Universitat Autònoma de Barcelona

Beyond the Structure of SAT Formulas

Jesús Giráldez Crú

IIIA
Institut d'Investigació en
Intel·ligència Artificial

CSIC
Consejo Superior de Investigaciones Científicas

Beyond the Structure of SAT Formulas

by

Jesús Giráldez Crú

A dissertation presented in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy in Computer Science

Advisor:
Dr. Jordi Levy Díaz

Tutor:
Dr. Jordi González Sabaté

February 24, 2016

UAB
Universitat Autònoma de Barcelona
Departament de Ciències de la Computació

In memory of my father.

You won’t be forgotten.

Abstract

Nowadays, many *real-world* problems are encoded into SAT instances and efficiently solved by modern SAT solvers. These solvers, usually known as *Conflict-Driven Clause Learning* (*CDCL*) SAT solvers, include a variety of sophisticated techniques, such as clause learning, lazy data structures, conflict-based adaptive branching heuristics, or random restarts, among others. However, the reasons of their efficiency in solving real-world, or *industrial*, SAT instances are still unknown. The common wisdom in the SAT community is that these technique exploit some *hidden* structure of real-world problems.

In this thesis, we characterize some important features of the underlying structure of industrial SAT instances. Namely, they are the *community structure* and the *self-similar structure*. We observe that most industrial SAT formulas, viewed as graphs, have these two properties. This means that (i) in a graph with a clear community structure, i.e. having high *modularity*, we can find a partition of its nodes into communities such that most edges connect nodes of the same community; and (ii) in a graph with a self-similar pattern, i.e. being *fractal*, its shape is kept after *re-scalings*, i.e., grouping sets of nodes into a single node. We also analyze how these structures are affected by the effects of CDCL techniques during the search.

Using the previous structural studies, we propose three applications. First, we face the problem of generating pseudo-industrial random SAT instances using the notion of *modularity*. Our model generates instances similar to (classical) random SAT formulas when the modularity is low, but when this value is high, our model is also adequate to model realistic pseudo-industrial problems. Second, we propose a method based on the community structure of the instance to detect relevant learnt clauses. Our technique augments the original instance with this set of relevant clauses, and this results into an overall improvement of the efficiency of several state-of-the-art CDCL SAT solvers. Finally, we analyze the classification of industrial SAT instances into families using the previously analyzed structure features, and we compare them to other classifiers commonly used in portfolio SAT approaches.

In summary, this dissertation extends the understandings of the structure of SAT instances, with the aim of better explaining the success of CDCL techniques and possibly improve them, and propose a number of applications based on this analysis of the underlying structure of SAT formulas.

Acknowledgements

This journey is (almost) finished. However, it could not have been possible without the help and support of several people.

First and foremost, I want to thank my advisor Jordi Levy, because this work would not have been possible without his unconditional support and advice. Thank you, Jordi, for having the door of your office always open for listening and helping me, for all those conversations that always pushed me to keep trying new ideas, and for your patience, your perseverance, and your optimism. You have taught me that there is no better reward than a work well done, and you have showed me how to enjoy doing research. Above all, they are the most important lessons learnt.

I also want to thank María Luisa Bonet and Carlos Ansótegui, whose guidance and dedication have definitely meant an important foundation of this work. I remember those discussions in María Luisa's or Jordi's offices when I started my PhD, trying to cover every new idea. Thank you for tenacity and your brilliant suggestions, for your help in the hardest moments and your motivation in the calmest ones.

Another cornerstone of this work has been the possibility of visiting other research groups. I would like to thank Peter J. Stuckey (NICTA, Melbourne, Australia), Laurent Simon (LaBRI, Bordeaux, France), and Mateu Villaret (UdG, Girona, Spain), for hosting me. The short research periods with you and your groups have meant important steps in the maturity of this thesis. But above that, I am really grateful for your points of view, which increased my motivation in research. Thank you for explaining new topics, for pointing me to the correct question, and for racking my brain in every discussion.

Another mention I cannot forget is for Joaquín Borrego Díaz and Gonzalo A. Aranda, my bachelor and master degrees advisors in the University of Sevilla (Spain), and who introduced me in this world of research. This goal would not have been reached if you had not received me that day I knocked to your doors.

