Impact of Community Structure on SAT Solver Performance

Zack Newsham[1], Vijay Ganesh[1],
Sebastian Fischmeister[1], Gilles Audemard[2], and Laurent Simon[3]

[1] University of Waterloo, Waterloo, Ontario, Canada
[2] Laboratoire Bordelais de Recherche en Informatique, Bordeaux Cedex, France
[3] Université Lille-Nord de France, CRIL - CNRS UMR 8188, Artois, F-62307 Lens

Abstract. Modern CDCL SAT solvers routinely solve very large industrial SAT instances in relatively short periods of time. It is clear that these solvers somehow exploit the structure of real-world instances. However, to-date there have been few results that precisely characterise this structure. In this paper, we provide evidence that the community structure of real-world SAT instances is correlated with the running time of CDCL SAT solvers. It has been known for some time that real-world SAT instances, viewed as graphs, have natural *communities* in them. A community is a sub-graph of the graph of a SAT instance, such that this sub-graph has more internal edges than outgoing to the rest of the graph. The community structure of a graph is often characterised by a quality metric called Q. Intuitively, a graph with high-quality community structure (high Q) is *easily separable* into smaller communities, while the one with low Q is not. We provide three results based on empirical data which show that community structure of real-world industrial instances is a better predictor of the running time of CDCL solvers than other commonly considered factors such as variables and clauses. First, we show that there is a strong correlation between the Q value and Literal Block Distance metric of quality of conflict clauses used in clause-deletion policies in Glucose-like solvers. Second, using regression analysis, we show that the the number of communities and the Q value of the graph of real-world SAT instances is more predictive of the running time of CDCL solvers than traditional metrics like number of variables or clauses. Finally, we show that randomly-generated SAT instances with $0.05 \leq Q \leq 0.13$ are dramatically harder to solve for CDCL solvers than otherwise.

1 Introduction

In the last few years, we have witnessed impressive improvements in the performance of conflict-driven clause-learning (CDCL) Boolean SAT solvers over real-world industrial SAT instances, despite the fact that the Boolean satisfiability problem is known to be NP-complete and the worst-case time complexity of our best solvers is exponential in the size of the formula. What is even more impressive is that these solvers perform extremely well even for never-before-seen classes of large industrial instances, where the biggest instances may have upwards of 10 million clauses and millions of variables in them. In other words, one cannot reasonably argue that these solvers are being hand-tuned for every class of real-world instances. It is all but clear that CDCL solvers employ a very general class of techniques, that have been robustly implemented and continuously tested for many applications ranging from software engineering to AI. All of this begs the question why CDCL solvers are so efficient, and whether they are exploiting some structural features of real-world instances. It is this question that we address in this paper.

In this paper, we present three results that show that there is correlation between the presence of natural communities [6] in real-world SAT instances [3, 4] and the running time of MiniSAT CDCL solver [7] (by extension many other CDCL SAT solvers that are either built using MiniSAT code or use the most important techniques employed by CDCL SAT solvers). Informally, a community [6] in a SAT formula, when viewed as the variable-incidence graph [4], is a sub-graph that has more edges internal to itself than going out to the remainder of the graph. There is previous work pointing to some correlation between community structure in SAT instances and performance of CDCL solvers [2]. However, we provide much stronger evidence as discussed in the Contributions sub-section below.

We characterise the structure of SAT instances through a well-known metric called the *Q value* [6] and the number of communities present in its graph. The Q value is a widely-accepted quality metric that measures whether the communities in a graph are *easily separable*. In particular, formulas [5] with high Q tend to have few inter-community edges relative to the number of communities, while those with low Q have lots of inter-community edges.

Contributions:

1. We show that there is a strong correlation between Literal Block Distance (LBD), introduced in a paper [5] by some of the authors on learnt clause quality, and number of communities. This correlation fits better and better as the search progresses. In their original paper [5], the authors suggested that the quality of a learnt clause can be measured using the LBD metric. I.e. the lower the LBD the better the learnt clause. They also suggested a learnt clause deletion policy, wherein clauses with high LBD were marked for deletion. The result we found in this paper suggests that low LBD clauses also are shared by very few communities.
2. We performed a regression analysis of the performance of the MiniSAT [7] solver over SAT 2013 competition instances [1], using a variety of factors that characterise Boolean formulas including number of variables, number of clauses, number of communities, Q and even ratios between some of these

[4] A variable-incidence graph of a Boolean SAT formula is one where the variables of the formula are nodes and there is an edge between two nodes if the corresponding variables occur in the same clause.

[5] In the rest of the paper, we do not distinguish a formula from its variable-incidence graph.

factors. We found that the number of communities and Q were more correlated with the running time of MiniSAT over these instances (real-world, hard combinatorial, and random) than the traditional factors like number of variables, clauses or the clause-variable ratio.

3. Additionally, we generated approximately 500,000 random Boolean formulas and made the surprising finding that MiniSAT finds it hard to solve instances with Q value lying in the range from 0.05 to 0.13, whereas it was able to easily solve the ones outside this range. While previous work [11] has shown that the clause-to-variable ratio is predictive of solver run time on randomly-generated instances (phase transition at clause-to-variable ratio of 4.2 [11]), this metric is not predictive at all of solver efficiency on real-world instances [15]. By contrast, according to our experiments, Q and number of communities measure for both real-world and random instances are correlated with the running time of MiniSAT (and by extension all solvers that are significantly similar to it algorithmically) on these instances.

