On the Hierarchical Community Structure of Practical SAT Formulas

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

[1] University of Waterloo, Waterloo, Canada
[2] Technion, Haifa, Israel
[3] University of Toronto, Toronto, Canada
[4] 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 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 [11], formal methods [14], security [18,46], and AI [9], thanks to their ability to solve large real-world instances with tens of millions of variables and clauses [40], notwithstanding the fact that the Boolean satisfiability (SAT) problem is known

[*] Joint first author

to be NP-complete and is believed to be intractable [17]. This apparent contradiction can be explained away by observing that the 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. Hence, this gap between theory and practice is rightly considered as 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 efficient because they somehow exploit the underlying structure in 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, namely, 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 over industrial instances. An example of such a parameter (resp. structure) is modularity (resp. 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 [45,49], backbones [30], treewidth, and branchwidth [1,39], among many others [39]; or have been used to prove complexity-theoretic bounds over randomly-generated classes of Boolean formulas such as clause-variable ratio (a.k.a., density) [16,41].

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 [29]).

Consequently, many attempts have been made at discovering correlative parameters that could form the basis of rigorous analysis [4,23]. Unfortunately, all 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

[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 [44].

instances have good community structure, i.e., high modularity [4], and that there is good-to-strong correlation between community structure and solver run time [34]. However, Mull et al. [31] later exhibited a dense 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 [15,37], 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 [43], and we expect this structure to be inherited by Boolean formulas modelling these systems.

Brief Overview of Results: Using a machine-learning (ML) classifier, we empirically demonstrate that the range of HCS-based parameter values taken by industrial instances is distinct from the range of values taken by randomly-generated or crafted instances. The accuracy of our classifier is very high, and we perform a variety of tests to ensure there is no over-fitting. We also show a good correlation between HCS-based parameter values and solver run time. From these two experiments, and in conjunction with theoretical analysis, we zero-in on three parameters as the most predictive, namely, leaf-community size, community degree, and the fraction of inter-community edges. To further understand their impact on solver run time, we perform several scaling studies which further strengthen our case that HCS parameters hold promise in explaining the power of CDCL SAT solvers over industrial instances.

Finally, we prove that HCS parameters have good theoretical properties. Specifically, we prove that linear expanders cannot be embedded in formulas with "good" HCS parameters (i.e., small community leaf size, small community degree, and small number of inter-community edges). Further, we show that as the range of parameter values improve, stronger varieties of expander graphs can be ruled out. Strong expansion properties of the underlying graph of a formula are one of the main tools used in proving proof complexity lower bounds.

Having said that, we also show that restricting certain subsets of these parameters, while avoiding previously known counterexamples, can be shown to allow for formulas that require large resolution refutations. However, the construction of these counterexamples is non-trivial, and indeed significantly more complicated than the counterexamples which suffice for community structure[31]. Nonetheless, we believe that hierarchy will eventually play an important role, alongside other parameters, in settling the long-standing and difficult question regarding the unreasonable performance of solvers over industrial instances.

Research Methodology: We also codify into a research methodology 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. These empirical tests include the following: first, there must exist strong statistical evidence (in the form of well-understood correlation measures and ML predictors) that witnesses the presence of the proposed structure (in terms of a range of proposed parameter values) in industrial instances and its absence from random or crafted ones; second, strong statistical evidence for correlation between sharply-defined good (resp. bad) values of parameters and low (resp. high) solver running time; third, independent parameter scaling that enables the isolation of the effects of each parameter. We believe that the combination of these requirements provides a strong basis for a correlative parameter to be considered worthy of further analysis.

Detailed List of Contributions[3]:

1. **Empirical Result 1 (HCS and Industrial Instances):** As stated above, we propose a set of parameters based on HCS of the variable-incidence graph (VIG) of Boolean formulas, and show that they are very effective in distinguishing industrial instances from random/crafted ones. In particular, we build a classifier that classifies SAT instances into the problem categories they belong to (e.g., industrial and randomly-generated). We show that our HCS parameters are robust in classifying SAT formulas into problem categories. The classification accuracy is approximately 99% and we perform a variety of tests to ensure there is no overfitting (See Section 5.2).
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 adjusted R^2 score of 0.85, much stronger than previous such results (see Section 5.3).
3. **Empirical Result 3 (Scaling Experiments of HCS Instances):** We empirically show, via scaling experiments, that HCS parameters such as depth and leaf-community size positively correlate with solving time. Finally, we empirically show 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 5.5).
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 [31], 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

[3]Instance generator, data, and full paper can be found here: `https://satcomplexity.github.io/hcs/`

degree, and inter-community edges) such that restricting them limits the size of embeddable expanders.

5. **Instance Generator:** Finally, we provide an HCS-based instance generator which takes input values of our proposed parameters and outputs a recursively generated formula that satisfies those values. This generator can be used to generate "easy" and "hard" formulas with different hierarchical structures.

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) model of Boolean formulas, 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 $x_1, \ldots, x_n$ 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 $P = \{V_1, \ldots, V_k\}$ of its vertices into communities that optimizes some measure capturing this intuition, for instance modularity [32], 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 [25]. Out of several equivalent such measures, the most convenient for HCS is the *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 [6].

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 [39] of the CDCL algorithm. Considerable work has already been done in attempting to identify exactly such a set of parameters [34]. 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 and possibly rigorous:

1. **Structure of Industrial vs. Random/Crafted Instances:** A prerequisite for a structure to be considered correlative is that industrial instances must exhibit a certain range of values for the associated parameters, while random and crafted instances must have a different distinct range of parameter values. 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) [34]. This could be demonstrated via a correlation experiment or by using a machine learning classifier that takes parameter values as input features and classifies instances as industrial, random, crafted etc. with high accuracy.
2. **Correlation between Structure and Solver Run Time:** Another requirement for a structure to be considered correlative is demonstrating that there is indeed 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. 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. 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 parameter is mergeability. In their work, Zulkoski et al. [48] 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 5.1, 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 [43], 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 smaller sub-communities. Although the notion of HCS has been widely studied [15,37], 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 [24]. 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.

