MDPI

Article

The Self-Similarity of Industrial SAT Instances

Carlos Ansótegui [1], Maria Luisa Bonet [2], Jesús Giráldez-Cru[3] and Jordi Levy [4,*]

1 DIEI, Universitat de Lleida (UdL), Spain; carlos@diei.udl.cat
2 LSI, Universitat Politècnica de Catalunya (UPC), Spain; bonet@cs.upc.edu
3 DaSCI, DECSAI, Universidad de Granada (UGR), Spain; jgiraldez@ugr.es
4 Artificial Intelligence Research Institute (IIIA-CSIC), Spain; levy@iiia.csic.es
* Correspondence: levy@iiia.csic.es; Tel.: +34 935809570

Version October 26, 2020 submitted to Mathematics

Abstract: In the last years, we have witnessed a remarkable progress of algorithms solving Boolean satisfiability (SAT). The success of these algorithms has been especially relevant in a large number of industrial or real-world applications, for which these SAT solvers are nowadays an essential core part of their solving processes. Interestingly enough, these applications include a very diverse and heterogeneous range of domains, such as hardware verification, planning, and cryptography, among others. Unfortunately, the reasons of the good performance of these solvers on this variety of industrial benchmarks are not completely understood yet. Since SAT solvers' efficiency is fundamental in various domains, obtaining a better understanding of these algorithms and the reasons of their good performance is crucial.

In order to shed light on this question, SAT solvers are often viewed as *complex systems* with many interconnected components (e.g., conflict analysis and learning mechanism, database management, search restarts) interacting in many unpredictable ways. There is the common belief that the resulting emergent behavior of these complex systems takes advantage of a certain underlying structure of the SAT formula, which is shared by the majority of these industrial problems regardless the domain they come from. Recently, there have been some attempts of characterizing this structure under the lens of complex networks, with the purpose of better understanding the success of the solvers, and potentially improving them.

In this paper, we analyze the structure of industrial SAT instances under the lens of self-similarity, and study how the execution of SAT solvers affect that structure. Many real-world graphs exhibit *self-similar structure* (with small *fractal dimension*), which means that after *rescaling* (replacing groups of nodes by a single node), the same kind of structure can be observed. In our analysis, in which we represent SAT instances as graphs, we observe that many industrial SAT formulas exhibit the same kind of structure. Moreover, we analyze how this structure evolves by effects of learning new clauses during the search. In particular, we observe that learned clauses usually contain variables that are *close* in the graph representation of the formula. This is, the learning mechanism tends to work *locally*. On the contrary, this learning mechanism on random SAT formulas –which do not exhibit any structure at all– is unable to generate these local clauses. This difference contributes to explain the success of modern SAT solvers on industrial problems.

Keywords: satisfiability; SAT; fractal dimension; complex networks.

1. Introduction

The Boolean Satisfiability Problem (SAT) is the problem of deciding whether the Boolean variables of a propositional formula can be assigned in such a way that the formula is evaluated as true. A SAT formula is said to be satisfiable if such an assignment does exist, and unsatisfiable otherwise. This problem is central in Computer Science, from both theory and applications perspectives. It is the first NP-complete problem, which means that existing algorithms can run during exponentially long executions, in the worst case. But some algorithms –the so known *modern SAT solvers*– show a very good performance solving a certain kind of instances:

instances encoding problems from real-world applications or industrial domains, such as hardware and software verification, scheduling, cryptography, or planning, among others [1–3].

It is well established that those industrial SAT instances have a very distinct nature to randomly generated SAT formulas. As a consequence, SAT solving algorithms more *specialized* in random benchmarks do not usually perform well on industrial ones, whilst solvers for industrial purposes usually show a very bad performance on random SAT formulas. For instance, these *industrial-specialized* solvers are able to solve very large industrial SAT instances with millions of variables in a few seconds, whereas they are unable to solve random SAT formulas with hundreds of variables in a few hours or even days.

Modern SAT solvers implement the Conflict-Driven Clause Learning (CDCL) algorithm [4]. Built on the basis of the classical DPLL algorithm [5,6], which is a depth-first search algorithm exploring the variables assignment space of the formula, CDCL incorporates a number of complex techniques such as conflict analysis and clause learning [7], sophisticated variable branching heuristics [8], clause removal policies to delete useless learned clauses [9], random restarts of the search [10], or pre-/in-processing techniques to detect variables dependencies and simplify the formula [11].

Since CDCL SAT solvers can be seen as complex systems where there are many unpredictable interactions between its components, the reasons that explain why these solvers show such a different performance on random and industrial instances, or more specifically, the reasons that explain their success on a large variety of industrial benchmarks, are not completely understood yet. It is therefore fundamental to shed light to this open question, which is the main motivation of this manuscript. A better understanding of these solvers and the reasons of their success will possibly help to improve them.

The main component of CDCL SAT solvers is clause learning [12,13]. These learned clauses are the result of the conflicts found during the search, and they prevent the solver to find the same conflicts in the future. Moreover, a very extended heuristic implemented in many CDCL SAT solvers is VSIDS [8]. This heuristic selects the next branching variable based on the already found conflicts. This results into a synergy between expanding the search (branching variables) and delimiting it (found conflicts); this synergy is somehow *captured* by learned clauses.[1] Therefore, learned clauses play a key role in the algorithm.[2]

A common intuition to explain the distinct performance of CDCL SAT solvers between random and industrial benchmarks comes from the belief that these solvers are able to exploit a certain underlying structure of the formula [9,16–20], that is shared by the majority of the benchmarks. In our work, we are inspired by the work of *complex networks* where the general structure of real-world graphs is studied. An extended observation is that the classical Erdős-Rényi random graph model [21] is not suitable to study the structure of these complex networks. In order to characterize the underlying structure of industrial SAT instances, we follow a similar approach. In particular, we represent SAT instances as graphs, and then analyze some graph features in these graph representations. For this purpose, we use two graph models. In one model, SAT formulas are represented as bipartite graphs where their nodes represent either the variables or the clauses of the formulas, and an edge between a variable-node and a clause-node indicates the existence of such a variable in such a clause. In the second model, nodes only represent variables, and there is an edge between two variable-nodes if there is a clause containing both variables.

Following this approach of representing SAT instances as graphs, an interesting observation is described in [22]. In this work, it is studied the community structure of industrial SAT formulas, and authors observe that most of the industrial formulas show a very clear community structure. In a graph with clear community structure, it can be found a partition of disjoint sets of nodes, called *communities*, such that the graph has more edges connecting nodes of the same community. In the context of SAT instances, this means that the variables of the formula mostly appear in clauses with the same subset of variables. Interestingly enough, they observe that

---

1 Every variable has an *activity* score, equally initialized. When a variable occurs in a conflict, it increases its activity, and it is very likely that such a variable occurs in the resulting learned clause. The VSIDS heuristic selects the variable with highest activity. Since this variable probably occurs in existing learned clauses, it is very likely it triggers new conflicts. This is the synergy between the conflicts and the heuristic, *captured* by learned clauses.

2 It is intriguing if they also play the same role in other common heuristics, as LRB [14] or VMTF [15].

the quality of this community structure tends to decrease by the addition of learned clauses.[3] It remains unclear, however, whether this decrease is achieved *quickly* or *slowly*.

In this work, we focus on the analysis of the *self-similarity* of SAT formulas and their *fractal dimension*. In a graph with self-similar structure, the structure of the graph is preserved at different *scales* (for instance, by grouping sets of nodes into a single nodes). It also means that the diameter d^{max} of the graph grows as $d^{max} \sim n^{1/d}$, being d its fractal dimension. Notice that in random graphs or small-world graphs $d^{max} \sim \log n$.

Our experimental evaluation shows that many application SAT instances exhibit self-similar structure with a small fractal dimension ranging between 2 and 4. We conjecture that this structure, as well as other common properties in these instances, such as community structure [22] or scale-free structure [23], are already existent in many of the high-level problems encoded into the SAT instances. Thus, for example, hardware-verification problems may exhibit this structure as a consequence of the circuits they encode. We also evaluate how the fractal dimension is affected by the effect of learning new clauses during the search. In particular, we observe that most learned clauses contain variables that are already related by other clauses. In the graph representation of the formula, this means that learned clauses create edges between already connected nodes (i.e., variables) or very *close* ones. In other words, the learning mechanism tends to work *locally*. This observation allows us to determine that the underlying structure of the formula is destroyed *slowly*.