This thesis has been developed in the very special environment of the Artificial Intelligence Research Institute (IIIA-CSIC). From my first day in the institute, that October 3rd, 2011, all the members of this great family made me feel at home. Thank you all for creating this perfect atmosphere. To Marc, Pere, Xavi, Mari Carmen, Cerami, Andrew, Toni, Pablo, Paula, Kemo and the rest of thesis-mates, for your help and your support, in the bad and in the good moments... for all the *CFB* that have been achieved (and the ones that still have to be passed). And to Tito, Felip, Pedro, Jesús, Jar, Roberto, Julián, Félix, Anna, Carles, Ramón, Ana, and the other members of the staff, who always were there to solve any problem and made me lucky for belonging to the daily life of this institute.

I want to thank as well all those persons that supported me during this adventure. To Eduardo, for your always proximity (even with these miles away), to Alejandro, for your long friendship regardless where we are, to Carlos, Gonzalo and Eva, because the time never passes between us (*Roma dixit*). To Cristina, for making me enjoy every chat with you, to Javi, for those amazing trips around the world, and to Manuel, for those special moments in Sevilla. To Dani, Alberto, Manolo and Gabri, for our (*not only*) engineering conversations, to Belén, for your pleasant company, to Fernando, for being my worst *rover* ever, and to all my *Habitados* (and co.) people, for your often visits bringing a part of the south to Barcelona. To Fran, for making the music our passion, and to Estrella, for our *let's swing* moments. To David, for sharing those *palmerito* evenings, to Sara, for your contagious happiness, to Dani, for our eternal discussions, and to all my friends in Barcelona, for all the moments we enjoyed together. To Belén, for your strength and enthusiasm. And, in general, to everyone that shared a part of this trip, wherever we shared it!

Finally, these last lines are for my family. To my mother, Conchi, to my brother, Antonio, and to my aunt, Dolores. Thank all of you for making me know everyday that I can count on you, and for your unconditional love and support.

– Jesús Giráldez Crú

This work has been partially funded by the Spanish Ministry of Science and Innovation (currently integrated in the Spanish Ministry of Economy and Competitiveness) "Formación de Personal Investigador" (FPI) grant BES-2011-049468 within the TASSAT project (TIN2010-20967-C04-01), by the CSIC project 201450E045, and by the Razonamiento, Satisfacción y Optimización project (TIN2015-71799-C2-1-P).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The *Boolean Satisfiability Problem*, also referred as the Propositional Satisfiability Problem or just Satisfiability, (*SAT* for short) has a long history in Computer Science, and it is one of the most popular problems in this domain. SAT is the problem of deciding if there exists an assignment of the set of Boolean variables of a propositional formula such that the formula is evaluated as `true` after replacing each variable by its assigned value.

The SAT problem is one of the most studied problems in Computer Science, becoming central from both theoretical and practical sides. It has strong connections to computational complexity theory since it was the first problem to be proven NP-complete by Cook [1971] and Levin [1973]. There is no known algorithm to solve NP-complete problems in polynomial time. In fact, such algorithm does not exist if the relation P≠NP is true. For this reason, SAT was basically used during decades to prove the NP-completeness of other problems (reducing them to SAT with a polynomial time algorithm), and hence to show their intractability. Therefore, SAT was predominantly considered as a theoretical problem, without practical application in *real-world* domains, whose applicability requires efficiency.

However, the situation in this context has dramatically changed in the last years with the irruption of more advanced SAT solving techniques and their sophisticated implementations. It is worth noting that the exponential time complexity of the known algorithms solving SAT only refers to the worst-case runtime, and thus many SAT instances can be actually solved much more quickly. In fact, the advances in SAT solving have provoked that many *large* real-world problems can be efficiently solved by modern solvers. For this reason, nowadays SAT has a high impact in some application domains of Computer Science, including formal verification of both hardware and software systems; planning, scheduling and other related topics of Artificial Intelligence; or new prominent fields such as cryptography or bioinformatics, among others. The instances of these real-world, or industrial, problems may be encoded into SAT instances and (efficiently) solved by modern SAT solvers. In contrast, relatively much smaller *random* instances are still intractable by state-of-the-art SAT solving algorithms.