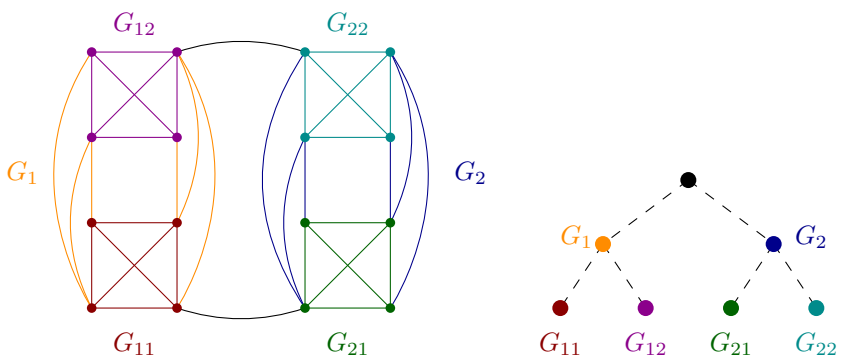


Fig. 1. A hierarchical decomposition (right) constructed by recursively maximizing the modularity of the graph (left).

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.
- For a given partition of a graph H, the *inter-community edges* are $E_{IC}(H) = \bigcup_{i,j} E(H_i, H_j)$, 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. In fact, in our experiments we use 49 different parameters of the HCS structure and then identify the most predictive ones among them. It turns out that leaf-community size (i.e., number of nodes in leaves of the HCS of a VIG), community degree, and inter-community edges are among the most predictive (See Section 5). In general, formulas with small community size, small community degree and few inter-community edges are easier to solve (See Section 5.6).

5 Empirical Results

5.1 Experimental Design

Choice of SAT Solver: We pre-process all formulas using standard techniques [28] and use MapleSAT as our CDCL solver of choice since it is a leading and representative solver [28].

Table 1. Summary of benchmark instances

Class Name	#SAT	#UNSAT	#UNKNOWN
`agile`	372	475	8
`crafted`	300	99	617
`crypto`	1052	355	3496
`random`	276	224	661
`verification`	197	2345	392

Choice of Benchmark Suite: In our experiments, we use a set of 10 869 instances from five different instance classes, which we believe is sufficiently large and diverse to draw sound empirical conclusions. We do 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 consider formulas which were previously used in the SAT competition from 2016 to 2018 [40]. Specifically, we take instances from five major tracks of the competition: Agile, Verification, Crypto, Crafted, and Random. We also generate additional instances for some classes: for verification, we scale the number of unrolls when encoding finite state machines for bounded model checking; for crypto, we encode SHA-1 and SHA-256 preimage problems; for crafted, we generate combinatorial problems using `cnfgen` [27]; and for random, we generate k-CNFs at the corresponding threshold CVRs for $k \in \{3, 5, 7\}$, again using `cnfgen`. A summary of the instances is presented in Table 1.

Computational Resources: For computing satisfiability and running time, we use SHARCNET's Intel E5-2683 v4 (Broadwell) 2.1 GHz processors [42], limiting the computation time to 5000 seconds[4]. For parameter computation, we still use SHARCNET, but we do not limit the type of processor because structural parameter values are independent of processing power.

Choice of Community Detection Algorithm: In our experiments, we use the Louvain method [8] to detect communities, taking the top-level partition as our communities. The Louvain method is typically more efficient and produces higher-modularity partitions than other known algorithms.

Parameters: We consider a small number of base parameters, in addition to HCS-based ones, to measure different structural properties of input VIGs. The base parameters are the number of variables, clauses, and binary clauses; the average and variance in the number of occurrences of each variable; the number of inter-community variables and inter-community edges; the numbers, sizes, degrees, and depths of the communities; modularity; and mergeability (see supplementary material[5] for a comprehensive list of parameter combinations).

[4] This value is the time limit used by the SAT competition.

[5] The supplemental material is available here as part of the full version of the paper: `https://satcomplexity.github.io/hcs/`

Table 2. Results for classification and regression experiments. For regression we report adjusted 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	0.848 ± 0.009
Top 5 features	`rootMergeability`	`rootInterEdges`
	`maxInterEdges/CommunitySize`	`lvl2Mergeability`
	`cvr`	`cvr`
	`leafCommunitySize`	`leafCommunitySize`
	`lvl2InterEdges/lvl2InterVars`	`lvl3Modularity`

5.2 HCS-based Category Classification of Boolean Formulas

It is conjectured that SAT solvers are efficient because they somehow exploit the underlying structure in industrial formulas. One may then ask the question whether our set of HCS parameters is able to capture the underlying structure that differentiates industrial instances from the rest, which naturally lends itself to a classification problem. Therefore, we build a multi-class Random Forest classifier to classify a given SAT instance into one of five categories: verification, agile, random, crafted, or crypto. Random Forests [10] are known for their ability to 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 use an off-the-shelf implementation of a Random Forest classifier implemented as `sklearn.ensemble.RandomForestClassifier` in scikit-learn [35]. We train 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 find that our classifier performs extremely well, giving an average accuracy score of 0.99 over multiple cross-validation datasets. Our accuracy does not depend on our choice of classifier. We find similar accuracy scores when we use C-Support Vector classification [36]. Further, we determine the five most important features used by our classifier. Since some of our features are highly correlated, we first perform a hierarchical clustering on the feature set based on Spearman rank-order correlations, choose a single feature from each cluster, and then use permutation importance [10] to compute feature importance. We present results in Table 2.

The robustness of our classifier indicates that HCS parameters are representative of the underlying structure of Boolean formulas from different categories. In the following section, we discuss an empirical hardness model for predicting the run time of the solver using the same set of features we used to classify instances into categories.

5.3 HCS-based Empirical Hardness Model

We use our HCS parameters to build an empirical hardness model (EHM) to predict the run time of a given instance for a particular solver (MapleSAT). Since the solving time is a continuous variable, we consider a regression model built using Random Forests. We use `sklearn.ensemble.RandomForestRegressor`, an implementation available in scikit-learn [35]. Before training our regression model, we remove instances which timed-out at 5 000 seconds as well as those instances that were solved almost immediately (i.e., zero seconds) to avoid issues with artificial cut-off boundaries. We then train our Random Forest model to predict the logarithm of the solving time using the remaining 1 880 instances, equally distributed between different categories.