This paper is an extended and revised version of [24]. Some preliminary results were also included in Jesús Giráldez-Cru's PhD thesis [25].

The rest of the paper proceeds as follows. Some preliminary concepts are introduced in Section 2. We study the self-similar structure of graphs and SAT formulas in Sections 3 and 4 respectively. We analyze how this structure is affected by the CDCL SAT solving techniques in Section 5. Finally, related works and conclusions are respectively described in Sections 6 and 7.

2. Preliminaries

Given a set of Boolean variables $X = \{x_1, \ldots, x_n\}$, a *literal* is an expression of the form x_i or $\neg x_i$. A *clause* c of size s is a disjunction of s literals, $l_1 \vee \ldots \vee l_s$. We note $s = |c|$, and say that $x \in c$, if c contains the literal x or $\neg x$. A *CNF formula* or *SAT instance* of length t is a conjunction of t clauses, $c_1 \wedge \ldots \wedge c_t$. A *k-CNF formula* is a conjunction of k-sized clauses.

An undirected graph is a pair (V, E) where V is a set of vertices and $E \subseteq V \times V$ is a set of edges between the vertices. The degree of a vertex x is defined as $\deg(x) = |\{y \in V | (x, y) \in E\}|$. A bipartite graph is a tuple (V_1, V_2, E) where $E \subseteq V_1 \times V_2$.

We represent SAT instances using two graph models. In the Variable Incidence Graph model (VIG, for short), vertices represent variables, and edges represent the existence of a clause relating two variables. A clause $x_1 \vee \cdots \vee x_n$ results into $\binom{n}{2}$ edges, one for every pair of variables.

Definition 1 (Variable Incidence Graph (VIG))**.** *Given a SAT instance Γ over the set of variables X, its variable incidence graph is a graph (X, E) with set of vertices the set of Boolean variables, and the set of edges E:*

$$(x, y) \in E \Leftrightarrow c \in \Gamma \wedge x \in c \wedge y \in c \tag{1}$$

In the Clause-Variable Incidence Graph model (CVIG, for short), vertices represent either variables or clauses, and edges represent the occurrence of a variable in a clause.

[3] The addition of learned clauses tends to destroy the community structure of the formula.

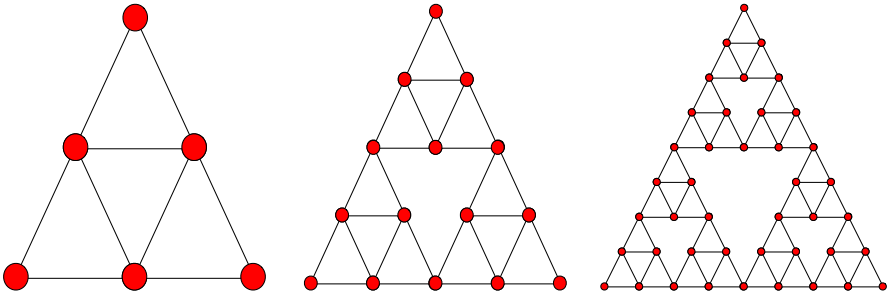

Figure 1. Example of a graph with self-similar structure at distinct scales.

Definition 2 (Clause-Variable Incidence Graph (CVIG))**.** *Given a SAT instance* Γ *over the set of variables* X*, its clause-variable incidence graph is a bipartite graph* $(X, \{c \mid c \in \Gamma\}, E)$*, with vertices the set of variables and the set of clauses, and a set of edges* E*:*

$$(x, y) \in E \Leftrightarrow y \in \Gamma \wedge x \in y \tag{2}$$

These models do not fully represent the SAT formula, ignoring some of its features. For instance, having the CVIG it cannot be known the sign of a variable in a clause, or the length of clauses cannot be known having the VIG. In the literature there are other graph models for SAT formulas capturing all their information. For instance, the factor graph (similar to the CVIG) with two kind of edges (one for positive literals and the other for negative ones). However, these simplified models have already all the information required for our analysis.

3. The Self-Similar Structure of Graphs

In this section, we describe the concepts and techniques used to analyze the self-similar structure of graphs. In particular, we introduce the Burning by Node Degree (BND) algorithm (see Alg. 1), which will be used to analyze the fractal dimension in very large graphs, as the graph representation of industrial SAT formulas are. We also analyze the relation between fractal dimension and diameter.

In Figure 1 we depict a graph with clear self-similar structure in order to graphically describe the concept of fractal dimension of a graph.

In order to define the fractal dimension of a graph, we follow the principle of self-similarity. We first need to define the concepts of distance between nodes and graph coverage. For this purpose, we will use the definition of box covering [26].

Definition 3. *The* **distance** *between two nodes of a graph is the minimum number of its edges we need to follow to go from one node to the other.*

Definition 4. *The* **diameter** d^{max} *of a graph is the maximal distance between any two nodes of the graph.*

Definition 5. *Given a graph* G*, a* **box** B *of size* l *is a subset of its nodes such that the distance between any pair of them is strictly smaller than* l*.*

Definition 6. *The* **minimum number of boxes** *of size* l*,* $M(l)$*, to* cover *the graph is the minimum number of boxes such that every node of the graph belongs to some box.*

Notice that $M(1)$ is equal to the number of nodes of G, and $M(d^{max}+1)$ is the number of connected components of the graph (i.e., $M(d^{max}+1) = 1$ if the graph is fully connected). Notice also that the function $M(l)$ has the same value for any l strictly greater than d^{max} (i.e., the number of connected components of the graph).

Definition 7. *A graph has the* **self-similarity** *property if the function* $M(l)$ *decreases polynomially, i.e.* $M(l) \sim l^{-d}$*, for some value* d*. In this case, we call* d *the* **dimension** *of the graph.*

In graphs with self-similar structure, this function $M(l)$ decreases *slowly*, in contrast to other graphs for which this functions decreases faster. Therefore, in this latter case of graphs, the diameter is comparatively smaller than in those graphs showing fractal dimension.

Lemma 8. *Computing the function $M(l)$ is NP-hard.* [4]

Proof. We prove that computing $M(2)$ is already NP-hard by reducing the graph coloring problem to the computation of $M(2)$. Given a graph G, let $\overline{G}$, the complement of G, be a graph with the same nodes, and where any pair of distinct nodes are connected in $\overline{G}$ iff they are not connected in G. Boxes of size 2 in $\overline{G}$ are cliques, thus they are sets of nodes of G without an edge between them. Therefore, the minimal number of colors needed to color G is equal to the minimal number of cliques needed to cover $\overline{G}$, i.e. $M(2)$. □

There are several efficient algorithms that approximate $M(l)$. They compute upper bounds of $M(l)$. They are called *burning* algorithms (see [27]). Following a greedy strategy, at every step they try to select the box that covers (burns) the maximal number of uncovered (unburned) nodes. Although they are polynomial algorithms, we still need to do some further approximations to make the algorithms of practical use in very large graphs.

First, instead of boxes, we will use *circles*.

Definition 9. *Given a graph G, a* **circle** *of* radius *r and* center *c is a subset of its nodes such that the distance between any of them and the node c is strictly smaller than r.*

Definition 10. *The* **minimum number of circles** *of radius r, $N(r)$, to* cover *the graph is the minimum number of circles such that every node of the graph belongs to some circle.*

Notice that any circle of radius r is inside of a box of size $2r-1$ (the opposite is in general false) and any box of size l is inside a circle of radius l (it does not matter what node of the box we use as center). Notice also that every radius r and center c characterizes a unique circle.

According to Hausdorff's dimension definition, $N(r) \sim r^{-d}$ also characterizes self-similar graphs of dimension d. We can approximate this fractal dimension using the *Maximum-Excluded-Mass-Burning (MEMB)* algorithm [27], which works as follows: Consider a graph G and a radius r. We compute an upper bound of the number of circles with radius r necessary to cover the graph $N(r)$. We start with all nodes set to unburned. At every step, for every possible node c, we compute the number of unburned nodes covered by the circle of center c and radius r, then select the node c that maximizes this number, and burn the new covered nodes. Starting with $r=1$, we repeat this process for every value of r until we get $N(r)=1$.

