On the Hierarchical Community Structure of Practical SAT Formulas

Chunxiao Li, Jonathan Chung, Soham Mukherjee, Marc Vinyals, Noah Fleming, Antonina Kolokolova, Alice Mu and Vijay Ganesh

June 28, 2021

On the Hierarchical Community Structure of Practical Boolean Formulas

Chunxiao Li[1*†], Jonathan Chung[1*], Soham Mukherjee[1,2], Marc Vinyals[3], Noah Fleming[4], Antonina Kolokolova[5], Alice Mu[1], and Vijay Ganesh[1]

[1] University of Waterloo, Waterloo, Canada
[2] Perimeter Institute for Theoretical Physics, Waterloo, Canada
[3] Technion, Haifa, Israel
[4] University of Toronto, Toronto, Canada
[5] Memorial University of Newfoundland, St. John's, Canada

Abstract. Modern CDCL SAT solvers easily solve industrial instances containing tens of millions of variables and clauses, despite the theoretical intractability of the SAT problem. This gap between practice and theory is a central problem in solver research. It is believed that SAT solvers exploit structure inherent in industrial instances, and hence there have been numerous attempts over the last 25 years at characterizing this structure via parameters. These can be classified as *rigorous*, i.e., they serve as a basis for complexity-theoretic upper bounds (e.g., backdoors), or *correlative*, i.e., they correlate well with solver run time and are observed in industrial instances (e.g., community structure). Unfortunately, no parameter proposed to date has been shown to be both strongly correlative and rigorous over a large fraction of industrial instances.
Given the sheer difficulty of the problem, we aim for an intermediate goal of proposing a set of parameters that is strongly correlative and has good theoretical properties. Specifically, we propose parameters based on a graph partitioning called Hierarchical Community Structure (HCS), which captures the recursive community structure of a graph of a Boolean formula. We show that HCS parameters are strongly correlative with solver run time using an Empirical Hardness Model, and further build a classifier based on HCS parameters that distinguishes between easy industrial and hard random/crafted instances with very high accuracy. We further strengthen our hypotheses via scaling studies. On the theoretical side, we show that counterexamples which plagued flat community structure do not apply to HCS, and that there is a subset of HCS parameters such that restricting them limits the size of embeddable expanders.

1 Introduction

Over the last two decades, Conflict-Driven Clause-Learning (CDCL) SAT solvers have had a dramatic impact on many sub-fields of software engineering [10], formal methods [12], security [16,45], and AI [8], thanks to their ability to solve

[*]Joint first author
[†]Work done in part while the authors were at the 2021 Satisfiability: Theory, Practice, and Beyond program at the Simons Institute, Berkeley, CA, USA.

large real-world instances with tens of millions of variables and clauses [38], notwithstanding the fact that the Boolean satisfiability (SAT) problem is known to be NP-complete and is believed to be intractable [15]. A plausible explanation of this apparent contradiction would be that NP-completeness of the SAT problem is established in a worst-case setting, while the dramatic efficiency of modern SAT solvers is witnessed over "practical" instances. However, despite over two decades of effort, we still do not have an appropriate mathematical characterization of practical instances (or a suitable subset thereof) and attendant complexity-theoretic upper and lower bounds. This gap between theory and practice is rightly considered one of the central problems in solver research by theorists and practitioners alike.

The fundamental premise in this line of work is that SAT solvers are able to find short proofs (if such proofs exist) in polynomial time (i.e., they are efficient) for industrial instances and that they are able to do so because they somehow exploit the underlying properties (a.k.a. structure) of such industrial Boolean formulas[1], and, further, that hard randomly-generated or crafted instances are difficult because they do not possess such structure. Consequently, considerable work has been done in characterizing the structure of industrial instances via parameters. The parameters discussed in literature so far can be broadly classified into two categories: correlative and rigorous[2]. The term *correlative* refers to parameters that take a specific range of values in industrial instances (as opposed to random/crafted) and further have been shown to correlate well with solver run time. This suggests that the structure captured by such parameters might explain why solvers are efficient. An example of such a parameter is modularity (more generally community structure [4]). By contrast, the term *rigorous* refers to parameters that characterize classes of formulas that are fixed-parameter tractable (FPT), such as backdoors [44,48], backbones [29], treewidth, and branchwidth [1,37], among many others [37], or have been used to prove complexity-theoretic bounds over randomly-generated classes of formulas such as clause-variable ratio (a.k.a., density) [14,39].

The eventual goal in this context is to discover a parameter or set of parameters that is both strongly correlative and rigorous, such that it can then be used to establish parameterized complexity-theoretic bounds on an appropriate mathematical abstraction of CDCL SAT solvers, thus finally settling this decades-long open question. Unfortunately, the problem with all the previously proposed rigorous parameters is that either "good" ranges of values for these parameters are not witnessed in industrial instances (e.g., such instances can have both large and small backdoors) or they do not correlate well with solver run time (e.g., many industrial instances have large treewidth and yet are easy to solve, and treewidth alone does not correlate well with solving time [28]).

Consequently, many attempts have been made at discovering correlative parameters that could form the basis of rigorous analysis [4,21]. Unfortunately, all

[1]The term industrial is loosely defined to encompass instances obtained from hardware and software testing, analysis, and verification applications.

[2]Using terminology by Stefan Szeider [43].

such correlative parameters either seem to be difficult to work with theoretically (e.g., fractal dimension [2]) or have obvious counterexamples, i.e., it is easy to show the existence of formulas that simultaneously have "good" parameter values and are provably hard-to-solve. For example, it was shown that industrial instances have high modularity, i.e., supposedly good community structure [4], and that there is good-to-strong correlation between modularity and solver run time [32]. However, Mull et al. [30] later exhibited a family of formulas that have high modularity and require exponential-sized proofs to refute. Finally, this line of research suffers from important methodological issues, that is, experimental methods and evidence provided for correlative parameters tend not to be consistent across different papers in the literature.