2 Background

In this Section, we provide some background on regression analysis, the concept of the community structure of graphs and how it relates to SAT formulas.

2.1 Community Structure of SAT Formulas

The idea of decomposing graphs into *natural communities* [6, 17] arose in the study of complex networks such as the *graph of biological systems* or the Internet. Informally, a network or graph is said to have community structure, if the graph can be decomposed into sub-graphs where the sub-graphs have more internal edges than outgoing edges. Each such sub-graph (aka module) is called a community. Modularity is a measure of the quality of the community structure of a graph. The idea behind this measure is that graphs with high modularity have dense connections between nodes within sub-graph but have few inter-module connections. It is easy to see informally that maximising modularity is one way to detect the optimal community structure inherent in a graph. Many algorithms [6, 17] have been proposed to solve the problem of finding an optimal community structure of a graph, the most well-known among them being the one from Girvan and Newman [6]. The quality measure for optimal community structure is often referred to as the Q value, and we will continue to call it similarly. There are many different ways of computing the Q value and we refer the reader to these paper [6, 17, 12].

We experimented with two different algorithms the Clauset-Neuman-Moore (CNM) algorithm [6] and the online community detection algorithm (OL) [17]. While we did find that the CNM algorithm resulted in a better community structure — evidenced by fewer communities with few links between them — we chose the OL algorithm because of its vastly superior run time. This was of particular importance due to the sheer size and number of the SAT instances we processed. Our initial experiments were conducted with an implementation of the CNM algorithm, then repeated with the OL algorithm. The results we present in Section 3.4 were observed, regardless of the choice of algorithm.

2.2 Linear Regression

In this paper we make use of linear regression techniques for the result that correlates the Q value and number of communities with the running time of the MiniSAT CDCL SAT solver. In case the reader is not familiar with this topic, we provide a very brief description of the basic ideas involved.

Given multiple independent factors and a single dependent variable, linear regression can be used to determined the relationship between the factors and the variables based on a provided model. For the scope of this paper the dependent variable will always be $log(time)$, while the independent factors (such as Q value, number of communities, variables, and clauses) will be appropriately specified for each experiment in Section 3.4. This model can either look only at the main effects of the factors specified, or at both the effects of factors and the interactions between them.

We provide a few important definitions below:

ANOVA stands for analyses of variance. In the scope of this paper, it is generated by the linear regression, and used to understand the influence that specific factors and interactions between factors have on the dependent variable.

R^2 represents the amount of variability in the data that has been accounted for by the model and is used to measure the *goodness of fit* of the model. It ranges from zero to one with one representing a perfect model. Due to the nature of the calculation, the R^2 value will increase when additional factors are added to the model. In this paper, we refer to the *Adjusted* R^2 which is modified to only increase if an added factor contributes positively to the model.

Confidence Levels are used to specify a certain level of confidence that a given statement is true. They can be used to calculate the likelihood of a given set of input values resulting in a given output (for example time), or they can be used to estimate the likelihood that a factor in a model is significant. They are measured in percent, typical values are 99.9, 99 and 95. Any result with a confidence level below 95% is considered unimportant in the context of this paper.

Confidence Intervals are used to provide a range for a value at a given confidence level, which is usually set to 95% or 99%. They show that with a given percentage probability, an estimated value will lie between a certain range.

Kolmogorov–Smirnov test is used to provide quantitative assurances that a provided sample belongs to a specified distribution, it results in a value between zero and one, with values approaching zero indicating that the provided sample does belong to the specified distribution.

Residuals is the difference between a fitted dependent variable and the corresponding provided dependent variable. It represents the amount of error for a given set of input factors when calculating the output.

3 Experimental Results

In this Section we describe our experimental results that correlate Q value/the number of communities with the running time of two CDCL solvers we considered in our experiments, namely, MiniSAT [8] and Glucose [5].

3.1 Correlation between LBD and Community Structure

In this section, we propose to link the number of communities contained in a clause with an efficient measure used in recent CDCL (Conflict-Driven Clause-Learning) solvers to score learnt clause usefulness. Interestingly, this measure is used in most of today's best SAT solvers.

We assume that the reader is familiar with the basic concepts and techniques relevant to CDCL SAT solvers. Briefly, these solvers branch by making decisions on the value of literals, and at any step of the search, ensure that all the unit clauses w.r.t the current partial assignment are correctly *propagated* until either an empty clause is found (the input formula is UNSAT) or a complete assignment is found (the input formula is SAT). The important point to be emphasised is that solvers learn clauses at a fast rate (generally around 5000 per seconds), which can overwhelm the memory of the system unless some steps are taken to routinely get rid of some of them. Such learnt clause deletion policies have come to be recognised as crucial to the efficiency of solvers. The trick to the success of such deletion policies is that somehow the solver has to differentiate *good* clauses from the ones that are not so good.

Prior to the 2009 version of the Glucose solver [5], deletion policies were primarily based on the past VSIDS activity of the clauses, i.e., learnt clauses with low VSIDS scores were deleted. However in their paper [5], the authors proposed that a better scoring mechanism for learnt clauses is to rank them by the number of distinct decision levels the variables in these clauses belonged to. This measure is called the Literal Block Distance (LBD) of a clause. The smaller the LBD score of a clause, the higher its rank. The intuition behind this scoring mechanism is the following: The lower the LBD score of a clause, the fewer the number of decision levels needed for this clause to be unit propagated or falsified again during the search. Clauses with LBD score of 2 are called glue clauses, because it allows for merging (glue) of two *blocks of propagations* with the aim of removing one decision variable (this notion in turn inspired the name of the Glucose solver). It is important to note that this behaviour is more likely to happen when using the phase saving heuristic for branching [14], because all variables are set to their last propagated value when possible.