The MEMB algorithm is still too costly for our purposes. Notice that in every step a node is selected as a center, the algorithm has previously visited every node of the graph in order to find the best center. So in order to make the algorithm more efficient, we apply the following algorithm, called *Burning by Node Degree (BND)*, and described in Alg. 1. First, we compute $N(1)$ as the number of nodes in the graph (line 1). Then, we iterate to compute $N(r)$ with $r>1$, until we reach the number of connected components of the graph (line 3). In each iteration, we use node degrees to order the set of nodes: $\langle c_1,\ldots,c_n\rangle$ such that $degree(c_i) \geq degree(c_j)$, when $i<j$ (line 7). Now, for $i=1$ to n, if c_i has not been burned in previous iterations, the circle of center c_i and radius r is selected, burning all the nodes contained in this circle (line 8-10). This is repeated until all nodes are burned, obtaining $N(r)$ for this value of r (line 11). Notice that this criterion may not maximize the number of burned nodes.

We recall that BND (see Alg. 1) computes upper bounds of the function $N(r)$ until it gets the number of connected components of the graph. Notice that $N(d^{max}+1)=1$, placing the center of this circle in any node

[4] In [27] it is stated the same result, but they prove the wrong reduction. They reduce the computation of $M(2)$ to the graph coloring problem.

Algorithm 1: Burning by Node Degree (**BND**)

Input: Graph $G = (V, E)$
Output: vector[int] N

```
1  N[1] := |V|;
2  int i := 2;
3  while N[i − 1] > connectedComponents(G) do
4      vector[bool] burned(|V|, false); // all nodes initially unburned;
5      N[i] := 0;
6      while existsUnburnedNode(burned) do
7          c := highestDegreeUnburnedNode(G, burned);
8          S := circle(c, i); // circle with center c and radius i;
9          foreach x ∈ S do
10             burned[x] := true;
11         N[i] := N[i] + 1;
12     i := i + 1;
```

of the path between the two most distant nodes. However, it is possible that BND computes the approximation of $N(r)$ for values of r greater than $d^{max} + 1$.

In the next section, the accuracy and performance of algorithms MEMB and BND will be compared. This analysis will allow us to justify the use of the latter in our experimental evaluation.

In this paper we analyze the function $N(r)$ for the graphs obtained from a SAT instance following the VIG and CVIG models. These two functions are denoted $N^V(r)$ and $N^C(r)$, respectively, and they relate to each other as follows.

Lemma 11. *If* $N^V(r) \sim r^{-d}$ *then* $N^C(r) \sim r^{-d}$.
If $N^V(r) \sim e^{-\beta r}$ *then* $N^C(r) \sim e^{-\frac{\beta}{2} r}$.

Proof. Notice that, for any formula, given a circle of radius r in the VIG model, using the same center and radius $2r - 1$ we can cover the same variable nodes in the CVIG model. With radius $2r$ we can also cover all clauses adjacent to some covered variable. Hence $N^C(2r) \leq N^V(r)$.

Conversely, given a circle of radius $2r$ in the CVIG model, we consider two possibilities. If the center is a variable node, we cover the same variables in the VIG model using a circle of radius r and the same center. If the center is a clause c, to cover the same variables in the VIG model, we need a circle of radius $r + 1$ centered in a variable node adjacent to c. Hence $N(r + 1) \leq N^C(2r)$.

Therefore $N^V(r + 1) \leq N^C(2r) \leq N^V(r)$, and $N^V(r) \sim N^C(2r)$. From this asymptotic relation, we can derive the two implications stated in the lemma. □

From the previous lemma, it can be concluded that if a SAT instances has a (perfect) self-similar structure, its fractal dimension is the same in both models VIG and CVIG. In this case, we would obtain a line if we represent $N(r)$ as a function of r in double-logarithmic scale, being $-d$ its slope. If there is an exponential decay in the function $N(r)$, then the decay factor in the VIG is double of the decay factor in the CVIG model. In that case, representing $N(r)$ in semi-logarithmic axes, we would obtain a line with slope $-\beta$. In our analysis, we always represent $N(r)$ in double-logarithmic axes. Therefore, if $N(r)$ decays exponentially, we will observe a concave curve.

3.1. Fractal Dimension versus Diameter

The function $N(r)$ determines the *maximal radius* r^{max} of a connected graph, defined as the minimum radius of a circle covering the whole graph minus one: $N(r^{max} + 1) = 1$. The maximal radius and the *diameter* d^{max} of a graph are also related, because $r^{max} \leq d^{max} \leq 2r^{max}$. From these relations we can conclude the following.

 doi:10.20944/preprints202010.0560.v1

Lemma 12. *For self-similar graphs (where $N(r) \sim r^{-d}$), the diameter is $d^{max} \approx n^{1/d}$, where d is the fractal dimension.*
In graphs where $N(r) \sim e^{-\beta r}$, the diameter is $d^{max} \approx \frac{\log n}{\beta}$.

Proof. The diameter of a graph and the maximal radius are related as $r^{max} \leq d^{max} \leq 2\, r^{max}$. Notice that, by definition of the function $N(r)$, we have $N(1) = n$, where n is the number of nodes, and $N(r^{max} + 1) = 1$.

Assuming $N(r) = C\, r^{-d}$ and replacing r by 1 we get $C = n$. Then, replacing r by $r^{max} + 1$, we get $1 = N(r^{max} + 1) = n\,(r^{max} + 1)^{-d}$. Hence, $r^{max} = n^{1/d} - 1$.

Assuming $N(r) = C\, e^{-\beta r}$ and replacing r by 1 we get $C = n\, e^{\beta}$. Then, replacing r by $r^{max} + 1$, we get $1 = N(r^{max} + 1) = n\, e^{-\beta\,(r^{max})}$. Hence, $r^{max} = \frac{\log n}{\beta}$. □

The diameter, as well as the *typical distance*[5] L of a graph, have been extensively used for characterizing graphs. For instance, *small world graphs* [28] are characterized by small typical distance $L \sim \log n$ and a large clustering coefficient.

Notice that this characterization works well for *families* of graphs since these features can be expressed as a function on the number of nodes. On the other hand, it may be inaccurate in the case of a particular graph, since it is hard to asses whether its typical distance is "small" or if its clustering coefficient is "large". Moreover, both the diameter and the typical distance of a graph are hard-to-compute features. Notice that even though there is a quadratic algorithm, computing them in huge graphs is a time-consuming task. This is the case of the application SAT instances. In fact, our approximation to the fractal dimension can be computed more efficiently than the diameter.

Additionally, the fractal dimension is independent of the graph size, and thus is a better feature to characterize the instance structure. As a consequence, we can analyze formulas of the same family and very distinct size, all having a similar structure. In particular, the fractal dimension will be similar in all the instances of the family, and the shapes of the function $N(r)$ will be resembling.

4. The Self-Similar Structure of SAT Instances

In this paper, we have used the set of industrial formulas of the SAT Competition 2013[6]. They are 300 instances grouped into 19 families: *2d-strip-packing*, *bio*, *crypto-aes*, *crypto-des*, *crypto-gos*, *crypto-md5*, *crypto-sha*, *crytpo-vmpc*, *diagnosis*, *hardware-bmc*, *hardware-bmc-ibm*, *hardware-cec*, *hardware-velev*, *planning*, *scheduling*, *scheduling-pesp*, *software-bit-verif*, *software-bmc* and *termination*. In Table 1, we report some statistics about these benchmarks. All these instances are *industrial*, in the sense that they come from a real-world problem. During the paper, we compare them to random 3-CNF formulas.

We have conducted an exhaustive analysis of these 300 industrial SAT instances, and 90 random 3-CNF formulas of 10^5 variables generated at different clause/variable ratios. We will see that most industrial instances are self-similar and have a small fractal dimension, i.e. $N(r) \sim r^{-d}$, for small d. In random instances $N(r)$ decays exponentially, i.e. $N(r) \sim e^{-\beta r}$. All experiments were carried out in a cluster of 9 nodes IBM dx360 M2, each of them with 32GB of RAM and 2 processors Intel(R) Xeon(R) CPU L5520 2.27 GHz, limiting all experiments to a single core and to a maximum of 4GB of RAM.

Before presenting the results of this experimental analysis, we present a comparison of the BND and the MEMB [27] algorithms, evaluating their performance and their accuracy. This analysis allows us to justify the use of the BND algorithm to compute the fractal dimension of SAT instances.