Hierarchical Community Structure of Boolean Formulas: Given the sheer difficulty of the problem, we aim for an intermediate goal of proposing a set of parameters that is strongly correlative and has good theoretical properties. Specifically, we propose a set of parameters based on a graph-theoretic structure called Hierarchical Community Structure (HCS), inspired by a commonly-studied concept in the context of hierarchical networks [13,35], which satisfies all the empirical tests hinted above and has better theoretical properties than previously proposed correlative parameters. The intuition behind HCS is that it neatly captures the structure present in human-developed systems which tend to be modular and hierarchical [41], and we expect this structure to be inherited by Boolean formulas modelling these systems.

Contributions[3]:

1. **Empirical Result 1 (HCS and Industrial Instances):** We show that a set of parameters based on the HCS of the variable-incidence graph (VIG) of Boolean formulas are effective in distinguishing industrial instances from random/crafted ones. Moreover, we build a classifier that robustly classifies SAT instances into the categories they belong to (verification, random, etc.). The classification accuracy is approximately 99% and we perform a variety of tests to ensure there is no overfitting (See Section 5.1).
2. **Empirical Result 2 (Correlation between HCS and Solver Run Time):** We build an empirical hardness model based on our HCS parameters to predict the solver run time for a given problem instance. Our model, based on regression, performs well, achieving an R^2 score of 0.83, much stronger than previous such results (See Section 5.2)
3. **Empirical Result 3 (Scaling Experiments of HCS Instances):** We empirically show, via scaling experiments, that HCS parameters such as community degree and leaf-community size positively correlate with solving time. We empirically demonstrate that formulas whose HCS decompositions fall in a good range of parameter values are easier to solve than instances with a bad range of HCS parameter values (See Section 5.4).

[3]Instance generator and data can be found at `https://satsolvercomplexity.github.io/hcs`. Also, for the full-length paper and appendices (with proofs of theorems in Section 6), please refer to the arXiv version of the paper [26].

4. **Theoretical Results:** We theoretically justify our choice of HCS by showing that it behaves better than other parameters. More concretely, we show the advantages of hierarchical over flat community structure by identifying HCS parameters which let us avoid hard formulas that can be used as counterexamples to community structure [30], and by showing graphs where HCS can find the proper communities where flat modularity cannot. We also show that there is a subset of HCS parameters (leaf-community size, community degree, and fraction of inter-community edges) such that restricting them limits the size of embeddable expanders (See Section 6).
5. **Instance Generator:** Finally, we provide an HCS-based instance generator which takes input values of our proposed parameters and outputs a formula that satisfies those values. This generator can be used to generate "easy" and "hard" formulas with different hierarchical structures (See Section 5.4).

Research Methodology: We also codify a set of empirical tests which we believe parameters must pass in order to be considered for further theoretical analysis. While other researchers have considered one or more of these tests, we bring them together into a coherent and sound research methodology that can be used for future research in formula parameterization (See Section 3). We believe that the combination of these tests provides a strong basis for a correlative parameter to be considered worthy of further analysis.

2 Preliminaries

Variable Incidence Graph (VIG): Researchers have proposed a variety of graphs to study graph-theoretic properties of Boolean formulas. In this work we focus on the Variable Incidence Graph (VIG), primarily due to the relative ease of computing community structure over VIGs compared to other graph representations. The VIG for a formula F over variables $x_1, \dots, x_n$ has n vertices, one for each variable. There is an edge between vertices x_i and x_j if both x_i and x_j occur in some clause C_k in F. One drawback of VIGs is that a clause of width w corresponds to a clique of size w in the VIG. Therefore, large width clauses (of size n^{ε}) can significantly distort the structure of a VIG, and formulas with such large width clauses should have their width reduced (via standard techniques) before using a VIG.

Community Structure and Modularity: Intuitively, a set of variables (vertices in the VIG) of a formula forms a community if these variables are more densely connected to each other than to variables outside of the set. An (optimal) community structure of a graph is a partition $P = \{V_1, \dots, V_k\}$ of its vertices into communities that optimizes some measure capturing this intuition, for instance modularity [31], which is the one we use in this paper. Let $G = (V, E)$ be a graph with adjacency matrix A and for each vertex $v \in V$ denote by $d(v)$ its degree. Let $\delta_P \colon V \times V \to \{0, 1\}$ be the community indicator function of a partition, i.e. $\delta_P(u, v) = 1$ iff vertices u and v belong to the same community in

P. The *modularity* of the partition P is

$$Q(P) := \frac{1}{2|E|} \sum_{u,v \in V} \left[A_{u,v} - \frac{d(u)d(v)}{2|E|} \right] \delta_P(u,v) \tag{1}$$

Note that $Q(P)$ ranges from -0.5 to 1, with values close to 1 indicating good community structure. We define the modularity $Q(G)$ of a graph G as the maximum modularity over all possible partitions, with corresponding partition $\mathcal{P}(G)$. Other measures may produce radically different partitions.
Expansion of a Graph: Expansion is a measure of graph connectivity [23]. Out of several equivalent such measures, the most convenient to relate to HCS is *edge expansion*: given a subset of vertices $S \subseteq V$, its edge expansion is $h(S) = |E(S, V \backslash S)|/|S|$, and the edge expansion of a graph is $h(G) = \min_{1 \leq |S| \leq n/2} h(S)$. A graph family G_n is an expander if $h(G_n)$ is bounded away from zero. Resolution lower bounds (of both random and crafted formulas) often rely on strong expansion properties of the graph [5].