The hypothesis we test in this sub-section is that the notion of *blocks* of propagations (i.e. a decision variable and all the propagated variables by unit propagation at the same decision level) is highly correlated to the idea of communities. To be more precise, if an input instance has high-quality community structure (communities with few inter-community edges) then we hypothesise that the conflict clauses that are shared between fewer communities are likely to cause more propagation per decision and hence are likely to have a lower LBD score. We verify our hypothesis, namely, that there is indeed a strong relationship between the number of communities of a clause (initial or learnt) and its LBD score computed by Glucose.

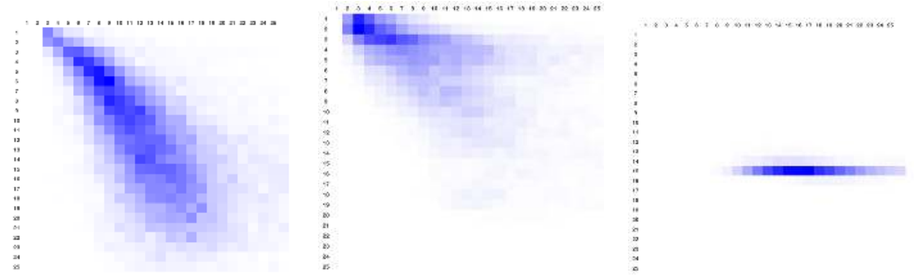
Intuitively, we consider clauses that are shared between very few communities as higher quality than the ones shared between many communities, because such clauses are localised to a small set of communities possibly enabling the solver to in-effect partition the problem into many "small set of communities" and solve them one at a time.

Experiment Set up: We limit our study to the set of industrial instances of the SAT 2013 competition (Applications category). This is in line with our observation that SAT instances obtained from real-world applications have good community structure, and consequently the notion of LBD scoring will likely have the biggest impact on performance for such instances than otherwise. Put it differently, if their is indeed a relationship between LBD and community structure of SAT instances, we hope to characterise it in this set of problems first. For our experiment, we store, for each formula of the 189 instances (SAT'13 Competition, Application Track), the value of the LBD and the number of different communities of each learnt clauses. (There are 300 application instances in the SAT'13, however we were able to compute the communities of only 189 of them due to resource constraints.)

We would like to emphasise few points here: (1) First, all instances were preprocessed using the SatELite simplifier before any experiments were conducted for this study. All CDCL solvers have pre-processing simplifying routines in them. It is very likely that these kind of simplified formulas are representative of the inherent structure of formulas CDCL solvers are efficient on. (2) Second, we computed the community structure using the Newman algorithm [12] (aka CNM algorithm) on the variable-incidence graph representation of the formula. Results were stored in a separate file once for all the experiments. For each SAT instance, the corresponding communities were first labeled. Then, for every instance we maintained a map from variables occurring in that instance to the label of the corresponding community the variable belonged to. (3) Third, Glucose was launched on these instances without any information regarding their community structure computed in step (2). Glucose computed the LBD values for the input instances, and stored them in a large trace file that was analysed later: for each instance, for each learnt clause, we compared the LBD value of that learnt clause and computed, thanks to (2), the number of distinct communities the learnt clause contained. Finally, note that we used a maximum number of conflicts for our study, not a time out. In Section 3.2, the maximum conflicts studied is set to the first 20,000 conflicts. In the Section 3.3, it is set to 100,000. Those values were chosen w.r.t. the statistical tools we used.

3.2 Observing the clear relationship between Communities and LBD by Heatmaps

It is not trivial to express a relationship between thousands of values, following unknown distribution functions, on hundreds of problems. Hence, to show the



(a) dated-5-13-u (138,808 / 97,775 / 0.9) (b) aaai10-planning (50,277 / 12776 / 0.91) (c) rbcl_xits_14 (2,220 / 725 / 0.53)

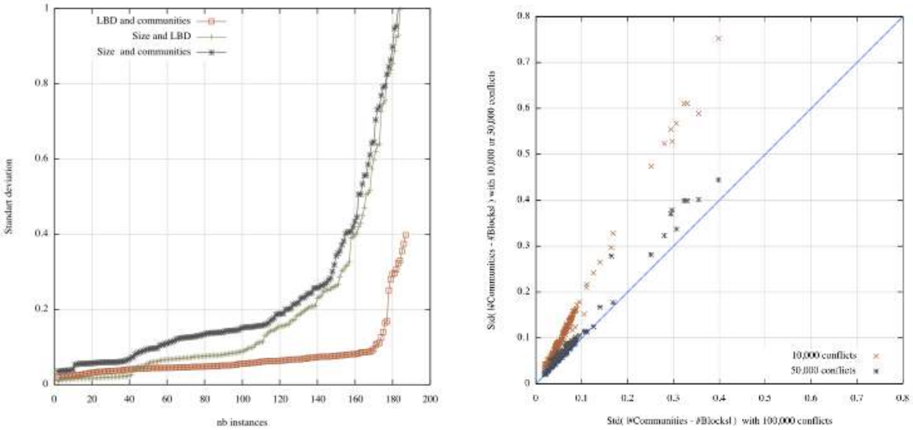
Fig. 1: Relationship between Literal Block Distance (LBD) and communities for a selection of instances. The x-axis corresponds to LBD, the y-axis to communities. Blue intensity represents the frequency of learnt clauses with the considered LBD and community value. For each instance, we provide in parenthesis, the number of variables, the number of communities, and the quality value (Q). The figure is analysed in Section 3.2