We execute the BND and the MEMB algorithms for the set of 300 industrial instances of the SAT Competition 2013. In this experiment, we set a timeout limit of half an hour. The BND algorithm is able to complete the computation in all the 300 instances. On the contrary, the MEMB algorithm only finishes in 17 instances. Also, the BND algorithm spends an average run-time of 0.11 seconds, while the MEMB algorithms spends 607.2 seconds in average. Finally, the function $N^C(r)$ approximated by both algorithms is very alike.

---

[5] The typical distance of a graph is the average of the distances between any two nodes.
[6] http://satcompetition.org/2013/

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 October 2020 doi:10.20944/preprints202010.0560.v1

Family	#instances	avg. #vars.	avg. #clauses
2d-strip-packing	5	12753.20	1267404.40
bio	5	46376.40	471256.00
crypto-aes	11	25285.90	82980.81
crypto-des	9	31590.22	95015.66
crypto-gos	30	1892.93	22883.60
crypto-md5	11	66894.00	267579.54
crypto-sha	30	4773.33	151027.86
crypto-vmpc	8	1061.50	171947.00
diagnosis	26	235279.42	1129797.46
hardware-bmc-ibm	4	1335863.00	5234846.00
hardware-bmc	3	114448.33	341918.00
hardware-cec	30	255757.16	754358.03
hardware-velev	21	282973.85	7478281.09
planning	25	469285.60	3323685.24
scheduling-pesp	30	37454.30	270133.10
scheduling	30	160310.13	722325.73
software-bit-verif	14	131492.71	378409.78
software-bmc	3	11216817.00	32697150.00
termination	5	219846.80	950546.60
SATComp2013	300	234984.90	1469306.71

Table 1. Statistics about the application benchmarks of the SAT Competition 2013.

This can be seen in Fig. 2, where it is represented the obtained approximation in the 17 SAT formulas solved by both algorithms in the given timeout. In the rest of our experimental analysis, we will only use the BND algorithm, given that the shapes of both approximations are very similar and that the MEMB algorithm is much slower.

We also emphasize that the MEMB algorithm is slightly more accurate in some cases. This explains why a few upper bounds of $N^C(r)$ that BND calculates are above the ones calculated by MEMB. The exact values of $N^C(r)$ are probably even lower, especially for high values of r (where the approximation is less accurate).

The VIG model that represent a random 2-SAT instance is exactly an Erdős-Rényi graphs. These formulas have a satisfiability threshold at $m/n = 1$, commonly known as the phase transition point. This means that all formula below (resp. above) this ratio are satisfiable (resp. unsatisfiable). It is also known that at $m/n = 0.5$ there is a percolation threshold. This means that the VIG of formulas above this ratio have a major connected component, whereas below this threshold the VIG graph breaks into many connected components. In the percolation point the formula is self-similar with a fractal dimension $d = 2$. Above this ratio, the function $N^V(r)$ decays exponentially. To the best of our knowledge, there is no known result of this kind for random 3-CNF formulas.

In Fig. 3 we represent the functions $N^V(r)$ and $N^C(r)$, approximated by the BND algorithm, for families of random 3-CNF instances with different clause/variable ratios. Although it is not showed in this figure, we observed that these functions does not depend on the number of variables of the family; they only depend on their ratio of clause-to-variable m/n.

In the phase transition point $m/n = 4.25$, the function $N^V(r)$ exhibit an exponential decay with $\beta = 2.3$, i.e., $N^V(r) \sim e^{-2.3r}$. Hence, $r^{max} = \frac{\log n}{2.3} + 1$. For example, with $n = 10^5$ variables random SAT formulas have a radius $r^{max} \approx 6$. The decay factor is even higher for higher values of m/n. The same phenomenon is observed in the CVIG model. However, the exponential decay is $\beta = 1.16 \approx 2.3/2$ in the family of formulas at the phase transition point. Hence, the decay in the VIG model is approximately double of the decay in the CVIG, as expected by Lemma 11. A percolation threshold is experimentally found at $m/n \approx 0.17$, where the main connected component also exhibits a fractal dimension $d = 2$.

In Fig. 4 we represent the function $N^C(r)$ for some application or industrial families of SAT instances. It can be observed that most of these families exhibit self-similar structure, with most fractal dimensions ranging between 2 and 4. All the formulas of the same family exhibit the same fractal dimension in many of the families, where this dimension seems to be a characteristic feature of the family. In many of the industrial families, all

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 October 2020 doi:10.20944/preprints202010.0560.v1

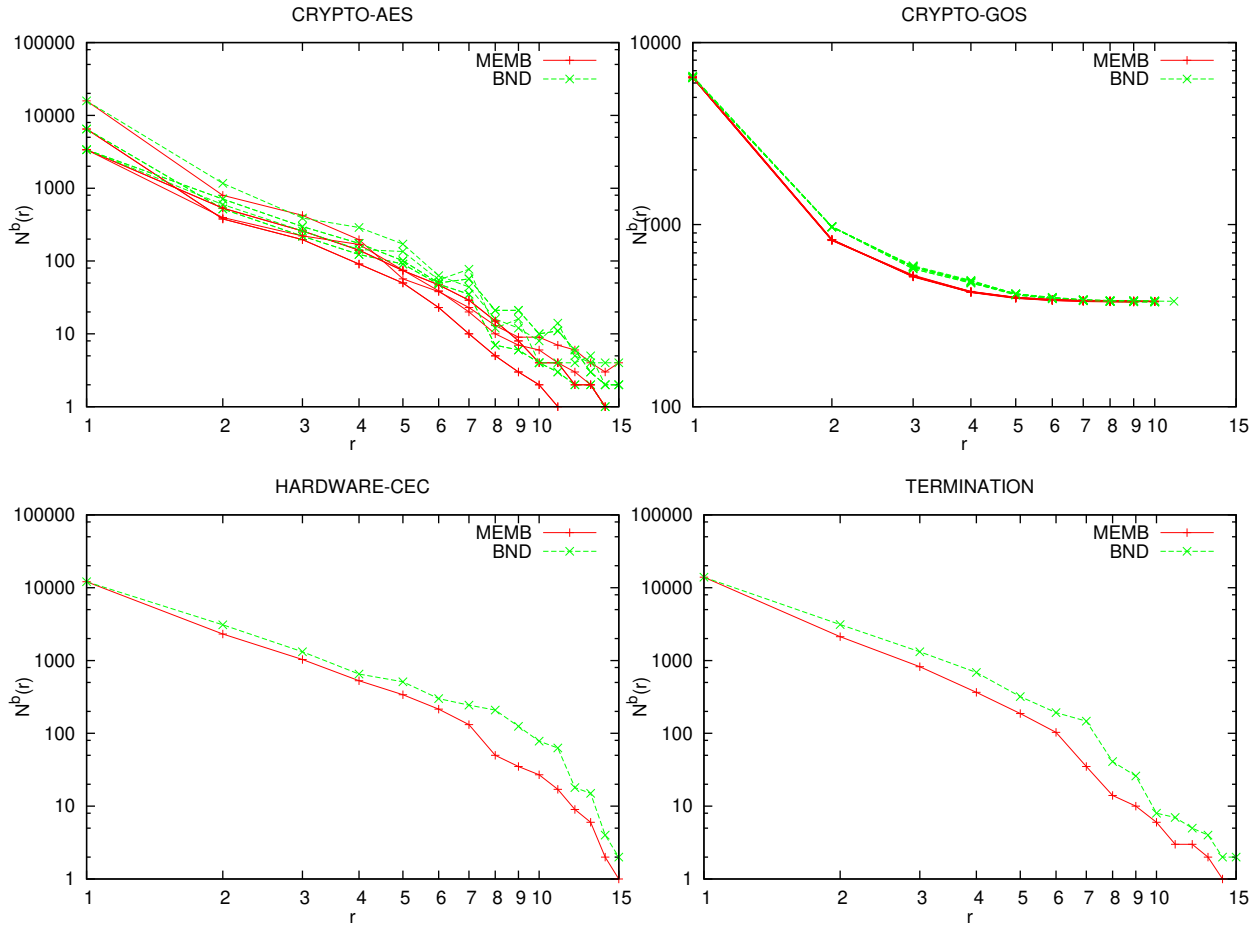

Figure 2. Upper bounds for $N^b(r)$ obtained with MEMB and BND algorithms, for the 17 industrial instances that MEMB is able to compute in 30 minutes, grouped by families.

instances have the same fractal dimension, being this dimension a particular property of the family. See, for instance, families *crypto-sha* or *diagnosis*. It is interesting to see that the formula size does not alter the value of the fractal dimension (in our representation in logarithmic scale, they all share the same function shape).