3 Research Methodology

As stated above, the eventual goal of the research presented here is to discover a structure and an associated parameterization that is highly correlative with solver run time, is witnessed in industrial instances, and is rigorous, i.e., forms the basis for an upper bound on the parameterized complexity [37] of the CDCL algorithm. Considerable work has already been done in attempting to identify exactly such a set of parameters [32]. However, we observed that there is a wide diversity of research methodologies adopted by researchers in the past. We bring together the best lessons learned into what we believe to be a sound, coherent, and comprehensive research methodology explained below. We argue that every set of parameters must meet the following empirical requirements in order to be considered correlative:

1. **Structure of Industrial vs. Random/Crafted Instances:** A requisite for a structure to be considered correlative is that industrial instances must fall within a certain range of values for the associated parameters, while random and crafted instances must have a different range. An example of such a structure is the community structure of the VIG of Boolean formulas, as parameterized by modularity. Multiple experiments have shown that industrial instances have high modularity (close to 1), while random instances tend to have low modularity (close to 0) [32]. This could be demonstrated via a correlation experiment or by building a classifier that takes parameter values as input features.
2. **Correlation between Structure and Solver Run Time:** Another requirement is correlation between parameters of a structure and solver run time. Once again, community structure (and the associated modularity parameter) forms a good example of a structure that passes this essential test.

For example, it has been shown that the modularity of the community structure of industrial instances (resp. random instances) correlates well with low (resp. high) solver run time [32]. One may use either correlation methods or suitable machine learning predictors (e.g., random forest) as evidence here.

3. **Scaling Studies:** To further strengthen the experimental evidence, we require that the chosen structure and its associated parameters must pass an appropriately designed scaling study. The idea here is to vary one parameter value while keeping as much of the rest of the formula structure constant as possible, and see its effect on solver run time. An example of such a study is the work of Zulkoski et al. [47], who showed that increasing the mergeability metric has a significant effect on solver run time.

Limitations of Empirical Conclusions: As the reader is well aware, any attempt at empirically discovering a suitable structure (and associated parameterization) of Boolean formulas and experimentally explaining the power of solvers is fraught with peril, since all such experiments involve pragmatic design decisions (e.g., which solver was used, choice of benchmarks, etc.) and hence may lead to contingent or non-generalizable conclusions. For example, one can never quite eliminate a parameter from further theoretical analysis based on empirical tests alone, for the parameter may fail an empirical test on account of benchmarks considered or other contingencies. Another well-understood issue with conclusions based on empirical analysis alone is that they by themselves cannot imply provable statements about asymptotic behavior of algorithms. However, one can use empirical analysis to check or expose gaps between the behavior of an algorithm and the tightness of asymptotic statements (e.g., the gap between efficient typical-case behavior vs. loose worst-case statements). Having said all this, we believe that the above methodology is a bare minimum that a set of parameters must pass before being considered worthy of further theoretical analysis. In Section 5, we go into further detail about how we protect against certain contingent experimental conclusions.
Limits of Theoretical Analysis: Another important aspect to bear in mind is that it is unlikely any small set of parameters can cleanly separate all easy instances from hard ones. At best, our expectation is that we can characterize a large subset of easy real-world instances via the parameters presented here, and thus take a step towards settling the central question of solver research.

4 Hierarchical Community Structure

Given that many human-developed systems are modular and hierarchical [41], it is natural to hypothesize that these properties are transferred over to Boolean formulas that capture the behaviour of such systems. We additionally hypothesize that purely randomly-generated or crafted formulas do not have these properties of hierarchy and modularity, and that this difference partly explains why solvers are efficient for the former and not for the latter class of instances. We formalize this intuition via a graph-theoretic concept called Hierarchical Community Structure (HCS), where communities can be recursively decomposed into

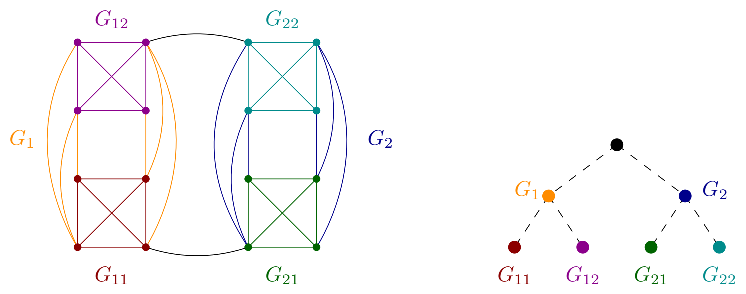

Fig. 1. A hierarchical decomposition (right) constructed by recursively maximizing the modularity of the graph (left).

smaller sub-communities. Although the notion of HCS has been widely studied [13,35], it has not been considered in the context of Boolean formulas before.

Hierarchical Community Structure Definition: A *hierarchical decomposition* of a graph G is a recursive partitioning of G into subgraphs, represented as a tree T. Each node v in the tree T is labelled with a subgraph of G, with the root labelled with G itself. The children of a node corresponding to a (sub)graph H are labelled with a partitioning of H into subgraphs $\{H_1, \ldots, H_k\}$; see Figure 1. There are many ways to build such hierarchical decompositions. The method that we choose constructs the tree by recursively maximizing the modularity, as in the hierarchical multiresolution method [22]. We call this the HCS decomposition of a graph G: for a node v in the tree T corresponding to a subgraph H of G, we construct $|\mathcal{P}(H)|$ children, one for each of the subgraphs induced by the modularity-maximizing partition $\mathcal{P}(H)$, unless $|\mathcal{P}(H)| = 1$, in which case v becomes a *leaf* of the tree. In the case of HCS decompositions, we refer to the subgraphs labelling the nodes in the tree as *communities* of G.