1.1 Motivation

It is well established the distinct nature of *random* and *industrial* SAT instances. While random formulas are (randomly) generated on demand following a known model or distribution, industrial instances are problems encodings from a real-world application. Notice that this *definition* of industrial SAT formulas is slightly ambiguous, in the sense that it includes problems from very different domains, and therefore, the resulting set of industrial instances is very heterogeneous.

Moreover, SAT solvers performance is very different when solving random or industrial SAT instances. There is no dominant technique for solving both random and industrial formulas. For this reason, SAT solvers are usually *specialized* in one kind of problems. We remark that in this thesis, the term *"SAT solver"* only refers to a complete and core SAT solver, excluding incomplete or portfolio algorithms (except when explicitly indicated).

In the SAT community, the intuition is that the distinct SAT solvers performance between solving random and industrial SAT instances comes from the existence of some kind of *hidden structure* in industrial formulas, which can be exploited by certain SAT solving techniques.

An industrial SAT instance is a representation of a higher-level application problem. In these high-level problem encodings, structure may exists, e.g., a circuit has a particular topology. However, this structure may be lost in the translation into SAT instances [Lu et al., 2003; Roy et al., 2004]. This means that one may not derive the original problem from its corresponding SAT formula. For this reason, it has been argued the existence, or not, of such structure in industrial SAT formulas, in the sense that the particularities of the original problem may be unknown in the corresponding SAT instance after the encoding process.

On the contrary, other (topological) features may arise in industrial SAT instances as a consequence of such encoding process. There is a extended agreement in the literature that such features do exist, and they are probably the reason of the success of modern SAT algorithms solving real-world problems. For instance, Williams et al. [2003] suggest that:

> *" The success of these methods appears to hinge on a combination of two factors: (1) practical combinatorial problem instances generally have a substantial amount of (hidden) tractable sub-structure, and (2) new algorithmic techniques exploit such tractable structure, through, e.g., randomization and constraint learning. "*

In the same direction, Hogg [1996] suggests that the difference between random models and industrial problems lies in the relations created by the encoding, which may create some structures into the SAT instance, being these structures very unlikely in random instances nevertheless:

> " *While* [random instances are] *simple to generate and treat theoretically, there remains the possibility that classes of problems that*

include more realistic structures or correlations among the constraints will behave in significantly different ways. Although it is difficult to characterize the detailed nature of problems that might generally arise in these cases, there are a number of plausible structural classes as well as some cases from particular applications such as class scheduling or graph colorings that arise in some numerical programs. "

Interestingly, the previous author states that characterizing the *underlying* structure of real-world problems is an intriguing problem by itself. One of the possible reasons of the hardness of such problem may be the heterogeneity of these instances, since their structure are not probably based on known models. Gomes and Selman [1997] agree with this argument:

" *Problem instances clearly have more structure than the random problem instances, but, on the other hand, they are not as regular as the structured mathematical problems.* [...] *It seems clear that random instances lack certain structure that is often present in realistic problems. On the other hand, the highly structured mathematical problems contain too much struck true from the perspective of realistic applications.* "

Therefore, structures do occur in real-world SAT instances, but they are neither completely regular nor completely random. This means that such structure may come from a regular known model, but they are probably perturbed by an element of randomness or uncertainty [Gent et al., 1999].

The success of solving this kind of problems cannot be explained without mentioning the modern SAT solving techniques. They are the so known *Conflict-Driven Clause Learning* (*CDCL*) SAT solvers.

CDCL SAT solvers are based on the classical algorithm [Davis et al., 1962; Davis and Putnam, 1960], called DPLL, a sound and complete algorithm to solve SAT. It has a depth-first search strategy; a value is assigned to each variable in each branch of the search. A conflict is found if there exists (at least) one clause with all its literals assigned, under the current partial assignment, but the clause remains unsatisfied. In that case, the algorithm backtracks. The search continues till a satisfying assignment is found (hence the formula is SAT), or till all assignments have been checked (or pruned) and none of them satisfies the formula (hence the formula is UNSAT). In order to speed-up the search, this algorithm also includes two basic rules. First, the *Unit Propagation* (UP) rule is applied when there exists a clause with all its variables but one assigned and the clause is still unsatisfied. This rule forces the value of such variable to satisfy that clause. The second rule is the *Pure Literal* (PL) elimination, which assigns all variables that only appear with one polarity (with their satisfying values). Both rules are repeatedly applied under the current partial assignment during the search.