In general, we observe that for small values of r, the polynomial decay is clearer, with an almost identical slope for all the formulas of the family.

On the other hand, for big values of r, we must make some remarks.

First, the upper bound on $N^C(r)$ that we compute may be inaccurate. Second, we identify two events. First, in some cases although the whole function can not be approximated by an exponential function, there is an abrupt decay (see some *hardware-cec* or *termination* instances, for example). This can be due to a small set of edges that connect distant part of the graph. Notice they may drastically reduce the number of circles required to cover the graph when the value of r is high, but they would have no effect for smaller values. Second, there is a long tail in some other cases (see *hardware-bmc-ibm*, for instance). This is due to the existence of (small) unconnected components in the graph. However, this tail disappears if we compute $N(r)$ only for the major component.[7]

Finally, there are some families where the $N(r)$ function decays exponentially, i.e., they are not self-similar. This is the case, for instance, of the *hardware-velev* family. However, we emphasize that this kind of families are rare, since most of the industrial formulas and families of our analysis do have self-similar structure, i.e., with a $N(r)$ function decaying polynomially.

[7] We can subtract from $N(r)$ the number of unconnected components, as an approximation, since most are covered with a few circles.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 October 2020 doi:10.20944/preprints202010.0560.v1

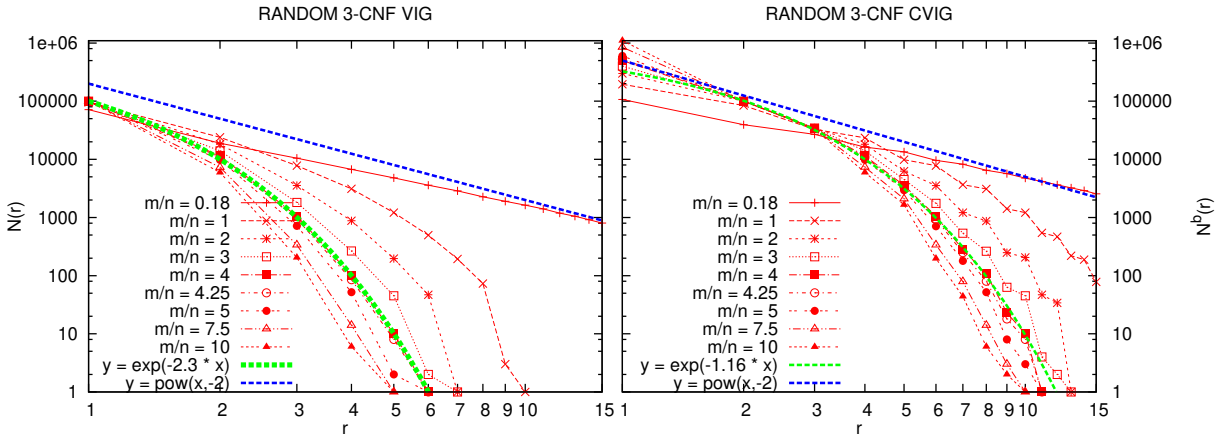

Figure 3. Functions $N^V(r)$ for VIG (left), and $N^C(r)$ for CVIG (right), for 3-CNF random formulas with distinct values of m/n. Formulas are generated using $n = 10^5$ variables and taking the major connected component, except for $m/n = 0.18$, where $n = 10^6$.

4.1. Fractal Dimension at Fine-Grained Scale

If a graph exhibits self-similar structure, the same structure can be observed at all scales. This means that we could group a subset of nodes covered (i.e., the ones covered by a circle) into a single node, without altering the same structure in the resulting graph. In our evaluation, we observe that this happens when r is small (the function behavior is $N(r) \approx C\, r^{-d}$ when r is small). However, this does not always happen for big values of r. This can be because the self-similar structure disappears at a coarse-grained scale, or because the function $N(r)$ is not well approximated. When a SAT instance has a small unsatisfiability proof, a small cycle can be observed in the graph models representing the formula (e.g., VIG and CVIG), suggesting that the fine-grained scale of the fractal dimension is more relevant. In other words, we conjecture that is more important the value of the fractal dimension at fine-grained scale, i.e. the slope of the function $N(r)$ for small values of r, rather than the existence of a self-similar structure. Therefore, in our evaluation, we note these fine-grained dimensions as d^V and d^C for the VIG and CVIG, respectively. We compute them as the interpolation, by linear regression, of $\log N(r)$ vs. $\log r$. We use the values of $N^V(r)$ and $N^C(r)$, for $r = 1, \ldots, 6$. Experimentally, we see that these approximations are accurate enough.

5. The Self-Similar Structure during SAT Solver Search

CDCL SAT solvers add during their execution new clauses learned from the conflicts the solver finds. A conflict represents a (partial) assignment of the variables which leads to a contradiction (i.e., an unsatisfied clause). These conflicts can be learned in order to prevent the solver to find them in the future. When a learned clause is unary (i.e., it only has one literal), this literal can be propagated (i.e., assigned), and this results in a simplification of the given formula. Given a unary clause x, clauses with literal x are completely removed (i.e., they are satisfied by x), and literals $\neg x$ are removed from the formula. On the other hand, learned clauses of bigger length may create new relations between variables, i.e., new edges in the graph representations of the formula.

Both, the simplification caused by learning unary clauses, and the addition of longer learned clauses, can alter the fractal dimension of the SAT instance. In general graphs, the addition of edges (preserving the nodes) can only increase its dimension, because existing circles may cover more nodes (due to the new edges), and thus the number $N(r)$ of circles required to cover the graph may decrease, whereas $N(1)$ (the number of nodes) remains unchanged. As a consequence, this can only increase the fractal dimension since the slope of $N(r)$ increases. In the VIG model, learning non-unary clauses only introduces new edges, thus as we argued the fractal dimension can only increase. On the contrary, this argument is more complicated in the CVIG model. Let $N(r)$ be the original number of circles and $N'(r)$ the minimum number of circles needed after adding L learned clauses. We have $N'(1) = N(1) + L$, since we add L new nodes. For $r > 1$, the whole graph can be covered with the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 October 2020 doi:10.20944/preprints202010.0560.v1

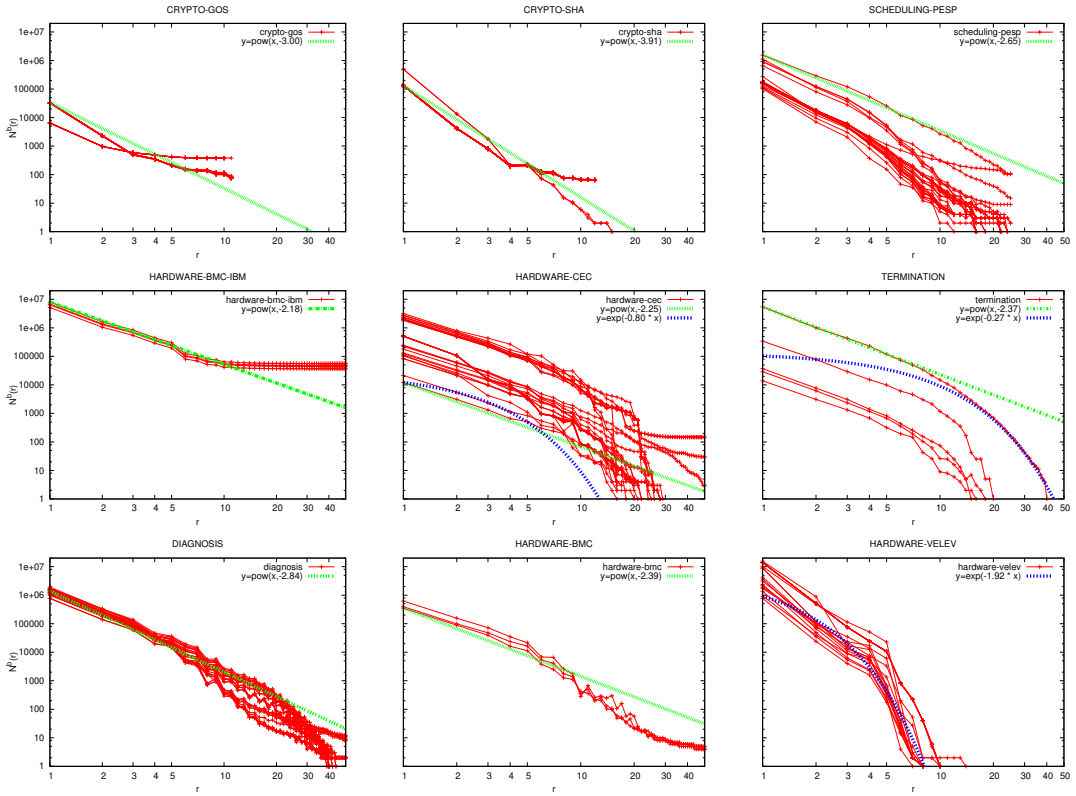

Figure 4. Function $N^C(r)$ for some industrial SAT formulas grouped by families.