We are interested in comparing the hierarchical community structures of Boolean formulas in conjunctive normal form, represented by their VIGs. For this comparison, we use the following parameters:

– The *community degree* of a community in a HCS decomposition is the number of children of its corresponding node.
– A *leaf-community* is one with degree 0.
– The *size* of a community is its number of vertices.
– The *depth* or *level* of a community is its distance from the root.
– The *inter-community edges* of a partition $\mathcal{P}(H)$ are $E_{IC}(H) = \bigcup_{H_i, H_j \in \mathcal{P}(H)} E(H_i, H_j)$, the edges between all pairs of subgraphs, and their endpoints $V_{IC}(H) = \bigcup E_{IC}$ are the *inter-community vertices*. Note that $2|E_{IC}(H)|/|H|$ is an upper bound for the edge expansion of H.

Note that these parameters are not independent. For example, changes in the number of inter-community vertices or inter-community edges will affect modularity. Since our hierarchical decomposition is constructed using modularity, this could affect the entire decomposition and hence the other parameters.

5 Empirical Results

We now turn to the results of our empirical investigations with HCS parameters. We computed 49 unique parameters capturing the HCS structure, together with several base parameters measuring different structural properties of input VIGs[4]. To compute the hierarchical community structure, we used the Louvain method [7] to detect communities and recursively call the Louvain method to produce a hierarchical decomposition. The Louvain method is considered to be more efficient and produces higher-modularity partitions than other known algorithms.

Experimental Design. In our experiments we used a set of 10 869 instances from five classes, which we believe is sufficiently large and diverse to draw sound empirical conclusions (See Appendix [26]). We did not explicitly balance the ratio of satisfiable instances in our benchmark selection because we expect our methods to be sufficiently robust as long as the benchmark contains a sufficient number of SAT and UNSAT instances.

In order to get interesting instances for modern solvers, we considered formulas which were previously used in the SAT competition from 2016 to 2018 [38]. Specifically, we took instances from five major tracks of the competition: agile, verification, crypto, crafted, and random. We also generated additional instances for some classes: for verification, we scaled the number of unrolls when encoding finite state machines for bounded model checking; for crypto, we encoded SHA-1 and SHA-256 preimage problems; for crafted, we generated combinatorial problems using `cnfgen` [25]; and for random, we generated k-CNFs at the corresponding threshold CVRs for $k \in \{3, 5\}$, again using `cnfgen`. A summary of the instances is presented in the Appendix.

We preprocessed all formulas using the MiniSAT preprocessor [17], and used MapleSAT [27] as our CDCL solver of choice since it is a leading and representative solver. The core of the preprocessing was a combination of variable elimination with subsumption and self-subsuming resolution [17]. For computing satisfiability and running time, we used SHARCNET's Intel E5-2683 v4 (Broadwell) 2.1 GHz processors [40], limiting the computation time to 5 000 seconds[5]. For parameter computation we did not limit the type of processor because structural parameter values are independent of processing power.

5.1 HCS-based Category Classification of Boolean Formulas

The question whether our set of HCS parameters is able to capture the underlying structure that differentiates industrial instances from the rest naturally lends itself to a classification problem. Therefore, we built a multi-class Random Forest classifier to classify a given SAT instance into one of the five categories:

[4]For a complete list, see: `https://satsolvercomplexity.github.io/hcs/data`
[5]This value is the time limit used by the SAT competition.

Table 1. Results for classification and regression experiments with HCS parameters. For regression we report R^2 values, whereas for classification we report the mean of the balanced accuracy score over 5 cross-validation datasets.

	Category	**Runtime**
Score	0.996 ± 0.001	0.825 ± 0.016

Top 5 features	`rootMergeability`
	`maxInterEdges/CommunitySize`
	`cvr`
	`leafCommunitySize`
	`lvl2InterEdges/lvl2InterVars`
		`rootInterEdges`
		`lvl2Mergeability`
		`cvr`
		`leafCommunitySize`
		`lvl3Modularity`

verification, agile, random, crafted, or crypto. Random Forests [9] can learn complex, highly non-linear relationships while having simple structure, and hence are easier to interpret than other models (e.g., deep neural networks).

We used an off-the-shelf implementation of a Random Forest classifier implemented as `sklearn.ensemble.RandomForestClassifier` in scikit-learn [33]. Using the default set of parameters in scikit-learn version 0.24, we trained our classifier using 800 randomly sampled instances of each category on a set of 49 features to predict the class of the problem instance. We found that our classifier performs extremely well, giving an average accuracy score of 0.99 over 5 cross-validation datasets. Further, the accuracy did not depend on our choice of classifier. In particular, we found similar accuracy scores when we used C-Support Vector classification [34] instead of Random Forests.

We also determined the five most important features used by our classifier. Since several features in our feature set are highly correlated, we first performed a hierarchical clustering on the feature set based on Spearman rank-order correlations. From the 22 clusters that were generated, we arbitrarily chose a single feature from each cluster as a representative member of the cluster f[6]. Using these 22 representative features, we then computed their importance using permutation importance [9]. In Table 1 we list the top five representative features from each cluster, not necessarily in order of importance.

5.2 HCS-based Empirical Hardness Model

We used our HCS parameters to build an empirical hardness model (EHM) to predict the run time of MapleSAT on a given instance. Since the solving time is a continuous variable, we considered a regression model built using Random Forests, namely `sklearn.ensemble.RandomForestRegressor` from scikit-learn [33]. Before training our regression model, we removed instances which timed-out at 5 000 seconds and those instances that were solved almost immediately (in zero seconds) to avoid issues with artificial cut-off boundaries. We

[6]See `https://satsolvercomplexity.github.io/hcs/data` for details on clusters.

then trained our Random Forest model using the default set of parameters in scikit-learn version 0.24 to predict the logarithm of the solving time using the remaining 1 880 instances, equally distributed between different categories.

We observed that our regression model performs quite well, with an R^2 score [42] of 0.83, which implies that in the training set, almost 83% of the variability of the dependent variable (i.e., in our case, the logarithm of the solving time) is accounted for, and the remaining 17% is still unaccounted for by our choice of parameters. Similar to category classification, we also looked for the top five predictive features used by our Random Forest regression model using the exact same process. We list the representative features in Table 1.