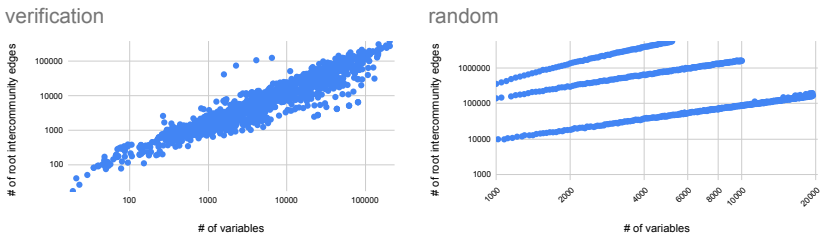
We observe that our regression model performs quite well, with an adjusted R^2 score of 0.84, which implies that in the training set, almost 84% of the variability of the dependent variable (i.e., in our case, the logarithm of the solving time) is accounted for, and the remaining 16% is still unaccounted for by our choice of parameters. Similar to category classification, we also look for the top five predictive features used by our Random Forest regression model using the same process. We list the parameters in Table 2.

As a check, we train our EHM on each category of instances separately. We find that the performance of our EHM varies with instance category. Concretely, agile outperforms all other categories with an adjusted 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 is shown by the instances in crypto, with a score of 0.48. This suggests that our choice 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 and it is not clear from our experiments (nor any previous experiments we are aware of) why crypto instances are hard for CDCL solvers.

Top-5 Parameters: The top features identified in our classification and regression experiments fall into five distinct classes of parameters: mergeability-based, modularity-based, inter-community edge based, CVR, and leaf-community size.

5.4 Scaling Studies of Industrial and Random Instances

In this experiment, we scaled the values of the top-5 formula parameters, identified by the classification and regression experiments given above, to get a fine-grained picture of the impact of these parameters on solver run time. The parameters CVR, mergeability and modularity have been studied by previous work. CVR is perhaps the most studied parameter among the three [12]. Zulkoski et al. [48] 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. [34] showed that modularity has good-to-strong correlation with solver run time. We more closely examine the remaining parameters: leaf-community size and inter-community edge based parameters.



(a) Verification **(b)** Random

Fig. 2. Log of number of root-level inter-community edges vs. log of number of variables, by instance class.

We next show that the range of values `rootInterEdges` takes for industrial instances is different from random instances, and we discuss the effect of scaling leaf-community size in Section 5.5. As we can see in Figure 2 (note that there are three separate data series in the random subplot, corresponding to random 3-, 5-, and 7-CNFs when ordered by growth rate), the number of inter-community edges at the root level grows proportionally to the number of variables of the formula, while the proportionality constant for random instances is at least an order of magnitude larger (two and three orders respectively for 5- and 7-CNFs).

5.5 Scaling via Instance Generator

In the experiments described here, we use an HCS instance generator to scale our HCS parameters individually while holding other parameters constant. We generate many instances with different HCS parameter values by specifying parameter values of CVR, power law parameter, hierarchical degree, depth, leaf-community size, inter-community edge density, inter-community variable density, and clause width. We note 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. [48] and Giráldez-Cru [22] for literature on the empirical behaviours of mergeability and power law respectively.

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. The generator is only intended to demonstrate the effect of varying our HCS parameters.

Results: We show that when fixing every other HCS parameters, increasing any of leaf-community size, depth and community degree increases the overall

hardness of the generated formula. This suggests that leaf-community size, depth and community degree are important HCS parameters to consider further.

5.6 Analysis of Empirical Results

The goal of our experimental work was to first ascertain whether HCS structure is witnessed in industrial instances, whether the parameter range is different between industrial and random/crafted, and finally whether there is any correlation with solver run time. On all three counts, the answer is a strong yes. In more detail, after identifying the top-5 parameters (namely, mergeability-based parameters, modularily-based parameters, CVR, leaf-community size, and inter-community edges) and based on the classification results, we can safely say that the HCS structure of industrial instances is empirically very different from that of random/crafted. In particular, industrial instances typically have small leaf-community size, high modularity, and low inter-community edges, while random/crafted have larger leaf-community size, low modularity, and a very high number of inter-community edges. Further, the correlation with solver run time is strong — much stronger than previously proposed parameters.

6 Theoretical Results

We present the following theoretical results that further strengthen the idea that HCS-based parameters (especially ones identified above in analysis of experimental results) may play a role in the final analysis of understanding of why SAT solvers are efficient over industrial instances. Based on our experimental work, we narrow down the most predictive HCS parameters to be leaf-community size, community degree, and inter-community edges. All of these parameters play a role in the theorems below. For a formula to have "good" HCS, we restrict the parameter value range as follows: the graph must exhibit small $O(\log n)$ leaf-community size, constant community degree, and have a small number of inter-community edges in each community.

Ideally, we would like to be able to theoretically establish a parameterized upper bound via HCS parameters. Unfortunately, our current state of understanding does not yet allow for that. A step towards such a result would be to show that formulas with good HCS (and associated parameter value ranges) form a family of instances such that typical methods of proving resolution lower bounds, such as those exploiting expansion properties [6], do not apply to them. More precisely, we would like to show that formulas with good HCS do not have VIGs which are expanders or have large expanders embedded in them, since most resolution lower bounds rely on expansion properties. In particular, for formulas with low width, edge expansion is closely related to boundary expansion, the parameter of choice for proving resolution lower bounds. With this in mind, we state several positive and negative results.

First, we make the observation that we restrict leaf communities of the HCS of a graph to have size $O(\log n)$ to avoid counterexamples where large hard

formulas are emdedded within them [31]. Indeed, practical instances having small leaf communities is supported by our experimental results. Further, we observe 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 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 bounded by $O(\log n)$ and the number of inter-community edges grows slowly. From this and the theorem above, we can show that graphs with very good HCS do not contain linear-sized expanders.

Hierarchical vs. Flat Modularity: It is well-known that modularity suffers from a *resolution limit* and cannot detect communities smaller than a certain threshold [19], and it is also known that HCS can avoid this problem in some instances [8]. 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

7.1 Correlative Parameters and Empirical Models

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 [31], i.e., there exist formulas that have good community structure and yet are hard for solvers. Although we identify this parameter as correlative, FPT results exist for a related concept, *hitting community structure* and *h-modularity* [21]. However, there is no evidence that hitting community structure is witnessed in industrial instances.

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 [7], making them intractable for current solvers.
SATzilla: SATzilla uses 138 disparate parameters [47], some of which are probes aimed at capturing a SAT solver's state at runtime, to predict solver running time. Unfortunately, there is no evidence that these parameters are amenable to theoretical analysis.