former circles and using L new circles, used to cover the new nodes, thus $N'(r) \leq N(r) + L$ in the worst case. Therefore, the dimension could only increase. However, CDCL SAT solvers do not add unary clauses to the formula, but propagate them. And this decreases the value of $N(1)$ in both models, while increasing the value of $N(r)$ (with $r > 1$) because of the elimination and/or simplification of the clauses after the propagation(s). As a result, the fractal dimension can only decrease. In the previous argument, we assume that $\log N(r)$ is perfectly lineal on $\log r$, and $N(r)$ is computed exactly. In practice, we compute an approximation of $N(r)$. Moreover, we compute the dimension by linear regression, since points are not aligned. Hence, the variation in the dimension due to learning is rather unpredictable.

In order to study the evolution of the fractal dimension during SAT solver search, we have conducted the following experiments. First, we analyze this phenomenon in random 3-CNF instances with 10^5 variables and distinct clause/variable ratios. It is very unlikely that solving these formulas produces any unary learned clauses, hence neither nodes nor edges are removed. In Fig. 5, we compare the original dimension d^C_{orig} of the formulas and its the fractal dimension $d^C_{learned}$ after adding learned clauses. It can be observed that learning new clauses increases the dimension of the formula, as expected theoretically, being is bigger for formulas with higher clause/variable ratio. This increase can be *quantified* repeating the same experiment and replacing the clauses learned by the solver by randomly generated clauses of the same length, and computing the new fractal dimension d^C_{random}; the results are also shown in Fig. 5. In this second experiment it can be observed that the increase in the dimension is bigger when learning random clauses: $d^C_{random} \geq d^C_{learned} \geq d^C_{orig}$. This means that actual learned clauses, even in random k-CNF formulas, tend to connect variables that were already *close* in the graph representation of the formula.

In application SAT formulas, in contrast, the solver usually learns some unary clauses. In Fig. 6, we represent separately the effects of simplifying the formula after *learning* unary clauses (left), and the effect of learning longer clauses (right), after 10^3 conflicts.

In most of the application benchmarks, simplifying the formulas with unary clauses slightly in general decreases its fractal dimension, as expected. This dimension is represented as d^C_{simp}. In fact, in some instances

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 October 2020 doi:10.20944/preprints202010.0560.v1

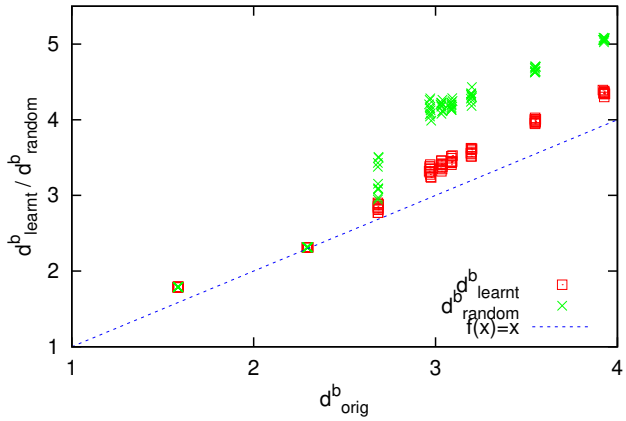

Figure 5. Relation between the original fractal dimension d^C_{orig}, and the dimension $d^C_{learned}$ after adding learned clauses, or after adding random clauses d^C_{rand}, in random 3-CNF formulas. learned clauses are computed after 10^3 conflicts.

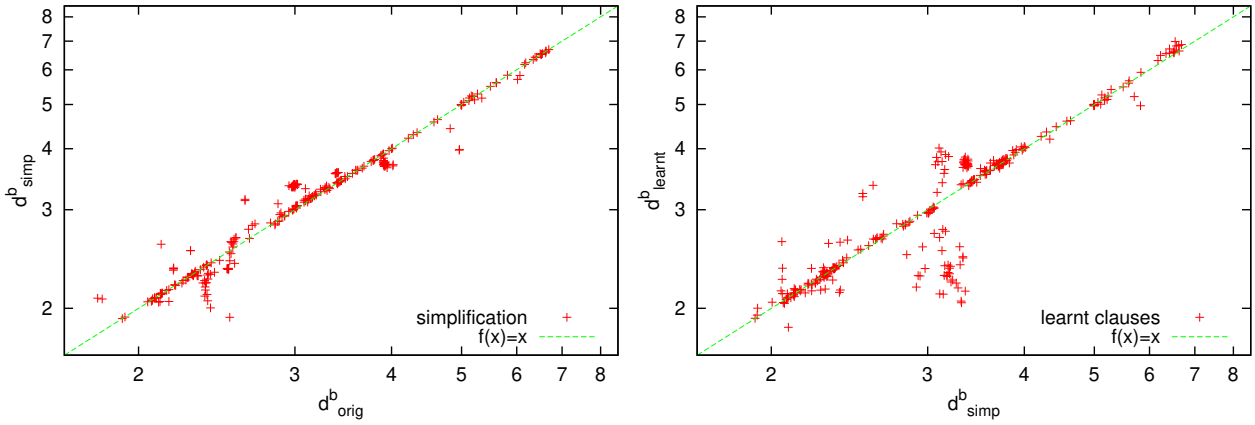

Figure 6. Relation between the original fractal dimension d^C_{orig} and the fractal dimension d^C_{simp} after simplifying the formula with the unary learned clauses (left), and relation between the fractal dimension d^C_{simp} and the fractal dimension $d^C_{learned}$ after simplification and adding learned clauses (right), for all industrial formulas. learned clauses are the result of 10^3 conflicts.

there is a huge difference between d^C_{simp} and d^C_{orig}. This is the case of families *crypto-md5* and *crypto-sha*. In these benchmarks, propagating unary clauses is more important, and produces more changes in the fractal dimension. Nevertheless, in a few instances the fractal dimension d^C_{simp} slightly increases. Analyzing the effects of learning longer clauses, it can be observed in most of the instances that the dimension $d^C_{learned}$ after adding such learned clauses slightly increases the fractal dimension w.r.t. d^C_{simp}, as expected.

This increase is particularly remarkable in the families *crypto-aes*, *crypto-gos* and *crypto-vmpc*. This suggests that learned clauses connect distant part of the formula, hence $N^C(r)$ drops off. Again, we also observe some cases where the dimension $d^C_{learned}$ decreases due to clause learning.