Additionally, we trained our EHM on each category of instances separately. We found that the performance of our EHM varies with instance category. Concretely, agile outperformed all other categories with an average R^2 value of 0.94, followed by random, crafted and verification instances with scores of $0.81, 0.85$ and 0.74 respectively. The worst performance was shown by the instances in crypto, with a score of 0.48.

5.3 HCS Parameter Value Ranges for Industrial/Random Instances

In the previous section, we reported on the top five parameters most predictive of the solver runtime in the context of our Random Forest regression model. These parameters can be divided into five distinct classes of parameters: mergeability-based, modularity-based, inter-community edge based, CVR, and leaf-community size. The parameters CVR, mergeability and modularity have been studied by previous work. CVR [11] is perhaps the most studied parameter among the three. Zulkoski et al. [47] showed that mergeability, along with combinations of other parameters, correlates well with solver run time; Ansotegui et al. [4] showed that industrial instances have good modularity compared to random instances; and Newsham et al. [32] showed that modularity has good-to-strong correlation with solver run time. We examined the remaining parameters, i.e. inter-community edge based parameters (`rootInterEdges`) and leaf-community size to gain a better understanding of the impact of these parameters on the problem structure and solver runtime, respectively. In this subsection, we look at how HCS parameters scale as the size of industrial instances increases. And in Section 5.4, we introduce a HCS instance generator, which we use to perform a set of controlled experiments. We then discuss how the hardness of the instances changes when certain HCS parameters are increased/decreased.

Observations. We observe that hierarchical decomposition generally produces leaf communities of maximal size comparable to the largest clause width, except for very unbalanced formulas (easy for other reasons). The community degree is highest at root level of every instance, and seems to be bounded by $O(\log n)$. This fits within the range of parameters considered in Section 6.

In Figure 2, we show how the inter-community edge based parameter `root-InterEdges` scales with the number of variables in a formula, for verification and random instances. We note that for random instances, `rootInterEdges` grows

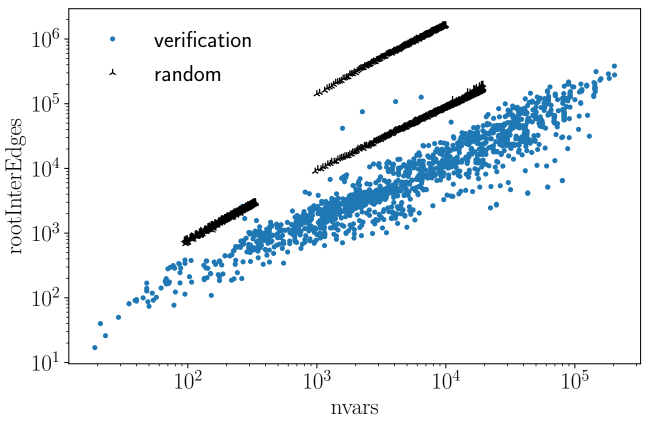

Fig. 2. Dependence of the number of inter-community edges at the root level (rootInterEdges) vs. the number of variables in a formula, for verification and random instances in our dataset. The two distinct lines (starting from the bottom) for random instances correspond to 3-CNFs and 5-CNFs, respectively.

linearly with the instance size, whereas in verification instances it grows sublinearly. This supports our intuition that graphs of hard (random) instances are expanders, whereas graphs of industrial instances are not.

5.4 Scaling Experiments with HCS parameters

Instance Generator. To isolate the effects of HCS parameters on solver runtime, we built an HCS instance generator to construct SAT instances with varying leaf-community size and other HCS parameters. On a high level, the instance generator constructs instances bottom-up, starting with random disjoint formulas of predefined CVR as leaf communities, then combining them recursively by introducing bridge clauses with variables in at least two sub-communities to form super-communities at that level, which in turn are combined at the following level. We point out that in our generator, modularity is specified implicitly through the above parameters, and we do not control for mergeability at all. We refer the reader to the works by Zulkoski et al. [47] and Giráldez-Cru [20] for literature on the empirical behaviours of mergeability and power law, respectively.

It is important to note that our HCS instance generator is not intended to be perfectly representative of real-world instances. In fact, there are multiple properties of our generated instances which are not reflective of industrial instances. For example, our generator assumes that all leaf-communities have the same size and depth, which is demonstrably untrue of industrial instances. In some cases, the communities produced by our generator might not be the same as the communities which would be detected using the Louvain method to perform a hierarchical community decomposition. For example, it might be possible to further decompose the generated "leaf-communities" into smaller communities. Thus, our generator is only intended to demonstrate the effect of varying HCS parameters on solver runtime.

Observations. We constructed formulas with varying CVR, power law parameter, hierarchical degree, depth, inter-community edge density, inter-community variable density, and clause width. We found evidence which suggests that increasing any of leaf-community size, depth, or community degree, while keeping every other HCS parameter fixed, increases the overall hardness of the generated formula. For example, we found that changing the size of leaf-communities from 15 variables to 20, the solving time changed from 4.96 seconds to upwards of 5000 seconds. Similarly, changing the depth from 4 to 5 resulted in an increase in solving time from 0.03 seconds to over 5000 seconds.

5.5 Discussion of Empirical Results

The goal of our experimental work was to first ascertain whether HCS parameters can distinguish between industrial and random/crafted instances, and whether these parameters show any correlation with CDCL solver runtime. The robustness of our classifier indicates that HCS parameters are indeed representative of the underlying structure of Boolean formulas from different categories. Further, our empirical hardness model confirms that the correlation of HCS parameters with solver run time is strong—much stronger than previously proposed parameters. We also find that our HCS parameters are more effective in capturing the hardness or easiness of formulas from industrial/agile/random/crafted, but not crypto. The crypto class is an outlier. It is not clear from our experiments (nor any previous ones) as to why crypto instances are hard for CDCL solvers.