On the basis of the DPLL algorithm, CDCL SAT solvers are built including four major components [Katebi et al., 2011]: (i) formula augmentation processes based on *learning clauses* from the conflicts, in order to reduce the search space [Marques-Silva and Sakallah, 1999; Moskewicz et al., 2001]; (ii) propagation engines implemented with *lazy data structures*, which speed-up the cost of UP [Moskewicz et al., 2001]; (iii) conflict-based adaptative branching *heuristics* [Moskewicz et al., 2001]; and (iv) random search *restarts* [Gomes et al., 1998]. Also, there exist nowadays two other components that have become crucial in the success of these solvers, and they are extensively used today in most CDCL SAT solvers. They are the database management strategy, which is commonly implemented with aggressive *clause removal policies* [Audemard and Simon, 2009], and the *preprocessing* [Eén and Biere, 2005] (and *inprocessing* [Biere, 2012]) variable elimination techniques. See [Marques-Silva et al., 2009] for a extended survey on CDCL SAT solvers.

It is important to remark that most of these techniques work experimentally on industrial SAT instances, but there is no formal proof demonstrating that these techniques always improve the performance of the solver in *all* cases, i.e., independently of the SAT instance. For example, learning a new clause may change the search because this new clause may provoke the propagation of some variables (by UP), and this results into different branchings and prunings, hence a different search. Equivalently, the same argument holds when a learnt clause is removed. Also, the addition or removal of learnt clauses may affect the heuristic, which is guided by the conflicts found in the existing clauses. Nevertheless, it is experimentally clear that these CDCL techniques improve the performance of the solver on most real-world problems. In fact, CDCL SAT solvers are the dominant techniques to solve this kind of benchmarks.

However, besides the tremendous advances in SAT solving technologies during the last years, including all techniques mentioned above, there are few works trying to understand the reasons of the power and efficiency of these techniques on solving real-world problems, as it is stated by Järvisalo and Niemelä [2008] or Audemard and Simon [2009], among others.

1.2 Challenges

Related to the structure of industrial SAT instances and its applications, there are many challenges to be achieved. In this section, we introduce the goals to be tackled in this thesis.

First of all, we focus our study in characterizing the structure of industrial SAT problems. As it is stated in [Hogg, 1996], this is an interesting problem by itself. Understanding the features of the resulting industrial SAT instances, produced by encoding higher-level problems, may help improve the state-of-the-art encodings techniques. Moreover, identifying the underlying structure that is shared by the majority of industrial formulas may result into a better classification of the set of real-world SAT instances. For instance, is it possible that some particular industrial instances are very *similar* to random problems, and consequently CDCL techniques perform worse in those instances? If this is the case, it would be natural to solve those instances using a SAT solver more *specialized* in random SAT instances, instead than a CDCL solver.

Another possible potential application of characterizing the underlying structure of industrial SAT instances is to improve portfolio SAT solving approaches [Kadioglu et al., 2011; Xu et al., 2008]. They are solution to the Algorithm Selection Problem [Rice, 1976]. This problem consists in predicting the *best* algorithm to solve a given instance in terms of some objective function, such as the expected runtime. In the context of SAT, portfolio solvers analyze some (pre-defined) features of a given instance in order to select the expected best solver to solve it, by using Machine Learning techniques[1]. Therefore, a better understanding of important features of the instance may improve the performance of these techniques.

In summary, the first goal of this work is stated in the following question:

Question 1. What is the underlying structure of industrial SAT instances?

The structure of industrial SAT instances is inevitably modified during the search by the effects of the addition and the removal of learnt clauses. Understanding the success of CDCL SAT solving techniques on industrial formulas requires understanding how the structure of such instances is affected during the search.

The main potential application of studying how the original structure of the instances is affected by CDCL technique is, in fact, improving CDCL techniques. As stated in the previous section, a lot of effort has been made, with success, in improving state-of-the-art CDCL techniques, after an extensive test-and-error process. However, there is still a lot of room to better understand why these techniques are so effective and efficient solving industrial problems. Therefore, a better comprehension of the consequences of CDCL techniques into the structure of the instance seems to be a mandatory step in order to better understand the success of these techniques, with the long-term aim of improving them.