In Fig. 7, we represent the variations in the fractal dimension after 10^3, 10^4 and 10^5 conflicts. It can be observed that fractal dimensions can increase or decrease slightly after 10^3 conflicts, depending on the impact of simplifications caused by unary clauses (producing a lower fractal dimension) or adding learned clauses (producing a higher the fractal dimension). Nevertheless, after a longer number of conflicts (e.g., 10^5 conflicts), there is a more clear tendency: the fractal dimension increases in most of the cases. This results matches the expected behavior of the solver, where most of the unary clauses are learned in the early stages of the search, and explains that the dimensions tends to increase.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 27 October 2020 doi:10.20944/preprints202010.0560.v1

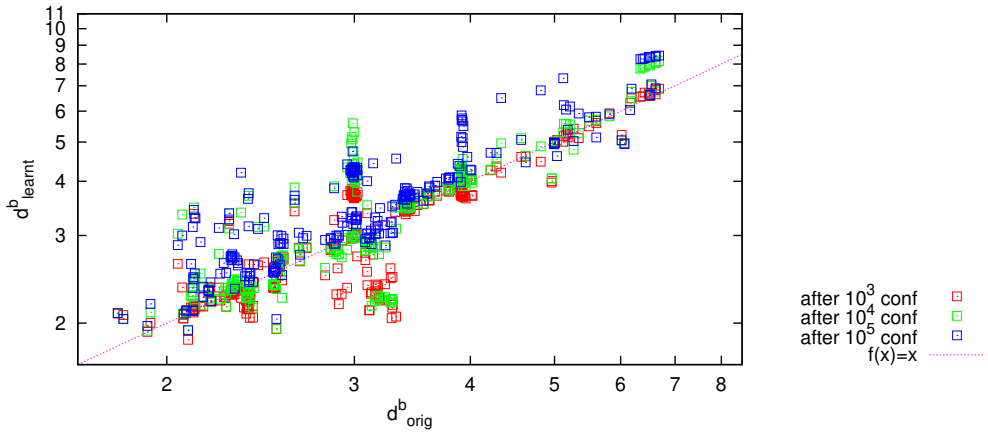

Figure 7. Relation between the original fractal dimension d^C_{orig} and the fractal dimensions after learning clauses $d^C_{learned}$, in industrial formulas.

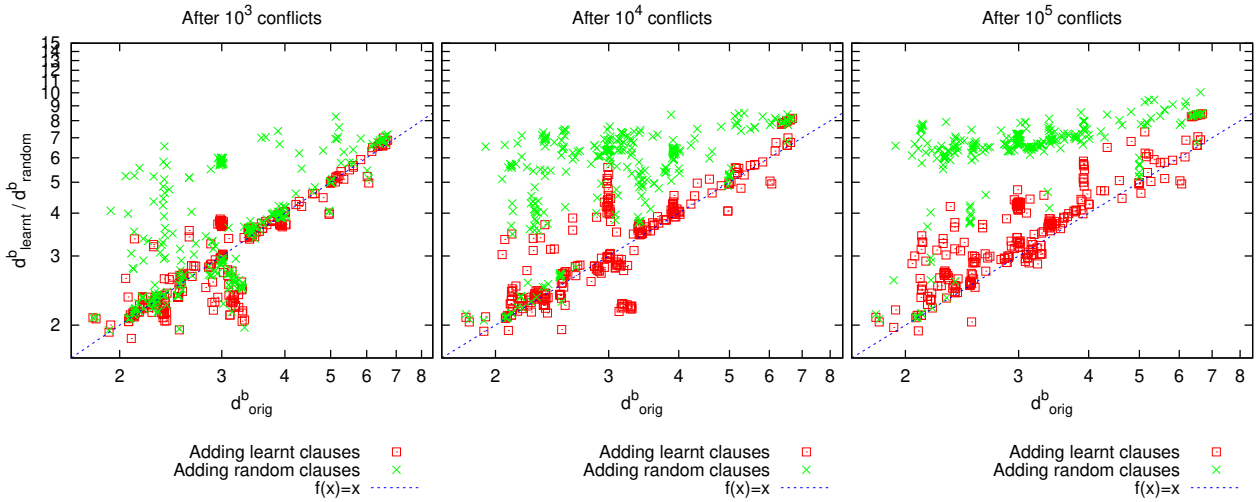

Figure 8. Relation between the original fractal dimension d^C_{orig} and the fractal dimensions after adding learned clauses $d^C_{learned}$, or after adding random clauses d^C_{random}, in industrial formulas.

Finally, we analyze whether learned clauses (tend to) connect distant part of the formula in these applications benchmarks. For this purpose, we compare the fractal dimension $d^C_{learned}$ of the formulas after adding learned clauses and the dimension d^C_{random} where the previous learned clauses are replaced by randomly generated clauses of the same length. In Fig. 8, we represent these two dimensions. It can be observed that d^C_{random} is considerably higher than $d^C_{learned}$.

This means that the clauses learned by the solver do not contribute to decrease $N(r)$, i.e., the number of circles required to cover the graph is similar. On the contrary, these random clauses would significantly decrease $N(r)$. Therefore, learned clauses mainly connect variables (nodes) that were already covered by the same circles, i.e., nodes already connected or close in the graph, and thus, clause learning acts quite locally in the formula.

6. Related Work

In the last ten years, there have been many works on the analysis of real-world SAT instances, seen as graphs [22–24,29–33]. The main aim of these studies is to better understand the success of CDCL techniques and help to improve them. They can also help develop new benchmarks generation models with more realistic properties [34–40]. This problem has been identified as an important challenge in the SAT community [41–43].

The definition of fractal dimension for graphs is based on the notion of *box covering* [26]. In [27] it is described a box-covering algorithm particularly suitable for complex networks, called

Maximum-Excluded-Mass-Burning (MEMB). They have used it to demonstrate the existence of self-similarity in many real-world networks [44,45]. There are other definitions of dimension for complex networks, like the one based on the *mass* (average number of vertices in a box) of the clusters [46]. The algorithm we propose in this paper is a variation of the MEMB algorithm, with the addition of a heuristic that allows us to process huge graphs, like the ones obtained from industrial SAT benchmarks. There are several models for generating scale-free graphs with fractal dimension [47–49].

7. Conclusions

The community on *complex networks* have developed methods and models to better understand the structure of real-world graphs. The classical Erdős-Rényi model, the first model in graph theory, cannot be used for the study of *real-world* networks, since this model does not generate the *structural* features these networks exhibit. Similarly, classical random SAT formulas cannot be used to better understand the underlying structure of industrial SAT instances and its relation to SAT solver performance. These instances are characterized by a particular structure, which may explain their distinct nature with respect to random formulas.

In this work, we have analyzed the *self-similarity* (or *fractal dimension*) of industrial SAT formulas, and how this structure evolves during the SAT solver search. We have found that most of industrial formulas show a self-similar structure, while random formulas do not. Their fractal dimension ranges, in general, between 2 and 4. Fractal dimension, typical distances and graph diameter are related (small dimension implies big distance and big diameter). Hence, industrial SAT instances have a big diameter. Intuitively, that means that we need long chains of implications to propagate a variable instantiation to others. Moreover, we have observed that fractal dimension increases due to learned clauses. However, this increase is bigger if we substitute learned clauses by random clauses of the same size. Therefore, learning *does not* contribute to connect distant parts of the formula very much. In other words, it works *locally*.

We think that the present study provides a step towards a practical explanation of why some SAT solvers perform better on industrial instances, and they do not on random formulas.

Funding: This work is partially supported by the EU H2020 Research and Innovation Programme under the LOGISTAR project (Grant Agreement No. 769142), MINECO-FEDER projects RASO (TIN2015-71799-C2-1-P) and TASSAT3 (TIN2016-76573-C2-2-P), the Spanish Ministerio de Economía y Competitividad under the EXASOCO project (ref. PGC2018-101216-B-I00), including European Regional Development Funds (ERDF). The third author is also supported by a MICINN Juan de la Cierva fellowship (ref. FJCI-2017-32420).

Abbreviations

The following abbreviations are used in this manuscript:

SAT	the Boolean satisfiability problem
CDCL	Conflict-Driven Clause Learning
DPLL	Davis-Putnam-Logemann-Loveland algorithm
VSIDS	Variable State Independent Decaying Sum heuristics
VIG	Variable Incidence Graph
CVIG	Clause-Variable Incidence Graph
MEMB	Maximum-Excluded-Mass-Burning algorithm
BND	Burning by Node Degree algorithm

1. Biere, A. Bounded Model Checking. In *Handbook of Satisfiability*; IOS Press, 2009; pp. 457–481.
2. Rintanen, J. Planning and SAT. In *Handbook of Satisfiability*; IOS Press, 2009; pp. 483–504.
3. Kroening, D. Software Verification. In *Handbook of Satisfiability*; IOS Press, 2009; pp. 505–532.
4. Silva, J.P.M.; Lynce, I.; Malik, S. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability*; IOS Press, 2009; pp. 131–153.
5. Davis, M.; Putnam, H. A Computing Procedure for Quantification Theory. *J. ACM* **1960**, *7*, 201–215. doi:10.1145/321033.321034.