7.2 Rigorous Parameters

Clause-Variable Ratio (CVR): Cheeseman et al. [12] 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 [20]. Satisfiability thresholds are poorly defined for industrial instances, and Coarfa et al. [16] 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 [29] 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 [48].
Backdoors: In theory, the existence of small backdoors [45,38] should allow CDCL solvers to solve instances quickly, but empirically backdoors have been shown not to strongly correlate with CDCL solver run time [26]. To the best of our knowledge, the literature does not contain any strong empirical support for backdoors alone as a predictive parameter.

8 Conclusions and Future Work

In this paper, we propose hierarchical community structure as a correlative parameter for explaining the power of CDCL SAT solvers over industrial instances. 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. We further identify the following core HCS parameters that are the most predictive, namely, leaf-community size, modularity, and inter-community edges. Indeed, these same parameters also play a role in our subsequent theoretical analysis. 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. 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. Beame, P., Pitassi, T.: Simplified and improved resolution lower bounds. In: 37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996. pp. 274–282. IEEE Computer Society (1996). https://doi.org/10.1109/SFCS.1996.548486
6. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow—resolution made simple. Journal of the ACM (JACM) **48**(2), 149–169 (2001)
7. 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
8. 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
9. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial intelligence **90**(1-2), 281–300 (1997)
10. 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`
11. 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)
12. 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)
13. Chvátal, V., Szemerédi, E.: Many hard examples for resolution. J. ACM **35**(4), 759–768 (1988). https://doi.org/10.1145/48014.48016
14. Clarke Jr, E.M., Grumberg, O., Kroening, D., Peled, D., Veith, H.: Model checking. MIT press (2018)
15. 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
16. 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

17. 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
18. 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
19. 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
20. 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)
21. Ganian, R., Szeider, S.: Community structure inspired algorithms for SAT and #SAT. In: Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing - SAT 2015. pp. 223–237 (2015). https://doi.org/10.1007/978-3-319-24318-4_17
22. Giráldez-Cru, J.: Beyond the Structure of SAT Formulas. Ph.D. thesis, Universitat Autònoma de Barcelona (2016)
23. 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
24. 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)
25. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin of the American Mathematical Society **43**(4), 439–561 (2006)
26. 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)
27. 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
28. 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
29. 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/
30. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity from characteristic 'phase transitions'. Nature **400**(6740), 133–137 (1999)
31. 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

32. 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
33. Newman, M.E.: Modularity and community structure in networks. Proceedings of the national academy of sciences **103**(23), 8577–8582 (2006)
34. 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
35. 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)
36. 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)
37. 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)
38. Samer, M., Szeider, S.: Backdoor trees. In: Automated Reasoning. vol. 1, pp. 363–368. Springer (Jan 2008)
39. 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)
40. SAT: The International SAT Competition. `http://www.satcompetition.org`, Accessed: 2021-03-06
41. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating hard satisfiability problems. Artificial intelligence **81**(1-2), 17–29 (1996)
42. SHARCNET: Sharcnet: Graham cluster. `https://www.sharcnet.ca/my/systems/show/114`, Accessed: 2021-03-06
43. Simon, H.A.: The architecture of complexity. Proceedings of the American Philosophical Society **106**(6), 467–482 (1962), `http://www.jstor.org/stable/985254`
44. 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)
45. 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`
46. 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
47. 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
48. 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

49. 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

Appendix

A Parameters Used for Classification and Regression

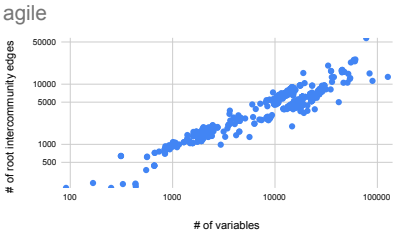
Although we only propose a small number of base HCS parameters, they can be combined in many ways to capture different structural information about the instance. The following table is a comprehensive list of all the computed features which were used for our machine learning classification and regression results. This includes our proposed HCS parameters as well as other parameters which we believe to be important, such mergeability.

Name	Description
`numVars`	the number of distinct variables in the formula
`numClauses`	the number of distinct clauses in the formula
`CVR`	`numClauses / numVars`
`dvMean`	the average number of times a variable appears
`dvVariance`	the variance in the number of times a variable appears
`numCommunities`	total number of communities
`numLeaves`	total number of leaf-communities
`avgLeafDepth`	average leaf depth
`depthMostLeaves`	the depth with the most leaf-communities
`rootInterVars`	the number of inter-community variables at the root level
`lvl2InterVars`	the average number of inter-community variables at depth 2
`lvl3InterVars`	the average number of inter-community variables at depth 3
`rootInterEdges`	the number of inter-community edges at the root level
`lvl2InterEdges`	the average number of inter-community edges at depth 2
`lvl3InterEdges`	the average number of inter-community edges at depth 3
`rootDegree`	the number of communities at the root level
`lvl2Degree`	the average number of communities at depth 2
`lvl3Degree`	the average number of communities at depth 3
`maxDegree`	the maximum degree over all levels
`rootModularity`	the modularity of the graph at the root level
`lvl2Modularity`	the average modularity at depth 2
`lvl3Modularity`	the average modularity at depth 3
`maxModularity`	the maximum modularity over all levels
`rootMergeability`	the mergeability score between all variables
`lvl2Mergeability`	the average mergeability score of a community at depth 2
`lvl3Mergeability`	the average mergeability score of a community at depth 3
`maxMergeability`	the maximum mergeability over all levels
`lvl2CommunitySize`	the average number of variables in a community at depth 2
`lvl3CommunitySize`	the average number of variables in a community at depth 3
`leafCommunitySize`	the average number of variables in a leaf-community

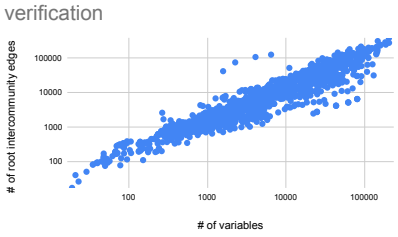
Name	Description
numLeaves / numCommunities	
rootInterEdges / rootInterVars	
lvl2InterEdges / lvl2InterVars	
lvl3InterEdges / lvl3InterVars	
max(interEdges / interVars)	the maximum interEdges / interVars ratio over all levels
rootInterEdges / rootCommunitySize	
lvl2InterEdges / lvl2CommunitySize	
lvl3InterEdges / lvl3CommunitySize	
max(interEdges / communitySize)	the maximum interEdges / communitySize ratio over all levels
rootInterVars / rootCommunitySize	
lvl2InterVars / lvl2CommunitySize	
lvl3InterVars / lvl3CommunitySize	
max(interVars / communitySize)	the maximum interVars / communitySize ratio over all levels
rootInterEdges / rootDegree	
lvl2InterEdges / lvl2Degree	
lvl3InterEdges / lvl3Degree	
rootInterVars / rootDegree	
lvl2InterVars / lvl2Degree	
lvl3InterVars / lvl3Degree	