The second challenge of this work is summarized as:

Question 2. How is this structure affected by CDCL SAT solving techniques?

Beyond the study of the underlying structure of industrial instances, stated in the previous two questions, we use it to face a number of interesting applications in the context of SAT instances and SAT solving techniques. In this dissertation, we focus on the following three applications: the generation of realistic pseudo-industrial random problems, improvements of CDCL techniques, and the classification of SAT instances into families. Let us summarize the interest of these applications.

The development of new benchmarks generation models is directly related to the analysis of the underlying structure of SAT problems. In the classical random SAT instance generation model, the difficulty of the instance is parametrized on one particular parameter: the clause/variable ratio. There exists a critical region in the value of this ratio where the hardest random instances are located [Selman et al., 1996]. On the contrary, there exists a reduced and limited number of industrial SAT benchmarks, and they do not have a parametrized degree of

[1]Portfolio solvers are trained in an offline process.

difficulty since they do not come from any know model or distribution. This was already stated by Williams et al. [2003]:

> “ *In average-case analysis, one studies the computational cost of solving problem instances drawn from a predefined problem distribution. Such an analysis can provide valuable insights, as demonstrated by the work on uniform random instance distributions (e.g. random k-SAT). However, the relatively basic distributions for which one can obtain average-complexity results appear to be quite far removed from the instance distributions one encounters in practice. In fact, formally defining the distribution of real-world problem instances is generally an open problem in itself.* ”

Therefore, the problem of generating *realistic* pseudo-industrial SAT instances is an interesting challenge. In fact, this problem was considered by Selman et al. [1997] as one of the ten most important challenges in *propositional reasoning and search* to be achieved in the following years. The original challenge was stated as:

> “ *CHALLENGE 10: Develop a generator for problem instances that have computational properties that are more similar to real-world instances.* ”

More references to this problem can be found in [Dechter, 2003; Kautz and Selman, 2003, 2007].

Nowadays, this challenge remains open. Notice that industrial SAT instances cannot be automatically generated on demand. For this reason, the existing benchmarks are known in advance[2]. In the flavor of a more fair evaluation of new solvers (e.g., in the SAT competitions), this kind of generator would be useful. Therefore, we set the following question:

Question 3. How can we generate more realistic pseudo-industrial random SAT problems?

The applicability of SAT to many real-world domains requires a continuous research and progress in SAT solving techniques. If the success of CDCL techniques solving industrial formulas is based on exploiting the underlying structure of such problems, it would be natural to develop techniques which explicitly use this structural information to improve their performance. Obviously, all advances in SAT solving technology have an important impact on other application areas. Therefore, we establish this goal:

Question 4. How can we use the underlying structure of instances to implement more efficient CDCL SAT solving techniques?

Finally, the last application we face in this thesis is the classification of SAT instances into families. This problem is addressed in portfolio approaches. The

[2]Except new benchmarks.

success of these approaches is based of the use of relevant features that allows an effective classification of instances into families, and this classification is used to predict the best algorithm to solve each family.

In the last years, many features of SAT instances have been proposed. In fact, some of them are related to the *structure* of the instance [Xu et al., 2008]. Therefore, a natural question is to analyze the relevance of the underlying structure on some classifiers commonly used in portfolio approaches, as well as its impact on the performance of these solvers. This challenge is summarized in the following question:

Question 5. What is the impact of the underlying structure on the classification of SAT instances?

1.3 Complex Networks

The inspiration of our analysis comes from the works of *complex networks*, where the general structure of real-world graphs is studied. Graph models offer a natural framework to represent and study interactions within physical or social structures. A graph is defined as a set of nodes, and a set of edges connecting them. Thus, nodes may represent the actors of the system, and edges may represent the interactions between them. Many constraint problems arise from such interactions [Hogg, 1996]. For instance, the number of interactions in the system often decreases with some measure of distance between the components, giving rise to problems with significantly more clustered or hierarchical constraints structures than would be expected from a random selection of constraints.