general trend, we chose to build one heatmap per instance: we compute the number of occurrences of each couple $(LBD, Community)$ on the considered problem and assign the intensity of a colour with respect to this number of occurrences. The result is shown for some characteristic instances Figure 1. As we can see, there is an obvious and clear relationship on the first two instances (a, b) which are the most frequent cases. Intensive colours follow the diagonal: many clauses have approximatively the same LBD and the same number of communities. This behaviour appears in most cases. All heatmaps are available on the web (a temporary url for the reviewing process available on request), with more or less a strong diagonal shape.

From these figures we can conclude that there has to be a strong relationship between LBD and number of communities. Small LBD learnt clauses add stronger constraints between fewer communities. This may allow the solver to focus its search on a smaller part of the search tree, avoiding *scattering*, the phenomena where the solver jumps between lots of communities creating learnt clauses that link these communities together thus making the structure of the SAT instance worse and consequently harder to solve. We think this study gives a new point of view of LBD and provides a new explanation of its efficiency. Of course, there exists a few cases that do not exhibit such a relationship. This is the case, for example, for the last example provided in the Figure 1.c. In this instance, it seems that in many cases all learnt clauses involve 15 communities and more than ten decision levels.

We do not yet have strong evidence that correlates classes of instance, and the average number of communities that a learnt clause belongs to. Our suspicion is that all the original problems for which CDCL solvers were designed, the BMC problems, may exhibit a particularly good relationship.

3.3 How close are Communities and LBD?



(a) Cumulative distribution function of standard deviation between different measures

(b) Evolution of the standard deviation during the search

Fig. 2: Some standard deviations. This figure is analysed Section 3.3

If the LBD seems heavily related to communities, the question is now how close to the LBD is it? In particular, on some extreme cases, the simple size of a clause could be a good predictor for its number of communities (clearly, the larger the clause is, the bigger the number of communities can be). Thus, we also have to ensure that (1) the LBD score is more accurate than the size of the clause for predicting the number of communities and (2) the more we update the LBD, the closest we are to the number of communities (the solver is consistent along its search).

In answer to the above question we present two figures: 2.a and 2.b. For both figures we computed each problem instance X, for each learnt clause c during the first 100,000 conflicts of Glucose solving X, the standard deviation δ_X of the values:

1. $|LBD(c) - \#Com(c)|$, representing the dispersion of the differences between the LBD of a clause c and its number of communities, shown as "LBD and communities" in the figure;
2. $|size(c) - \#Com(c)|$, representing the dispersion of the differences between the size of a clause c and its number of communities, shown as "Size and communities";
3. $|LBD(c) - size(c)|$, representing the dispersion of the differences between the LBD of a clause c and its size, shown as "Size and LBD";

In the first experiment, we try to see how close LBD and size are to the number of communities. We represent in Figure 2 the cumulative distribution function of all the δ_X for each of the three cases. This figure clearly highlights that the relationship between LBD and number of communities is much more accurate (it is less than 0.1 for a large majority of instances) than the relationship between size and LBD or between size and communities. Thus, as a first conclusion, we see that, in the large majority of the cases, the LBD is really close to the number of communities. The only hypothesis here is that the standard deviation has some meaning over the analysed data, which seems to be a plausible hypothesis.

The last experiment we conducted studies the evolution of LBD scores during the execution of Glucose. We focus now on the values of δ_X accounting for the dispersion of values $|LBD(c) - \#Com(c)|$ for each instance X. We compare the values obtained after 10,000 conflicts with the values obtained after 100,000 (shown as "10,000 conflicts" on the figure) and the values obtained after 50,000 conflicts with, again, the values obtained after 100,000 (shown as "50,000 conflicts"). The comparison is done by the two scatter plots in Figure 2.b. Two conclusions can be drawn from this. It seems clear that the longer the solver is running, the more accurate it is at estimating the number of communities of clauses by the LBD. This may also be explained by the fact that, the longer the solver is running, the longer it is working on fewer communities/LBD, thus focusing on small subparts of the problem. However, which one of these two hypothesis is more accurate is still an ongoing work.

3.4 Experimental Setup: Correlation between Solve Time and Community Structure

In this section, we present the hypothesis that it is possible to correlate the characteristics of SAT instance C and the running time of CDCL SAT solvers in solving C. Previous attempts in this direction have largely focused on characterising the hardness of solving SAT instances in terms of number of variables, clauses or the clause-variable ratio [9]. In our experiments, we go beyond variables, clauses, and their ratio to also considering number of communities and the Q value (modularity). To test this hypothesis we performed two experiments. The first experiment we did was to correlate the above-mentioned characteristics and the running time of the MiniSAT solver over all instances in the SAT 2013 competition [1]. The second experiment we performed was a controlled one, wherein, we randomly-generated instances varying a subset of their characteristics such as number of variables, clauses, Q value, and number of communities and keeping the rest constant. We then ran MiniSAT on these randomly generated instances and recorded its running time. We then plotted the running time against changing Q to see how the two are correlated. All the data for these experiments is available at [13].