B Scaling Studies: Inter-community Edges vs. Number of Variables

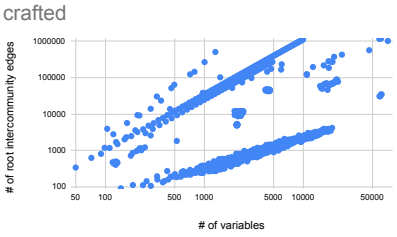
In our scaling studies, we change the number of variables and observe that the scaling behaviour of the total number of inter-community edges in the formula is different for different classes of instances. Plots of this scaling behaviour for each of the instance classes considered in this work are presented here:



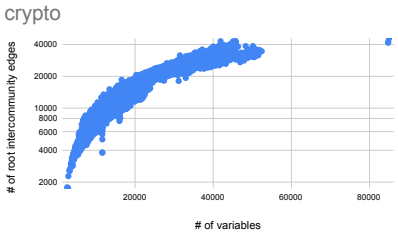
(a) Agile



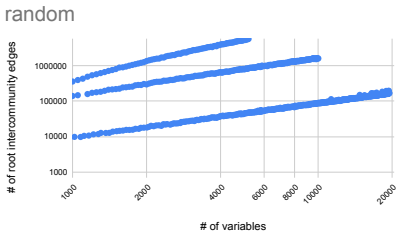
(b) Verification



(c) Crafted



(d) Crypto



(e) Random

C Detailed Theoretical Results

C.1 Embedding expanders in HCS graphs

It would be nice if good HCS decomposition would disallow large expanders embeddable in the graph. Let us consider graphs with the following restrictions on HCS.

1. Small leaf size: all leaves are of size at most $O(\log n)$.
2. Bounded community degree: for every community at every level, the number of immediate sub-communities is bounded by $O(\log n)$.
3. Small number of inter-community edges (the smaller the better).

The reason for the small leaf size condition is to avoid Mull, Fremont, and Seshia [31] counterexample. The second condition, bounded community degree, avoids counterexamples in section C.3. The final property is used in the following theorem.

In the following, we use G to refer both to a family of graphs as well as a specific representative of that family (the reason for the family is to be able to talk about asymptotic behaviour of parameters).

Claim. Suppose a graph G with $|G| = n$ has a community decomposition into $C_1, \ldots, C_\ell$ maximizing the modularity (flat, single-level decomposition). Let H be a subgraph of G, with $|H| = f(n)$ for some $f(n) \in O(n)$. Then either H is not an expander, or the number of inter-community edges is $\Omega(f(n))$, or there exists a community containing a subgraph of H of size at least $f(n)(1 - o(1))$.

Proof. Suppose that H is an expander graph of size $f(n)$ with edge expansion $h(H) = \alpha$, and let S be the largest subset of H which is within a single C_i (wlog, say $S \subseteq C_1$). Suppose that $|S| = \delta \cdot f(n)$ for some constant δ. Then the number of edges out of C_1 is at least $\alpha \cdot \min\{\delta, 1-\delta\} f(n)$, and thus the the total number of inter-community edges is $\Omega(f(n))$. If $|S| = o(f(n))$, then it is possible to split communities $C_1 \ldots C_t$ into 2 sets with roughly the same number of vertices of H on each side (up to a subconstant factor); the number of edges of H going across this cut should be close to $\alpha f(n)$, again $\Omega(f(n))$. □

Theorem 3. *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 $\Omega(f(n))$. Then G does not contain an expander of size $f(n)$ as a subgraph.*

Proof. Let H be a subgraph of G of size $f(n)$. Let C be the smallest sub-community in the hierarchical decomposition of G containing at least $(1-o(1))$-fraction of H. Note that if H is an expander, then any subgraph of H on $1-o(1)$ vertices is also an expander.

First, since $|H| \in \omega(poly(\log n))$, C cannot be a leaf. Therefore, C it will be partitioned into sub-communities $C_1 \ldots C_\ell$ by the hierarchical decomposition. Since we assumed that C is the smallest community containing $(1-o(1))$ fraction

of H, each sub-community of C can contain at most a constant fraction of H. Then by claim C.1, since $|C| \in \Omega(f(n))$ and so by assumption the number of inter-community edges in this decomposition is bounded by $o(f(n))$, H is not an expander. □

One of the main measures of "quality" of HCS is modularity value of the decompositions throughout the hierarchy: the higher values correspond to better structural properties. Let us start by relating modularity to the number of inter-community edges. Note that while both the definition of expander and modularity are essentially relying on estimating how far the number of edges across the "best" partition is from the expected value, the dependence on the size of the communities is somewhat different.

First, to simplify our proof, consider decomposition into 2 communities; we can do it thanks to the following claim.

Claim. [33] Suppose that $C_1 \dots C_t$ is a decomposition of C optimizing modularity. Then for some $S = \{C_{i_1}, \dots, C_{i_\ell}\}$ the partition into S and $\overline{S}$ maximizes modularity among 2-partitions.

Corollary 1. *For any graph G, the best modularity of a 2-partition is a lower bound for the modularity of an optimal partition, with edges in the the best 2-partition a subset of inter-community edges in the optimal partition.*

To simplify notation, for a set $S \subset V$, define $vol(S) = \Sigma_{v \in S} deg(v)$. Let $e_{out}(C)$ be the set of edges leaving a community C_i, so $\Sigma_{C \in P} e_{out}(C)$ is the set of all intercommunity edges in a partition P. Now, we can restate the formula for modularity of a partition P as follows, using the fact that $\Sigma_{C \in P} vol(C) = 2|E|$:

$$Q(P) = 1 - \frac{1}{2|E|} \Sigma_{C \in P} e_{out}(C) - \frac{1}{4|E|^2} \Sigma_{C \in P} (vol(C))^2$$