We also identified the top five (representative) parameters in terms of their importance in predicting the category (classification) or runtime of an instance (regression). The accuracy for classification and regression with only the top features features dropped to 0.94 and 0.77, respectively, suggesting that only a few parameters are likely to play a role in closing the question on why solvers are efficient for industrial instances. Note that a classification accuracy of 0.99 is likely to suggest that our model is over-fitting. Fortunately, in our case our models are trained over a large set of instances obtained via very different methods (e.g., random over various widths, different kinds of crafted, verification instances from different domains), and therefore, there is sufficient entropy in our data set so that overfitting is unlikely to be a concern for the robustness of our model.

In our investigation of parameters based on inter-community edges and leaf-community size, we found that industrial instances typically have small average leaf-community size, high modularity, and relatively few inter-community edges, while random/crafted have larger average leaf-community size, low modularity, and a very high number of inter-community edges. This suggests that leaf-community size and the fraction of inter-community edges, as well as community degree, are important HCS parameters to consider further.

6 Theoretical Results

In this section, we show that hierarchical decomposition avoids some of the pitfalls of flat community structure, a promising correlative parameter for explaining easiness of the industrial instances [32]. Community structure was theoretically shown to be insufficient by Mull et al. [30], where they showed that formulas with good community structure can have random formulas embedded in them either in a community or over the inter-community edges. To avoid embedding a random formula in a community, its size has to be small (relative to the entire graph), and avoiding expanders over inter-community edges requires that there not be too many communities. A way to be able to restrict both is to consider a hierarchical decomposition, limiting both the number of sub-communities (community degree) in each level of the decomposition, as well as the leaf community size thus avoiding the most important issues that flat community structure suffers from.

Based on our experimental work, we narrow down the most predictive HCS parameters to be leaf-community size, community degree, and the number inter-community edges in each decomposition. These parameters also play a role in our theoretical results below. For a formula to have "good" HCS, we restrict the parameter ranges as follows: the graph must exhibit $O(\log n)$ leaf-community size and community degree, and have a small number of inter-community edges in each decomposition of a community. These assumptions are supported by our experimental results (See Appendix [26]). We show that these restrictions are necessary in Appendix, where we also present a significantly simplified proof of the result of Mull et al. [30].

Bounding the Size of Expanders in Good HCS Graphs. Ideally, we would like to be able to prove an upper bound on proof size or search time which depends on the HCS parameters of a formula. Unfortunately, our current state of understanding does not allow for that. A step towards such a result would be to show that formulas with good HCS (and associated parameter value ranges) are not susceptible to typical methods of proving resolution lower bounds. Currently, all resolution bounds exploit *expansion* properties – typically *boundary expansion* – of the CNF formula (or more precisely its bipartite constraint-variable incidence graph (CVIG)). Therefore our goal is to show that formulas with good HCS parameters have poor expansion properties, and also do not have large expanding subgraphs embedded within them. Note that the VIG is related to the CVIG by taking the square of its adjacency matrix, from where it follows that, for formulas with low width, if the VIG is not edge-expanding then the CVIG is not vertex-expanding. Furthermore, again for formulas with low width, vertex expansion is closely related to boundary expansion. Hence we only need to focus on VIG edge expansion. With this in mind, we state several positive and negative results.

First, we observe (see Appendix) that if the number of inter-community edges at the top level of the decomposition grows sub-linearly with n and at least two sub-communities contain a constant fraction of vertices, then this graph family is not an expander. Unfortunately, we can also show (see Appendix) that graphs with good HCS can simultaneously have sub-graphs that are large expanders, with the worst case being very sparse expanders, capable of "hiding" in the hierarchical decomposition by contributing relatively few edges to any cut. To avoid that, we require an explicit bound on the number of inter-community edges, in addition to small community degree and small leaf-community size. This lets us prove the following statement.

Theorem 1. *Let $G = \{G_n\}$ be a family of graphs. Let $f(n) \in \omega(poly(\log n))$, $f(n) \in O(n)$. Assume that G has HCS with the number of inter-community edges $o(f(n))$ for every community C of size at least $\Omega(f(n))$ and depth is bounded by $O(\log n)$. Then G does not contain an expander of size $f(n)$ as a subgraph.*

Note that our experiments show that the leaf size and depth in industrial instances are relatively small and the number of inter-community edges grows slowly. From this and the theorem above, we can show that graphs with *very* good HCS properties do not contain linear-sized expanders.

Lower Bounds Against HCS: We are also able to show several of strong lower bounds on formulas with good HCS (see Appendix). For a number of combinations of parameters, we show that restricting ourselves to "good" ranges of these parameters does not rule out formulas which require superpolynomial size resolution refutations. Our most striking counterexample essentially shows that if the degree of the VIG is more than a small constant, then it is possible to embed formulas of superpolynomial resolution complexity. In contrast with the previous results on the size of embeddable expanders in instances with good HCS, this result shows how to embed a sparse expander of superlogarithmic size.

Hierarchical vs. Flat Modularity: It is well-known that modularity suffers from a *resolution limit* and cannot detect communities smaller than a certain threshold [18], and that HCS can avoid this problem in some instances [7]. In Appendix we provide an asymptotic, rigorous statement of this observation.

Theorem 2. *There exists a graph G whose natural communities are of size $\log(n)$ and correspond to the (leaf) HCS communities, while the partition maximizing modularity consists of communities of size $\Theta\left(\sqrt{n/\log^3 n}\right)$.*

7 Related Work

Community Structure: Using modularity to measure community structure allows one to distinguish industrial instances from randomly-generated ones [4]. Unfortunately, it has been shown that expanders can be embedded within formulas with high modularity [30], i.e., there exist formulas that have good community structure and yet are hard for solvers.