3.5 Community Structure and SAT 2013 Competition Instances

We performed the following steps in this experiment. First, we attempted to calculate the community structure of every SAT instance from all categories (hard combinatorial, random, and application) the SAT 2013 competition [1].

For this we used the OL algorithm [17]. Due to the size of some of the formula it was not possible to get this information for every instance. As such we were only able to run the community structure analysis on approximately 800 instances. The generated results were then aggregated with the solve time of the MiniSAT solver (from the SAT 2013 competition website [1]) for each instance.

The analysis was performed on the $log(time)$ rather than raw recorded time due to the presence of a large number of timeouts, which would have a skewed distribution. Having said that, our results are similarly strong without this constraint. In addition to this, we standardised our data to have a mean of zero and a standard deviation of one. This is standard practice when performing regression on factors that have large differences in scale, and ensures that importance is not falsely reported based only on scale.

After formatting the data as described, we fitted a linear regression model to it using a stepwise regression technique to choose the best model, this was identified as:

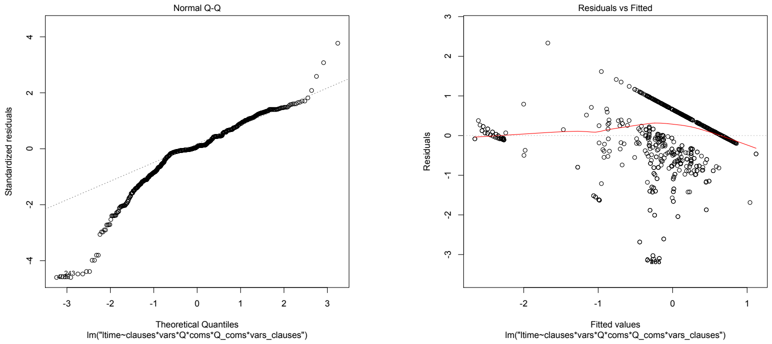
$$log(time) = |V| \oplus |CL| \oplus \mathrm{Q} \oplus |CO| \oplus \mathrm{QCOR} \oplus \mathrm{VCLR}$$

Where V is the set of all variables in a formula, CL is the set of all clauses, CO is the set of all communities, QCOR is the ratio of $\frac{Q}{|CO|}$, and VCLR is the ratio of $\frac{|V|}{|CL|}$. In this model the $\oplus$ operator denotes that the factor and all interactions with all other factors were to be considered. Performing the regression resulted in a residual vs fitted plot, shown in Figure 3(b) where the x-axis shows the fitted values and the y-axis shows the residuals. As well as a normal quantile plot shown in Figure 3(a) where the x-axis shows the standardised residuals plotted against a randomly generated normally distributed sample with the same mean and standard deviation on the y-axis. In the normal quantile plot, the presence of a slight curve in the line is indicative that the distribution of the data may be bimodal, as such we are unable to measure confidence intervals for the accuracy of the model. In addition to this, the residual plot shows that the data may be biased, but at least has relatively even variance. Unfortunately, the presence of the timeout results has played a role in the biased nature of the experiment, however dropping them from the results entirely leads to a bias in the opposite direction.

The adjusted R^2 of our model is 0.5159. While this relatively low R^2 value indicates that there is some factor we have not considered, our model is far better than any previous model, which relied only on number of variables and clauses. This model, which takes the form of

$$log(time) = |V| \oplus |CL| \oplus \mathrm{VCLR}$$

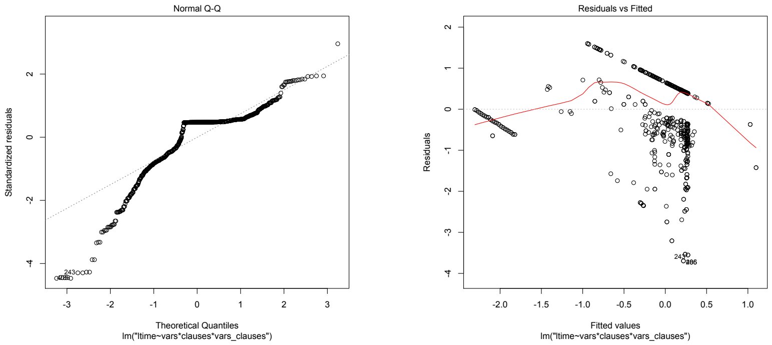
is also given for comparison and results in an adjusted R^2 of 0.3148 — making it significantly less predictive than our model. In addition to this, the more distinct S shape, and presence of sharp curves in Figure 4(a) shows that the distribution of the data is less normal. This result is confirmed when using the Kolomogorov-Sminov (KS) method to test *goodness of fit*. Our model results in



(a) Plot of normal and theoretical quantiles

(b) Residuals vs Fitted values

Fig. 3: Plots for the model including community metrics $\mathrm{R}^2 = 0.5159$



(a) Plot of normal and theoretical quantiles

(b) Residuals vs Fitted values

Fig. 4: Plots for the model without community metrics $\mathrm{R}^2 = 0.3148$

a KS value of 0.1283 compared with the previously available model which gave a KS value of 0.3154, this lack of normality makes it impossible to estimate confidence intervals for the results. However, it is possible to rank the factors by importance (because the data was standardised prior to regression). The results in Figures 3(a) and 3(b) show that our model, while not perfect, is a major step towards being a predictor for solve time. This is confirmed when viewing the results of the regression shown in the Table 1 (This table can also be viewed from our website [13]).