The optimal Q of a graph G is then $Q = Q(G) = \max_P Q(P)$. Let $Q_2 = Q_2(G) = \max_{P, |P|=2} Q(P)$ (that is, optimal modularity over 2-partitions). Note that $Q(G) \geq Q_2(G)$ for any G. Let P be a partition of G into a set S and its complement $\overline{S}$; we will always assume, without loss of generality, that $|S| \leq |\overline{S}|$. Let $|V| = n$ and $|E| = m$ (not to be confused with the number of clauses in the formula for which G is a VIG). Then

$$\begin{aligned} Q(\{S, \overline{S}\}) &= 1 - e_{out}(S)/2m - e_{out}(\overline{S})/2m - (vol(S)/2m)^2 - (vol(\overline{S})/2|E|)^2 \\ &= 1 - e_{out}S/m - \frac{1}{4m^2}(vol(S)^2 + (2m - vol(S))^2) \\ &= vol(S)/m - vol(S)^2/2m^2 - e_{out}(S)/m \\ &= \frac{vol(S)}{m}(1 - vol(S)/2m) - e_{out}(S)/m \end{aligned}$$

From there, we get that $e_{out}(S) = vol(S)(1 - vol(S)/2m) - Q(S)m$. Note that since $Q_2 = max_S Q(\{S, \overline{S}\})$, $e_{out}(S) = vol(S)(1 - vol(S)/2m) - Q_2 m$. Therefore, if $vol(S)(1 - vol(S)/2m) - Q_2 m = o(f(n))$, then so is $e_{out}(S)$.

C.2 HCS Avoids the Resolution Limit in Some Graphs

We prove Theorem 2, restated below more formally.

Theorem 2 (restated). *There exists a graph G whose natural communities are of size $\log n$ and correspond to $\mathcal{P}_h(G)$, while $\mathcal{P}(G)$ consists of communities of size $\sqrt{n/\log^3 n}$.*

We use as the separating graph G the ring of cliques example of [19], which can be built as follows. Start with a collection of $q = n/c$ cliques $C_1, \ldots, C_q$, each of size $c = \log n$. Fix a canonical vertex v_i for each clique C_i. Add edges between v_i and v_{i+1}, wrapping around at q. The number of vertices is n and the number of edges is $m = \frac{n}{c}\binom{c}{2} + \frac{n}{c} = n(c+1)/2 + o(n)$. The degree of most vertices is $c-1$, except for the canonical vertices which have degree $c+1$. The natural partition of G into communities is the set of cliques $C_1, \ldots, C_q$.

We say that a subgraph of G preserves the cliques if for every clique, it contains either all or none of its vertices. We say that a partition of G preserves the cliques if every element of the partition preserves the cliques.

To prove that $\mathcal{P}_h(G) = \{C_1, \ldots, C_q\}$ we need the following two observations.

Lemma 1. *Let H be a subgraph of H that preserves the cliques. Then any partition of H optimizing subgraph modularity preserves the cliques.*

Proof. Let $V_1, \ldots, V_k$ be a partition and assume that clique C_i contains vertices in at least two different sets V_j. We claim that the partition $V_1 \setminus C_i, \ldots, V_k \setminus C_i, C_i$ has greater modularity and that the number of split cliques decreases, therefore iterating this procedure until no clique is split concludes the lemma.

To prove the claim, let $U_j = V_j \cap C_i$ and note that the change in modularity is at least

$$
\begin{aligned}
2m\Delta Q &\geq -4\left(1 - \frac{(c+1)^2}{2m}\right) + \sum_{j \neq j'} \sum_{u \in U_j} \sum_{v \in U_{j'}} \left(1 - \frac{d(u)d(v)}{2m}\right) && (2)\\
&\geq -4 + \sum_{j \neq j'} |U_j||U_{j'}| \left(1 - \frac{(c+1)^2}{2m}\right) && (3)\\
&\geq -4 + \frac{1}{2} \sum_{j \neq j'} |U_j||U_{j'}| && (4)\\
&\geq -4 + \frac{1}{4} \sum_{j \neq j', |U_j|>0, |U'_j|>0} |U_j| + |U_{j'}| && (5)\\
&\geq -4 + c/4 > 0 && \square
\end{aligned}
$$

Lemma 2. *Let H be a subgraph of H that preserves the cliques. If H contains $q \geq 2$ cliques, then any partition of H optimizing subgraph modularity contains at least 2 elements.*

Proof. Let V_1, V_2 be the partition where the first half of the cliques in H are contained in V_1 and the rest in V_2. The change in modularity with respect to the singleton partition is at least

$$2m\Delta Q \geq -2\left(1 - \frac{(c+1)^2}{2m}\right) + \sum_{u \in V_1} \sum_{v \in V_2} \frac{d(u)d(v)}{2m} \tag{6}$$

$$\geq -2 + (cq/2)^2 \frac{(c-1)^2}{2m} = -2 + \frac{n^2c^2}{9nc} > 0 \qquad \square$$

We can now compute $\mathcal{P}_h(G)$ by induction, using the induction hypothesis that every node of the hierarchical tree is a subgraph that preserves the cliques. The root is the whole graph G and trivially preserves the cliques. Every node of the hierarchical tree preserves the cliques, then by Lemma 1 all its children preserve the cliques. This implies that all of the leaves preserve the cliques. However, a leaf cannot consist of more than one clique, otherwise it would contradict Lemma 2. It follows that all the leaves are single cliques as we wanted to show.

Next we prove that the partition optimizing modularity has large elements, for which we need the following observations.

Let $V_1, \ldots, V_k$ be a partition that preserves the cliques. We define the operation "move clique a to position b" as follows. Assume $a < b$. For $a \leq i < b$ we assign clique i to the set containing clique $i+1$, and clique b to the set containing clique a. Analogous if $a > b$.

Lemma 3. *The modularity of G is maximized by a partition of contiguous cliques.*

Proof. By Lemma 1 the partition consists of unions of cliques. Assume a set contains non-contiguous cliques. Moving two intervals of cliques next to each other increases the modularity, therefore we can keep repeating this procedure until the sets only consist of contiguous intervals. $\square$

In what follows we assume that n is large enough for it not to make it a difference when we treat variables as being continuous when they are in fact discrete.