Heterogeneity: Unlike uniformly-random formulas, the variable degrees in industrial formulas follow a powerlaw distribution [3]. However, degree heterogeneity alone fails to explain the hardness of SAT instances. Some heterogeneous random k-SAT instances were shown to have superpolynomial resolution size [6], making them intractable for current solvers.

SATzilla: SATzilla uses 138 disparate parameters [46], some of which are probes aimed at capturing a SAT solver's state at runtime, to predict solver running time. Unfortunately, there is little or no evidence that most of these parameters are amenable to theoretical analysis.

Clause-Variable Ratio (CVR): Cheeseman et al. [11] observed the satisfiability threshold behavior for random k-SAT formulas, where they show formulas are harder when their CVR are closer to the satisfiability threshold. Outside of extreme cases, CVR alone seems to be insufficient to explain hardness (or easiness) of instances, as it is possible to generate both easy and hard formulas with the same CVR [19]. Satisfiability thresholds are poorly defined for industrial instances, and Coarfa et al. [14] demonstrated the existence of instances for which the satisfiability threshold is not equal to the hardness threshold.

Treewidth: Although there are polynomial-time non-CDCL algorithms for SAT instances with bounded treewidth [1], treewidth by itself does not appear to be a predictive parameter of CDCL solver runtime. For example, Mateescu [28] showed that some easy instances have large treewidth, and later it was shown that treewidth alone does not seem to correlate well with solving time [47].

Backdoors: In theory, the existence of small backdoors [44,36] should allow CDCL solvers to solve instances quickly, but empirically backdoors have been shown not to strongly correlate with CDCL solver run time [24].

8 Conclusions and Future Work

In this paper, we propose HCS as a correlative set of parameters for explaining the power of CDCL SAT solvers over industrial instances, which also has good theoretical properties. Empirically, HCS parameters are much more predictive than previously proposed correlative parameters in terms of classifying instances into random/crafted vs. industrial, and in terms of predicting solver run time. Among the top five most predictive parameters, three are HCS parameters, namely leaf-community size, modularity and fraction of inter-community edges. The remaining two are cvr and mergeability. We further identify the following core HCS parameters that are the most predictive among all HCS parameters, namely, leaf-community size, modularity, and fraction of inter-community edges. Indeed, these same parameters also play a role in our subsequent theoretical analysis, where we show that counterexamples to flat community structure do not apply to HCS, and that restricting certain HCS parameters limits the size of embeddable expanders. In the final analysis, we believe that HCS, along with other parameters such as mergeability or heterogeneity, will play a role in finally settling the question of why solvers are efficient over industrial instances.

References

1. Alekhnovich, M., Razborov, A.: Satisfiability, Branch-Width and Tseitin Tautologies. computational complexity **20**(4), 649–678 (Dec 2011). https://doi.org/10.1007/s00037-011-0033-1
2. Ansótegui, C., Bonet, M.L., Giráldez-Cru, J., Levy, J.: The Fractal Dimension of SAT Formulas. In: Proceedings of the 7th International Joint Conference on Automated Reasoning - IJCAR 2014. pp. 107–121 (2014). https://doi.org/10.1007/978-3-319-08587-6_8
3. Ansótegui, C., Bonet, M.L., Levy, J.: Towards Industrial-Like Random SAT Instances. In: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence. pp. 387–392 (2009)
4. Ansótegui, C., Giráldez-Cru, J., Levy, J.: The Community Structure of SAT Formulas. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing - SAT 2012. pp. 410–423 (2012). https://doi.org/10.1007/978-3-642-31612-8_31
5. Ben-Sasson, E., Wigderson, A.: Short Proofs are Narrow—Resolution Made Simple. Journal of the ACM (JACM) **48**(2), 149–169 (2001)
6. Bläsius, T., Friedrich, T., Göbel, A., Levy, J., Rothenberger, R.: The Impact of Heterogeneity and Geometry on the Proof Complexity of Random Satisfiability. In: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021. pp. 42–53 (2021). https://doi.org/10.1137/1.9781611976465.4
7. Blondel, V., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast Unfolding of Communities in Large Networks. Journal of Statistical Mechanics Theory and Experiment **2008** (Apr 2008). https://doi.org/10.1088/1742-5468/2008/10/P10008
8. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial intelligence **90**(1-2), 281–300 (1997)
9. Breiman, L.: Random Forests. Mach. Learn. **45**(1), 5–32 (Oct 2001). https://doi.org/10.1023/A:1010933404324, https://doi.org/10.1023/A:1010933404324
10. Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R.: EXE: Automatically Generating Inputs of Death. ACM Transactions on Information and System Security (TISSEC) **12**(2), 1–38 (2008)
11. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the Really Hard Problems Are. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence. pp. 331–337. IJCAI'91 (1991)
12. Clarke Jr, E.M., Grumberg, O., Kroening, D., Peled, D., Veith, H.: Model Checking. MIT press (2018)
13. Clauset, A., Moore, C., Newman, M.E.J.: Hierarchical Structure and the Prediction of Missing Links in Networks. Nature **453**(7191), 98–101 (May 2008). https://doi.org/10.1038/nature06830
14. Coarfa, C., Demopoulos, D.D., San Miguel Aguirre, A., Subramanian, D., Vardi, M.Y.: Random 3-SAT: The Plot Thickens. Constraints **8**(3), 243–261 (Jul 2003). https://doi.org/10.1023/A:1025671026963
15. Cook, S.A.: The Complexity of Theorem-Proving Procedures. In: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing. pp. 151–158 (1971). https://doi.org/10.1145/800157.805047
16. Dolby, J., Vaziri, M., Tip, F.: Finding Bugs Efficiently With a SAT Solver. In: Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 195–204 (2007). https://doi.org/10.1145/1287624.1287653