6. Davis, M.; Logemann, G.; Loveland, D.W. A machine program for theorem-proving. *Commun. ACM* **1962**, *5*, 394–397. doi:10.1145/368273.368557.
7. Silva, J.P.M.; Sakallah, K.A. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Computers* **1999**, *48*, 506–521. doi:10.1109/12.769433.
8. Moskewicz, M.W.; Madigan, C.F.; Zhao, Y.; Zhang, L.; Malik, S. Chaff: Engineering an Efficient SAT Solver. Proc. of the 38th Design Automation Conf. (DAC'01), 2001, pp. 530–535.
9. Audemard, G.; Simon, L. Predicting Learnt Clauses Quality in Modern SAT Solvers. Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09), 2009, pp. 399–404.
10. Gomes, C.P.; Selman, B.; Kautz, H.A. Boosting Combinatorial Search Through Randomization. Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98), 1998, pp. 431–437.
11. Eén, N.; Biere, A. Effective Preprocessing in SAT Through Variable and Clause Elimination. Proc. of the 8th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'05), 2005, pp. 61–75.
12. Sakallah, K.A.; Marques-Silva, J. Anatomy and Empirical Evaluation of Modern SAT Solvers. *Bulletin of the EATCS* **2011**, *103*, 96–121.
13. Elffers, J.; Giráldez-Cru, J.; Gocht, S.; Nordström, J.; Simon, L. Seeking Practical CDCL Insights from Theoretical SAT Benchmarks. Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI'18), 2018, pp. 1300–1308. doi:10.24963/ijcai.2018/181.
14. Liang, J.H.; Ganesh, V.; Poupart, P.; Czarnecki, K. Learning Rate Based Branching Heuristic for SAT Solvers. Proc. of the 19th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'16), 2016, pp. 123–140.
15. Biere, A. CaDiCaL at the SAT Race 2019. Proceedings of SAT Race 2019 : Solver and Benchmark Descriptions, 2019, pp. 8–9.
16. Williams, R.; Gomes, C.P.; Selman, B. Backdoors To Typical Case Complexity. Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI'01), 2003, pp. 1173–1178.
17. Gomes, C.P.; Selman, B. Problem Structure in the Presence of Perturbations. Proc. of the 14th Nat. Conf. on Artificial Intelligence (AAAI'97), 1997, pp. 221–226.
18. Hogg, T. Refining the Phase Transition in Combinatorial Search. *Artif. Intell.* **1996**, *81*, 127–154.
19. Gent, I.P.; Hoos, H.H.; Prosser, P.; Walsh, T. Morphing: Combining Structure and Randomness. Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99), 1999, pp. 654–660.
20. Järvisalo, M.; Niemelä, I. The effect of structural branching on the efficiency of clause learning SAT solving: An experimental study. *J. Algorithms* **2008**, *63*, 90–113.
21. Erdós, P.; Rényi, A. On Random Graphs. *Publicationes Mathematicae* **1959**, *6*, 290–297.
22. Ansótegui, C.; Giráldez-Cru, J.; Levy, J. The Community Structure of SAT Formulas. Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'12), 2012, pp. 410–423.
23. Ansótegui, C.; Bonet, M.L.; Levy, J. On the Structure of Industrial SAT Instances. Proc. of the 15th Int. Conf. on Principles and Practice of Constraint Programming (CP'09), 2009, pp. 127–141.
24. Ansótegui, C.; Bonet, M.L.; Giráldez-Cru, J.; Levy, J. The Fractal Dimension of SAT Formulas. Proc. of the 7th Int. Joint Conf. on Automated Reasoning (IJCAR'14), 2014, pp. 107–121.
25. Giráldez-Cru, J. Beyond the Structure of SAT Formulas. PhD thesis, Universitat Autónoma de Barcelona, 2016.
26. Mandelbrot, B.B. *The fractal geometry of nature*; Macmillan, 1983.
27. Song, C.; Gallos, L.K.; Havlin, S.; Makse, H.A. How to calculate the fractal dimension of a complex network: the box covering algorithm. *Journal of Statistical Mechanics: Theory and Experiment* **2007**, *2007*, P03006.
28. Walsh, T. Search in a Small World. Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99), 1999, pp. 1172–1177.
29. Katsirelos, G.; Simon, L. Eigenvector Centrality in Industrial SAT Instances. Proc. of the 19th Int. Conf. on Principles and Practice of Constraint Programming (CP'12), 2012, pp. 348–356.
30. Ansótegui, C.; Giráldez-Cru, J.; Levy, J.; Simon, L. Using community structure to detect relevant learnt clauses. Proc. of the 18th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'15), 2015, pp. 238–254.
31. Ansótegui, C.; Bonet, M.L.; Giráldez-Cru, J.; Levy, J. On the Classification of Industrial SAT Families. Proc. of the 18th Int. Conf. of the Catalan Association for AI (CCIA'15), 2015, pp. 163–172. doi:10.3233/978-1-61499-578-4-163.
32. Baud-Berthier, G.; Giráldez-Cru, J.; Simon, L. On the Community Structure of Bounded Model Checking SAT Problems. Proc. of the 20th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'17), 2017, pp. 65–82. doi:10.1007/978-3-319-66263-3_5.

33. Ansótegui, C.; Bonet, M.L.; Giráldez-Cru, J.; Levy, J. Structure features for SAT instances classification. *J. Applied Logic* **2017**, *23*, 27–39. doi:10.1016/j.jal.2016.11.004.
34. Slater, A. Modelling More Realistic SAT Problems. Proc. of the 15th Australian Joint Conf. on Artificial Intelligence (AJCAI'02), 2002, pp. 591–602.
35. Ansótegui, C.; Bonet, M.L.; Levy, J. Towards Industrial-Like Random SAT Instances. Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09), 2009, pp. 387–392.
36. Burg, S.; Kaufmann, M.; Kottler, S. Creating Industrial-Like SAT Instances by clustering and reconstruction. Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'12), 2012, pp. 471–472.
37. Järvisalo, M.; Kaski, P.; Koivisto, M.; Korhonen, J.H. Finding Efficient Circuits for Ensemble Computation. Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'12), 2012, pp. 369–382.
38. Giráldez-Cru, J.; Levy, J. A modularity-based random SAT instances generator. Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI'15), 2015, pp. 1952–1958.
39. Giráldez-Cru, J.; Levy, J. Generating SAT instances with community structure. *Artif. Intell.* **2016**, *238*, 119–134.
40. Giráldez-Cru, J.; Levy, J. Locality in Random SAT Instances. Proc. of the 26th Int. Joint Conf. on Artificial Intelligence (IJCAI'17), 2017, pp. 638–644.
41. Selman, B.; Kautz, H.A.; McAllester, D.A. Ten Challenges in Propositional Reasoning and Search. Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97), 1997, pp. 50–54.
42. Kautz, H.A.; Selman, B. Ten Challenges Redux: Recent Progress in Propositional Reasoning and Search. Proc. of the 9th Int. Conf. on Principles and Practice of Constraint Programming (CP'03), 2003, pp. 1–18.
43. Dechter, R. *Constraint Processing*; Morgan Kaufmann, 2003.
44. Song, C.; Havlin, S.; Makse, H.A. Self-similarity of complex networks. *Nature* **2005**, *433*, 392–395.
45. Song, C.; Havlin, S.; Makse, H.A. Origins of fractality in the growth of complex networks. *Nature Physics* **2006**, *275*, 275.
46. Shanker, O. Defining Dimension of a Complex Network. *Modern Physics Letters B* **2007**, *21*, 321–326.
47. Dorogovtsev, S.N.; Goltsev, A.V.; Mendes, J.F.F. Pseudofractal scale-free web. *Physical Review E* **2002**, *65*, 066122.
48. Ravasz, E.; Barabási, A.L. Hierarchical organization in complex networks. *Phys. Rev. E* **2003**, *67*, 026112. doi:10.1103/PhysRevE.67.026112.
49. Jung, S.; Kim, S.; Kahng, B. Geometric fractal growth model for scale-free networks. *Phys. Rev. E* **2002**, *65*, 056101. doi:10.1103/PhysRevE.65.056101.