Lemma 4. *The modularity of G is maximized by a partition of equal-sized elements.*

Proof. By Lemma 3 the partition consists of intervals of cliques. Assume interval V_a is smaller than interval V_b. Let C_i be an endpoint of V_b. Then assigning clique C_i to V_a and moving it next to an endpoint of V_a increases the modularity, therefore we can repeat this procedure until the sets are balanced. $\square$

Lemma 5. *The modularity of G is maximized by a partition with elements of size $\sqrt{n/\log^3 n}$.*

Proof. By Lemma 4 the partition consists of equal-sized intervals of cliques. Let k be the size of the partition and let q be the number of cliques in each block. The modularity is

$$2mQ = k\Bigg(qc(c-1) + 2(q-1) - \frac{1}{2m}\Big(q^2(c+1)^2 + \tag{7}$$

$$+ q(q(c-1))(c+1)(c-1) + (q(c-1))^2(c-1)^2\Big)\Bigg) \tag{8}$$

$$= (n/cq)\left(q(c^2-c+2) - 2 - \frac{q^2}{2m}\left((c+1)^2 + (c-1)^2(c+1) + (c-1)^4\right)\right) \tag{9}$$

$$= (n/c)(c^2-c+2) - 2n/cq - \frac{nq}{2mc}\left((c+1)^2 + (c-1)^2(c+1) + (c-1)^4\right) \tag{10}$$

which is a function of the form $a_0 - a_1 q^{-1} - a_2 q$ and is maximized when its derivative is 0 at point

$$q = \sqrt{\frac{a_1}{a_2}} \geq \sqrt{\frac{2n/c}{nc^3/2m}} \geq \sqrt{\frac{2n}{c^3}} \tag{11}$$

□

This completes the proof of Theorem 2.

C.3 Justification for Parameter Choices

For a formula to have "good" HCS, we require that a number of parameters fall within appropriate ranges. One could hope that a single parameters of HCS might be sufficient in order to guarantee tractability, while also capturing a sufficiently large set of interesting instances. A natural candidate parameter would be *high smooth modularity*: The modularity of the VIG is large, and the decrease in modularity from the parent to a non-leaf child in the HCS is sufficiently bounded. This captures the recursive intuition of HCS: each community should either be a leaf or should have a good partition into communities; this is closely related to the average modularity at each level, which is a parameter used in our experiments. However, in this section we show that if we are after provable tractability, it is unlikely that a single parameter of HCS will suffice. In doing so, we motivate using an ensemble of parameters as we have done in our experiments, as well as justify why we have chosen many of the parameters that we have.

Formally, we show that combinations of the following ranges of parameters admit formulas which are exponentially hard to refute in resolution.

- *High Root Modularity*: The modularity of G is sufficiently large.
- *High Smooth Modularity*: The modularity of G is sufficiently large and the modularity of every non-leaf node is bounded below by a function of its level.
- *Bounded Leaf Size*: Each of the leaf-communities of the optimal HCS decomposition is sufficiently small — in particular, of size $O(\log n)$.

- *High Depth*: There is a sufficiently deep path in the optimal HCS decomposition.
- *Minimum Depth*: There is a lower bound on the depth of every path in the optimal HCS decomposition.

One finding that we would like to highlight is that in order to enure that the formula is tractable, the leaf-communities of the optimal HCS decomposition cannot be both large (of size $\omega(\log n)$) and unstructured; indeed, HCS says nothing about the structure of the leaf-communities.

Finally, we note that we can still construct hard formulas which have "good" HCS parameters. However, the construction of these formulas is highly contrived and non-trivial and we do not see a way to simplify them. Indeed, they are far more contrived than the counterexamples to modularity given by [31]. We take this as empirical evidence that instances with good HCS avoid far more hard examples than formulas with high modularity.

Throughout, it will be convenient to make use of the highly sparse VIGs provided by random CNF formulas. For positive integer parameters m, n, k, let $\mathcal{F}(m, n, k)$ be the uniform distribution on formulas obtained by picking m k-clauses on n variables uniformly at random with replacement. It is well known that for any k, the satisfiability of this $F \sim \mathcal{F}(m, n, k)$ is controlled by the clause density $\Delta_k := m/n$: there is a threshold of Δ_k after which $F \sim \mathcal{F}(m, n, k)$ becomes unsatisfiable with high probability. Furthermore, random k-CNF formulas near this threshold ($\Delta_k = O(2^k)$ suffices) require resolution refutations of size 2^{n^ε} for some constant $\varepsilon > 0$ with high probability [13,5]. The VIG of such a formula is sparse (it has $\Theta(n)$ edges) and with high probability the maximum degree is $O(\log n)$. Furthermore, with high probability F is expanding, and therefore the edges are distributed roughly uniformly throughout the VIG.

For brevity, our arguments section will be somewhat informal; however, it should be clear how to formalize them.

Root Modularity. Mull, Fremont, and Seshia [31] proved that having a highly modular VIG does not suffice to guarantee short resolution refutations. To do so, they extended the lower bounds on the size of resolution refutations of random k-CNF formulas [13,5] to work for a distribution of formulas whose VIGs have high modularity, thus showing the existence of a large family of hard formulas with this property.

A much simpler proof of their result — albeit with slightly worse parameters — can be obtained as follows: Let $F \sim \mathcal{F}(m, n, k)$ with k, m set appropriately so that F is hard to refute in resolution with high probability, and let F' be the formula obtained by taking t copies of F on disjoint sets of variables. It can be checked that the modularity of the partition of the VIG of F' which has t communities, one corresponding to each of the copies of F has modularity $1 - o(1/t)$. Setting t sufficiently large, we obtain a formula whose VIG has high modularity. As each copy of F is on distinct variables, refuting F' is at least as hard as refuting F. Thus, a lower bound of $2^{\Omega(n^\varepsilon)}$ for some constant $\varepsilon > 0$ follows from the known lower bounds on refuting F in resolution. If we let $v = nt$

be the number of variables of F' then this lower bound is of the form $2^{\Omega((v/t)^{\varepsilon})}$ and is superpolynomial provided $t = o(n/\log n)$. This argument also applies to hierarchical community structure.

Observe that each leaf-community of F' is a formula F on v/t variables which is hard to refute in resolution. This shows that if we want to ensure polynomial-size resolution proofs, then we cannot allow the leaf communities of the HCS decomposition to be both unstructured and large (of size $\omega(\log n)$). The simplest way to avoid this is to require the size of the leaf-community to be bounded by $O(\log n)$. However, in a later paragraphs we will show that this restriction is not sufficient on its own.

Root Modularity and Maximum Depth The previous example can be modified in order to rule out requiring an *upper bound on the depth* and *high smooth modularity*. We will use the simple observation the at the optimal community structure decomposition of t disjoint cliques on n vertices has t communities, one for each clique. Furthermore, HCS will not decompose any of these cliques, and therefore the optimal HCS has a single level and maximum depth assumption.