Bottomline Result: The main result we found from the regression is that the Q factor is involved in every one of the significant interactions at a 99.9% confidence level. In addition to this we found that $|V|$ (number of variables) alone is not significant, and $|CL|$ (number of clauses) alone is only marginally significant. Furthermore, $|CO|$ (number of communities) proved to be the most predictive effect, as well as being involved with numerous other interactions that are also significant.

3.6 Community Structure and Random Instances

In this Section, we describe the experiments, where we ran MiniSAT on a large set of randomly-generated SAT instances, to better understand the effects of varying the various factors of the input formulas in a controlled fashion. We ran a controlled experiment in which approximately 550,000 formulas were generated and executed. In performing this experiment we discovered that there is a large increase in average solution time when the $0.05 \leq Q \leq 0.13$. This can be seen clearly in Figure 5(a).

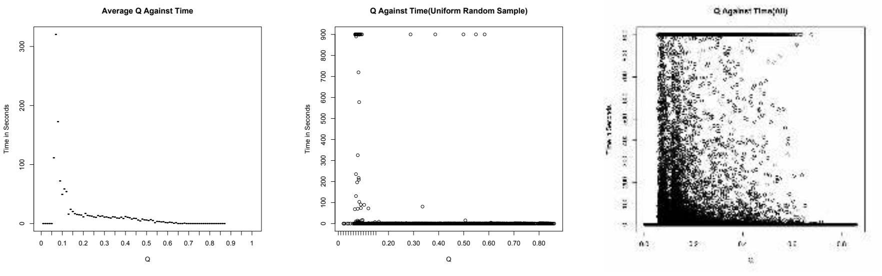
The formulas were generated by varying the number of variables from 500 to 2000 in increments of 100, the number of clauses from 2000 to 10,000 in increments of 1000, the desired number of communities from 20 to 400 in increments of 20, and the desired Q value, from zero to one in increments of 0.01. Each individual trial was repeated three times with the same characteristics. This was necessary due to the non-deterministic nature of the generation technique. The resulting experiments were ran in a random order for several hours to generate a large volume of data.

To generate a specific instance we perform the following actions: Let us assume that the set of variables be denoted as $V = \{V_i : 0 \leq i < n_v\}$ where n_v is the desired number of variables. Similarly, let the set of groups be $G = \{G_x : 0 \leq x < n_g\}$ where n_g is the desired number of groups. A group is a rough estimate of a community, and is used only to guide the generator in producing a structured problem.

First, we assign variables to groups such that each group $G_x = \{V_y : y = r_v * |G| + x; 0 \leq r_v < \frac{|V|}{|G|}\}$, where r_v is randomly selected. Next, we generate the set of clauses $C = \{C_z : 0 \leq z < n_c\}$ as follows, where n_c is the desired number of clauses such that $C_z = \{V_{z1} \vee V_{z2} \vee V_{z3}\}$. Each clause is constructed as follows: First, a group G_x and a variable $V_{z1} \in G_x$ are randomly selected. This is followed by a selection of another variable V_{z2}

Factor	Estimate	Std. Error	t value	$Pr(>\|t\|)$	Sig
$\|CO\|$	-1.237e+00	3.202e-01	-3.864	0.000121	***
$\|CL\| \odot Q \odot QCOR$	-4.226e+02	1.207e+02	-3.500	0.000492	***
$\|CL\| \odot Q$	-2.137e+02	6.136e+01	-3.483	0.000523	***
$\|CL\| \odot Q \odot \|CO\| \odot QCOR \odot VCLR$	-1.177e+03	3.461e+02	-3.402	0.000702	***
$\|CL\| \odot Q \odot \|CO\|$	-6.024e+02	1.774e+02	-3.396	0.000719	***
$Q \odot QCOR$	3.415e+02	1.023e+02	3.339	0.000881	***
Q	1.726e+02	5.200e+01	3.318	0.000947	***

Table 1: List of the factors with 99.9% significance level. $\odot$ indicates an interaction between two or more factors and *Sig* stands for Significance. The full table is listed in Appendix Table 2



(a) Average Time (b) Stratified Sample (c) All instances

Fig. 5: A plot of Q against time, for three different data sets

from either G_x or V with probability of q of being selected from G_x. Finally, a third variable V_{z3} is selected from either G_x or V with probability of q of being chosen from G_x. The value q (lies between 0 and 1) can be used as a rough estimator of Q, the modularity of the formula and is provided as input to the generator. The result is a randomly-generated 3-CNF formula.

During our analyses of the results, we discovered that our random generation technique resulted in far more results in the $0.05 \leq Q \leq 0.12$ range than in any other range. To ensure that the results seen were not because of this discrepancy, we re-ran many of the trials, focusing on data outside of this range, and generated approximately another 307,000 formula.

To ensure an unbiased analyses we also performed basic analyses on a stratified random sample taken uniformly across the range of Q. From the 545,000 results 2250 were randomly sampled, with 250 results taken from each range of 0.1, as there were no results with $Q > 0.9$ this range was not included in the sample. This process ensured their was no bias in the results based purely on frequency. The resulting sample is shown in Figure 5(b) which shows that when $0.05 \leq Q \leq 0.12$, the formula take far longer to solve. while this range is slightly different from the results of the full dataset ($0.05 \leq Q \leq 0.13$) This can be explained by the reduced dataset.

