methods in System Design 24(1), 5–24 (2004)