Let F' be the formula constructed in the previous example. We will modify each copy of F so that its VIG is a clique K_n. Let $i \in [t]$, and for the ith copy of F do the following: pick a variable $\hat{x}_i \neq v_i^*$ and note that there is a direction $\alpha_i \in \{0, 1\}$ in which it can be set such that the complexity of refuting $F \restriction (\hat{x} = \alpha_i)$ in resolution is at most half the complexity of refuting F. Add an arbitrary clause of length n containing every variable in F such that $\hat{x}$ occurs positively if $\alpha_i = 1$ and negatively otherwise. Observe that the VIG of F is now a clique.

After this process, the VIG of F' has t disjoint cliques. It remains to see that F' is still hard to refute in resolution. This follows because applying the restriction which sets $\hat{x}_i = \alpha_i$ for all $i \in [t]$ leaves us with t disjoint copies of F' which require $2^{\Omega(n^{\varepsilon})}$ size resolution refutations, and because resolution complexity is closed under restriction.

Smooth and Bounded Leaf Size. Next, we show that requiring the optimal HCS decomposition to have *high smooth modularity* and to have *bounded leaf size* is not sufficient in order to ensure small resolution refutations. Let $F \sim \mathcal{F}(m, n, 3)$ be a random 3-CNF formula with $m = \Theta(n)$ set so that F is hard to refute in resolution whp. Let G be the VIG of F, which has $\Theta(n)$ edges and degree $O(\log n)$ whp. Let $p = O(\log n)$ and K_p be a clique on p vertices. Let $2 \leq t \leq p$ be any integer and let F_K be any t-CNF formula on p variables such that every variable occurs in some clause; this requires at most p^2 clauses. Observe that the VIG of F_K is K_p. Furthermore, observe that F_K is satisfied by any truth assignment that sets at least $p - t + 1$ variables to true.

Using K_p and G we construct a family of formulas that is hard to refute in Resolution and whose VIG has the desired properties. Let G' be the rooted product of G and K_p, that is let v^* be an arbitrary vertex of K_p, create n copies of K_p, and identify the ith vertex of G with the vertex v^* of the ith copy of K_p.

Next, we show that because G is sparse and each K_p is a clique, G' has modularity which tends to 1 with n.

Lemma 6. *Let G be a graph of order n, size $m = \Theta(n)$, and degree $O(\sqrt{n})$; let K_p be a clique on p vertices with $p = \omega(1)$ and $p = o(n)$; and let G' be the rooted product of G and K_p. Then $Q(G') = 1 - o(1)$.*

Proof. Let $m' \leq m + np^2/2$ be the number of edges of G'. Consider the partition P given by the n copies of K_p. We have

$$Q(P) \geq \frac{np^2}{2m'}\left[1 - \frac{(d+p)^2}{2m'}\right] \geq \left[1 - \frac{m}{m'}\right]\left[1 - \frac{(d+p)^2}{2m'}\right] \tag{12}$$

and $m/m' = o(1)$ because $p = \omega(1)$, while $(d+p)^2/2m' = o(1)$ because $d = O(\sqrt{n})$ and $p = o(n)$. Hence $Q(G') \geq Q(P) = 1 - o(1)$. □

As well, by a similar argument to the proof of Lemma 1, the optimal HCS decomposition will preserve cliques at each level (i.e. none of the cliques K_p will be decomposed during the HCS decomposition. Furthermore, by an argument similar to Lemma 5, each node in the HCS decomposition will have too many children. Finally, observe that because the variables of the random CNF formula F are distributed uniformly, the non-clique edges in G will be distributed approximately uniformly between the cliques. Taken together, this implies that the modularity of every non-leaf node in the HCS cannot decrease too much compared to the modularity of its parent. In particular, we satisfy the high smooth modularity condition.

It remains to show that we can construct a formula with the same VIG which is hard to refute in resolution. Let F' be the formula obtained in the same way: begin with the formula F. Make n copies of F_K and choose some vertex v_i^* from the ith copy of F_k. Identify v_i^* with x_i, the ith variable of F. Observe that the VIG of F' is exactly G'. It remains to argue that F' is hard to refute in Resolution. Let V_i be set of variables on which the ith copy of F_K depends. Let $\rho \in \{0, 1, *\}^{pn}$ be the restriction which sets all variables in $V_i \setminus v_i^*$ to true for all $i \in [n]$ and does not set each v_i^* (that is $\rho(v_i^*) = *$ for all v_i^*). We claim that $F' \restriction \rho = F$. Indeed, each clause in each F_K depends on at least two variables, of which at least one is set to true because $t \geq 2$. The only variables on which F depends are v_i^* for $i \in [n]$, which have been left untouched; thus $F \restriction \rho = F$. The lower bound on F' follows from the lower bound on $2^{\Omega(n^\varepsilon)}$ lower bound on F together with the fact that the complexity of resolution proofs is closed under restriction.

Deep, Smooth, Bounded Leaf Size. We extend the previous example to show that if, in addition to the previous restrictions, we require the HCS decomposition have *high depth*, then this is still insufficient to guarantee short resolution proofs. To see this, let F' be the formula constructed in the previous example, and let F'' be a formula containing only positive literals and whose VIG satisfies smooth HCS, bounded leaf-community size, and whose optimal

HCS decomposition has a sufficiently deep root-to-leaf path. Note that such a formula F'' exists if and only if there exists a formula satisfying these properties with no restriction on the polarity of the literals. Therefore, we can assume the existence of F'', as otherwise this set of parameters does not capture any instances and is therefore not useful.

We claim that the VIG corresponding to $F' \wedge F''$ will satisfy their shared properties. First, observe that $F' \wedge F''$ has high modularity: the modularity of both F' and F'' is high by assumption and therefore the partition which is the union of the maximum modularity partitions of F' and F'' will have high modularity. Furthermore, because F' and F'' are disjoint, the HCS decomposition will never never put (a part of) both of these formulas in the same community. Thus, the maximum modularity partition is the union of the maximum partitions of F' and F''. From then onwards, because F' and F'' are smooth, the HCS decomposition of $F' \wedge F''$ will be smooth. As well, because F'' has a deep decomposition, this implies that the decomposition of $F' \wedge F''$ will also contain a deep path. Finally, because F'' is satisfiable and F' requires exponential length refutations in resolution, $F' \wedge F''$ does as well.