In our study, we use two ways to represent the SAT instances as graphs. One model represents them as bipartite graphs, where variables and clauses are nodes, and edges represent the presence of a variable in a clause. In the second model, variables are nodes, and edges between pairs of nodes (pairs of variables) indicate the existence of a clause in which those two variables appear. Other representations of SAT instances into graphs are discussed in Section 9.2.1.

The classical *Erdös-Rényi random graph model* [Erdös and Rényi, 1959] was one of the best studied during the last century, and set the basis of graph theory. In this model, edges are randomly assigned to pairs of nodes, and hence, the degree of nodes follows a binomial distribution. Random SAT instances, represented as graphs, follow this model. For instance, the variability of number of occurrences of variables is very small in large formulas. In the context of real-world networks, two main models have been defined.

A first model is the *small-world* topology, proposed by Watts and Strogatz [1998], as a new model to describe the structure of some social, biological and technological networks. They identify that these graphs are neither totally regular nor totally random. Therefore, this model is characterized by short path lengths (like random graphs) and high clustering factors (like regular lattices).

The second one is the *scale-free* model, introduced by Albert et al. [1999] to describe the structure of the World Wide Web. They show that the WWW, viewed as a graph, has a structure that cannot be described by the classical random graph model, i.e., it is very different from what one would expect if edges existed independently and at random. The name of this model comes from the fact that, in this new model, the degree of nodes follows a power-law distribution $p(k) \sim k^{-\alpha}$, and this distribution is scale-free. *Power-law (zeta and Pareto) distributions* are characterized by a big variability, consequence of a polynomially decreasing tail. A small fraction of the individuals is responsible for most of the average, in what is popularly known as the *80:20 rule* (i.e., 80% of the land is owned by the 20% of the population). Many other heterogeneous distributions are also called power-law or *heavy-tailed* when their tail decreases polynomially, in contrast to other classical distributions, like normal, Poisson, or binomial that have an exponentially decreasing tail.

In [Barabási and Albert, 1999], *preferential attachment* is presented as a method to generate scale-free graphs. In this method, the probability that a new node is attached to an existing one is proportional to the degree of this last. This mechanism results into graphs where nodes degree follows a power-law distribution. This is proposed as an explanation of the emergence of scaling in growing networks.

Experience tells us that power-law distributions are as frequent in nature, if not more frequent, as exponentially decreasing distributions. In the recent years, it has been observed that many other real-world graphs, like some social and metabolic networks, also have a scale-free structure. In other works, other interactions in real-world networks have been studied. For instances, Papadopoulos et al. [2012] suggest that the growing of real-world networks is explained by two dimensions of attractiveness. One of them is the popularity: new connections are preferentially made to the most popular nodes (i.e., preferential attachment). The other dimension is the similarity: new nodes are also likely to be connected with *similar* nodes, even when they are not popular. In [Aldecoa et al., 2015] it is described an hyperbolic graph generator, which integrates all these features.

The topology of graphs has also been studied in the context of search problems. Also, studying general properties of such problems can also help develop new benchmarks generation models. In Chapter 3, we reference related works on the underlying structure of real-world instances, and on the generation of realistic pseudo-industrial problems. It is worth mentioning two of the most important works which inspire this thesis. First, Ansótegui et al. [2009a] study the distribution of frequencies of variable occurrences and clause sizes in order to detect scale-free structures in industrial SAT instances. Second, Ansótegui et al. [2009b] propose a method to generate realistic pseudo-industrial random SAT instances, in which the number of occurrences follows a power-law distribution. Both works are briefly introduced in Section 3.3 and in Section 3.4, respectively.

1.4 Contributions

First, we focus our analysis in (better) characterizing the underlying structure of real-world SAT instances. To this effect, we study two aspects of graphs (hence, of formulas). They are: (i) the *community structure* of formulas, using the notion of *modularity*, and (ii) the *self-similarity structure* of instances and their *fractal dimension.* These analyses extend the previous work of Ansótegui et al. [2009a] about the scale-free structure of industrial SAT instances. In what follows, we present each one of these two concepts in more detail.

The notion of *modularity* was introduced by [Newman, 2004] to analyze the community structure of graphs. Having high modularity means that nodes can be grouped into sets or communities, such that, most edges connect nodes of the same community. The notion of *community* is more general than the notion of *connected component.* In particular, it allows the existence of (a few) connections between communities.