Figure 5(b) appears to shows that for all values of $Q < 0.05$ *or* $Q > 0.12$, almost all the formulas finish in approximately zero seconds, however this is only because of the scale, in reality while a large number of them do complete very quickly, in less than one second, numerous other results take varying amounts of time anywhere between zero and 900 seconds (the timeout).

The data collected from each result is as follows: number of variables, number of clauses, number of communities, Q metric, result and time, Prior to analyses we ensured the quality of the experiments by checking that we had a good distribution of results in both the SAT and UNSAT categories, and that in both there was a reasonable distribution across time. While we did discover a more even distribution of time in the case of the UNSAT vs SAT formula, it is not enough to affect the results. Similarly, while the majority of the results are in the lower end of the scale, this confirms the result that for the majority of the range of Q, most SAT instances are relatively easy to solve. Once the distribution of SAT vs UNSAT was determined, Q was plotted against time for all results Figure 5(c). From this plots we see a very clear trend in the relationship between Q and solution time, namely that when $0.05 \leq Q \leq 0.13$ there is a significant increase in average solution time. A more clear representation of these results is Figure 5(a), which plots Q on the x-axis against the average execution time of the formulas on the y-axis.

In addition to the result showing that when $0.05 \leq Q \leq 0.13$ the formula is hard, we also noticed several interesting features of these graphs. From these graphs we made several observations; Firstly, when looking at the full dataset as shown in Figure 5(c), it can be seen that none of the 2500 formula with a $Q < 0.05$ had a solve time of $> 100ms$. Secondly, while not immediately clear from looking at Figure 5(c) we discovered that none of the SAT formula had a $Q < 0.1096$, while we think it would be possible to generate a satisfiable formula that had a Q value in this range, it does not typically occur in randomly generated instances.

Bottomline Result: The basic takehome message here is that, when we accounted for potential bias in the generation process and eliminated instances that were quick to solve, we got the following result: randomly-generated instances with Q values in the range $0.05 \leq Q \leq 0.12$ (for the reduced set of instances uniformly binned for Q values ranging from 0 to 0.9 in increments of 0.1) were unsually hard for MiniSAT to solve compared to instances outside this range, and this result only depends on the Q value and not on any other factor such as number of variables or clauses.

4 Related Work

In [3] Levy et al introduced the concept of SAT problems having community structure. The paper showed that numerous problems in the SAT 2010 race contained very high modularity compared to graphs of any other nature. It was also suggested in this paper that SAT solvers are able to exploit this *hidden structure* in order to achieve good solve times. However, the paper was unable to explain what characteristics of community structure leads to poor or good solve times. Also, in [4] the authors state that while SAT solvers have shown improvements in solve times for numerous industrial applications, their has been less success in improving the solve time of randomly generated instances. They posit that this is due to the lack of structure present in randomly generated instances.

In [16] Xu et al describe a SAT solver that chooses its algorithms based on 48 features of the input formula. While they did use certain graph theoretic concepts, such as node-degree statistics, they did not consider the concept of communities as a feature of the input. The list of 48 features could be used in a more comprehensive model than the ones used in our regression. In [10] Habet et al present an empirical study of the effect of conflict analyses on solution time in CDCL solvers.

In [2] the authors present a the notion of fractal dimensions in SAT solvers, they have discovered that as the SAT solver progresses the fractal dimension increases when new learnt clauses are added to the formula. They have also discovered that learnt clauses do not connect distant parts of the formula (ones with long shortest paths between nodes), as one would expect. This is interesting when combined with the work we present stating that clauses which are comprised of variables in a small number of communities are more useful to the solver. This means that even when a learnt clause that does connect distant variable in the formula is added, it is not as useful as a clause that connects locally occurring variables.

5 Conclusions

In this paper we presented evidence that the community structure present in real world SAT instances is correlated with solution time of CDCL SAT solvers. First, we highlighted a relationship between Literal Block Distance (LBD), a measure indicating the importance of a learnt clause in CDCL solver, and community structure. In particular, learnt clauses that are shared by few communities are highly correlated with high-quality learnt clauses with low LBD scores. In other words, we have a new measure (number of communities shared by a learnt clause) of quality of learnt clauses that correlates with a very successful existing one (LBD). This result provides new insights into the efficiency of the LBD measure and should be considered to improve solver performance. Second, we introduced a model, that while not perfect, is a first step towards a predictive model for the solution time of SAT instances. Finally, we presented a result showing that randomly generated instances are particularly difficult to solve, regardless of number of clauses or variables, when their modularity is between 0.05 and 0.13.

6 Future Work

As mentioned in Section 3.4, our regression is one of the early predictive model for solve time of a CDCL solver based on community structure. However, there are many other factors not discussed in this paper that may play an important role in determining solve time, factors such as: median/mean size of clauses, the number of clauses that feature a subset of variables that appear together in another clause, the number of unique pairs of variables appearing in a clause, or the size of the largest clique in the variable graph. Any or all of these features may play a role in determining solution time. In the future we intend to explore as many of these, and as many of the 48 features from [16] as necessary to improve our model. Another potential for research is implementing a solver that takes advantage of community structure to improve solve time, this could be implemented in several ways, one of which is to create a clause deletion heuristic based on the community structure (as opposed to LBD deletion policy). Another could be to implement a decision heuristic that chooses variables that appear in learnt clauses with variables from very few number of communities. The idea being the more local a conflict clause is to a community, the higher its quality.