17. Eén, N., Biere, A.: Effective Preprocessing in SAT Through Variable and Clause Elimination. In: Bacchus, F., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing. pp. 61–75. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
18. Fortunato, S., Barthélemy, M.: Resolution Limit in Community Detection. Proceedings of the National Academy of Sciences **104**(1), 36–41 (2007). https://doi.org/10.1073/pnas.0605965104
19. Friedrich, T., Krohmer, A., Rothenberger, R., Sutton, A.M.: Phase Transitions for Scale-Free SAT Formulas. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. p. 3893–3899. AAAI'17, AAAI Press (2017)
20. Giráldez-Cru, J.: Beyond the Structure of SAT Formulas. Ph.D. thesis, Universitat Autònoma de Barcelona (2016)
21. Giráldez-Cru, J., Levy, J.: A Modularity-Based Random SAT Instances Generator. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015. pp. 1952–1958 (2015), `http://ijcai.org/Abstract/15/277`
22. Granell, C., Gomez, S., Arenas, A.: Hierarchical Multiresolution Method to Overcome the Resolution Limit in Complex Networks. International journal of bifurcation and chaos **22**(07), 1250171 (2012)
23. Hoory, S., Linial, N., Wigderson, A.: Expander Graphs and Their Applications. Bulletin of the American Mathematical Society **43**(4), 439–561 (2006)
24. Kilby, P., Slaney, J., Thiebaux, S., Walsh, T.: Backbones and Backdoors in Satisfiability. In: Proceedings of the National Conference on Artificial Intelligence. vol. 3, pp. 1368–1373 (Jan 2005)
25. Lauria, M., Elffers, J., Nordström, J., Vinyals, M.: CNFgen: A Generator of Crafted Benchmarks. In: Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17). pp. 464–473 (Aug 2017). https://doi.org/10.1007/978-3-319-94144-8_18
26. Li, C., Chung, J., Mukherjee, S., Vinyals, M., Fleming, N., Kolokolova, A., Mu, A., Ganesh, V.: On the hierarchical community structure of practical sat formulas. arXiv preprint arXiv:2103.14992 (2021)
27. Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning Rate Based Branching Heuristic for SAT Solvers. In: Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing - SAT 2016. pp. 123–140 (2016). https://doi.org/10.1007/978-3-319-40970-2_9
28. Mateescu, R.: Treewidth in Industrial SAT Benchmarks. Tech. Rep. MSR-TR-2011-22, Microsoft (Feb 2011), `https://www.microsoft.com/en-us/research/publication/treewidth-in-industrial-sat-benchmarks/`
29. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining Computational Complexity from Characteristic 'Phase Transitions'. Nature **400**(6740), 133–137 (1999)
30. Mull, N., Fremont, D.J., Seshia, S.A.: On the Hardness of SAT with Community Structure. In: Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 141–159 (Jul 2016). https://doi.org/10.1007/978-3-319-40970-2_10
31. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69**(2) (Feb 2004). https://doi.org/10.1103/physreve.69.026113
32. Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G., Simon, L.: Impact of Community Structure on SAT Solver Performance. In: Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part

of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. pp. 252–268 (2014). https://doi.org/10.1007/978-3-319-09284-3_20

33. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
34. Platt, J.C.: Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In: Advances in Large Margin Classifiers. pp. 61–74. MIT Press (1999)
35. Ravasz, E., Somera, A.L., Mongru, D.A., Oltvai, Z.N., Barabási, A.L.: Hierarchical Organization of Modularity in Metabolic Networks. science **297**(5586), 1551–1555 (2002)
36. Samer, M., Szeider, S.: Backdoor Trees. In: Automated Reasoning. vol. 1, pp. 363–368. Springer (Jan 2008)
37. Samer, M., Szeider, S.: Fixed-Parameter Tractability. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS press, second edn. (Feb 2021)
38. SAT: The International SAT Competition. `http://www.satcompetition.org`, Accessed: 2021-03-06
39. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating Hard Satisfiability Problems. Artificial intelligence **81**(1-2), 17–29 (1996)
40. SHARCNET: SHARCNET: Graham Cluster. `https://www.sharcnet.ca/my/systems/show/114`, Accessed: 2021-03-06
41. Simon, H.A.: The Architecture of Complexity. Proceedings of the American Philosophical Society **106**(6), 467–482 (1962), `http://www.jstor.org/stable/985254`
42. Steel, R.G.D., Torrie, J.H.: Principles and Procedures of Statistics. McGraw-Hill (1960)
43. Szeider, S.: Algorithmic Utilization of Structure in SAT Instances. Theoretical Foundations of SAT/SMT Solving Workshop at the Simons Institute for the Theory of Computing (2021)
44. Williams, R., Gomes, C.P., Selman, B.: Backdoors To Typical Case Complexity. In: IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence. pp. 1173–1178 (2003), `http://ijcai.org/Proceedings/03/Papers/168.pdf`
45. Xie, Y., Aiken, A.: Saturn: A SAT-Based Tool for Bug Detection. In: Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005. pp. 139–143 (2005). https://doi.org/10.1007/11513988_13
46. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: Features for SAT. `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/` (2012), accessed: 2021-02
47. Zulkoski, E., Martins, R., Wintersteiger, C.M., Liang, J.H., Czarnecki, K., Ganesh, V.: The Effect of Structural Measures and Merges on SAT Solver Performance. In: Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming. pp. 436–452 (2018). https://doi.org/10.1007/978-3-319-98334-9_29
48. Zulkoski, E., Martins, R., Wintersteiger, C.M., Robere, R., Liang, J.H., Czarnecki, K., Ganesh, V.: Learning-Sensitive Backdoors with Restarts. In: Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming. pp. 453–469 (2018). https://doi.org/10.1007/978-3-319-98334-9_30