References

1. 2013 sat competition. http://satcompetition.org/2013/. Accessed: 2014-01-31.
2. Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, and Jordi Levy. The fractal dimension of sat formulas. *arXiv preprint arXiv:1308.5046*, 2013.
3. Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of sat formulas. In *Theory and Applications of Satisfiability Testing–SAT 2012*, pages 410–423. Springer, 2012.
4. Carlos Ansótegui and Jordi Levy. On the modularity of industrial sat instances. In *CCIA*, pages 11–20, 2011.
5. Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *proceedings of IJCAI*, pages 399–404, 2009.
6. Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.
7. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and applications of satisfiability testing*, pages 502–518. Springer, 2004.
8. Niklas Een and Niklas Sörensson. Minisat: A sat solver with conflict-clause minimization. *Sat*, 5, 2005.
9. Ian P Gent and Toby Walsh. The sat phase transition. In *ECAI*, pages 105–109. PITMAN, 1994.
10. Djamal Habet and Donia Toumi. Empirical study of the behavior of conflict analysis in cdcl solvers. In *Principles and Practice of Constraint Programming*, pages 678–693. Springer, 2013.
11. David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of sat problems. In *AAAI*, volume 92, pages 459–465. Citeseer, 1992.
12. Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.
13. Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, Gilles Audemard, and Laurent Simon. Community Structure of SAT Instances Webpage with Data and Code. https://ece.uwaterloo.ca/~vganesh/satcommunitystructure.html.
14. Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *proceedings of SAT*, pages 294–299, 2008.
15. Moshe Vardi. Phase transition and computation complexity. http://www.lsv.ens-cachan.fr/Events/fmt2012/SLIDES/moshevardi.pdf, 2012.
16. Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *J. Artif. Intell. Res.(JAIR)*, 32:565–606, 2008.
17. Wangsheng Zhang, Gang Pan, Zhaohui Wu, and Shijian Li. Online community detection for large complex networks. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1903–1909. AAAI Press, 2013.

7 Appendix

Factor	Estimate	Std. Error	t value	$Pr(>\lvert t\rvert)$	Sig
$\lvert CO\rvert$	-1.237e+00	3.202e-01	-3.864	0.000121	***
$\lvert CL\rvert \odot Q \odot QCOR$	-4.226e+02	1.207e+02	-3.500	0.000492	***
$\lvert CL\rvert \odot Q$	-2.137e+02	6.136e+01	-3.483	0.000523	***
$\lvert CL\rvert \odot Q \odot \lvert CO\rvert \odot QCOR \odot VCLR$	-1.177e+03	3.461e+02	-3.402	0.000702	***
$\lvert CL\rvert \odot Q \odot \lvert CO\rvert$	-6.024e+02	1.774e+02	-3.396	0.000719	***
$Q \odot QCOR$	3.415e+02	1.023e+02	3.339	0.000881	***
Q	1.726e+02	5.200e+01	3.318	0.000947	***
$Q \odot \lvert CO\rvert \odot QCOR$	9.451e+02	2.927e+02	3.229	0.001292	**
$Q \odot \lvert CO\rvert$	4.839e+02	1.503e+02	3.220	0.001335	**
$\lvert V\rvert \odot QCOR$	-3.177e+01	1.004e+01	-3.164	0.001617	**
$\lvert CL\rvert \odot \lvert V\rvert \odot VCLR$	-1.263e+01	4.503e+00	-2.805	0.005163	**
$\lvert CL\rvert \odot \lvert V\rvert \odot QCOR \odot VCLR$	-2.521e+01	9.008e+00	-2.798	0.005263	**
$\lvert V\rvert$	-1.376e+01	4.947e+00	-2.782	0.005526	**
$QCOR$	-1.057e+01	3.912e+00	-2.701	0.007065	**
$\lvert CL\rvert \odot \lvert V\rvert \odot QCOR$	2.096e+01	7.894e+00	2.656	0.008073	**
$(Intercept)$	-4.949e+00	1.950e+00	-2.538	0.011327	*
$\lvert CL\rvert \odot QCOR$	9.486e+00	3.792e+00	2.502	0.012556	*
$\lvert CL\rvert \odot \lvert V\rvert$	9.641e+00	3.933e+00	2.451	0.014456	*
$QCOR \odot VCLR$	9.035e+00	3.789e+00	2.385	0.017323	*
$VCLR$	4.452e+00	1.892e+00	2.353	0.018845	*
$\lvert CL\rvert$	4.299e+00	1.894e+00	2.270	0.023507	*
$\lvert V\rvert \odot QCOR \odot VCLR$	1.700e+01	7.556e+00	2.250	0.024755	*
$\lvert V\rvert \odot VCLR$	8.059e+00	3.811e+00	2.115	0.034769	*
$\lvert CL\rvert \odot \lvert V\rvert \odot Q \odot QCOR$	-4.680e+02	2.298e+02	-2.036	0.042060	*
$\lvert CL\rvert \odot \lvert V\rvert \odot Q$	-2.373e+02	1.167e+02	-2.034	0.042268	*
$\lvert CL\rvert \odot \lvert V\rvert \odot Q \odot \lvert CO\rvert$	-6.594e+02	3.315e+02	-1.989	0.047042	*
$\lvert CL\rvert \odot \lvert V\rvert \odot Q \odot \lvert CO\rvert \odot QCOR$	-1.286e+03	6.469e+02	-1.988	0.047160	*

Table 2: List of all significant effects, three stars indicates the highest level of confidence that the effect is important. $\odot$ indicates an interaction between two or more factors and *Sig* stands for Significance