Classical random Erdös-Rényi graphs do not have community structure, thus the modularity is very low. On the other hand, scale-free formulas, which follow a power-law distribution, have community structure with higher modularity than a random graph. This modularity can be guaranteed [Dinh and Thai, 2011].

The existence of a self-similar structure would mean that after *rescaling* (replacing groups of nodes by a single node, for example), we would observe the same kind of structure. It would also mean that the diameter d^{max} of the graph with n nodes grows as $d^{max} \sim n^{1/d}$, where d is the fractal dimension of the graph, and not as $d^{max} \sim \log n$, as it occurs in random graphs or small-world graphs. Therefore, actions in some part of the graph may not *propagate* to other parts as fast as in random graphs.

In our study, we conclude that most industrial instances have a clear community structure with very high modularity. We also discuss how existing clause learning techniques and activity-based heuristics already take advantage, *indirectly,* of this community structure. Also, our analysis shows that many industrial formulas are self-similar with a small fractal dimension. We think that the self-similarity, as well as the scale-free structure and the community structure, is already present in many of the high-level problems encoded as SAT instances. We also show how this fractal dimension is affected by the addition of learnt clauses during the execution of SAT solvers.

This analysis is hold by the following publications:

1. Ansótegui, C., Giráldez-Cru, J., and Levy, J. (2012). The community structure of SAT formulas. In *Proceedings of the 15st International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 410–423.

2. Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2014). The fractal dimension of SAT formulas. In *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14)*, pages 107–121.

and (partially) covers the challenges stated in Questions 1 and 2 ("*What is the underlying structure of industrial SAT instances?*" and "*How is this structure affected by CDCL SAT solving techniques?*", respectively). In Section 9.2.4, we discuss other interesting structure features that may be analyzed to complement the previous study.

Using the previous studies on the underlying structure of industrial SAT instances, we propose three applications. Namely, they are a new model of random generation of SAT instances, a technique to detect relevant learnt clauses, and a analysis of the impact of using structure features on the classification of industrial SAT families.

First, we propose a new model of generation of *pseudo-industrial* random SAT instances based on the community structure. One important motivation for the development of such generators is to *isolate* some known properties of these real-world problems. This allows us to study the impact of these properties on the performance and behavior of SAT solvers. This is the approach used by Ansótegui et al. [2009b] to generate power-law SAT instances. Using that generator, they observe that CDCL SAT solvers focus their decisions on the most frequent variables. In the case of community structure, similar questions also arise. For instance, do SAT solvers concentrate their decisions on variables of the same (or few) communities? Do the conflicts found by the solver relate variables of the same community? How does the activity of each community evolve along the execution of the search? Answering these questions may help to better understand the different ingredients of modern SAT solvers and their impact on the solving process, with the long-term aim of improving them.

This generator allows us to generate formulas for any given value of modularity. For a high modularity, the resulting instance is more adequate to model industrial problems than the classical random model. On the contrary, with a low modularity we can generate SAT instances very similar to classical random formulas. We show that this model works appropriately for different input values of number of variables and clauses. We also show that the phase transition point SAT-UNSAT, dependent on the ratio clause/variable, does not depend on the modularity, and it is located at the same place. We also give empirical evidence that the performance of SAT solvers is consistent with the expected properties of the generated formulas, i.e., SAT solvers *specialized* in industrial problems perform better in high modular instances than SAT solvers *specialized* in random formulas, and vice versa. Finally, we use this generator to answer the questions stated in the previous paragraph. In particular, we analyze the relations between the community structure and the decisions, the conflicts and the activity of the variables that the solver manages along its execution.

This work is hold by the following publication:

3. Giráldez-Cru, J. and Levy, J. (2015). A modularity-based random SAT instances generator. In *Proceedings of the 24st International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1952–1958.

4. Giráldez-Cru, J. and Levy, J. (2016). Generating SAT instances with community structure. *Artificial Intelligence.* Accepted with revisions.

and (partially) covers the goals stated in Question 3 ("*How can we generate more realistic pseudo-industrial random SAT problems?*").

Second, we present a technique which uses the underlying structure of industrial SAT instances to improve the performance of CDCL SAT solvers. In particular, we use the community structure to detect relevant learnt clauses. Nowadays, it is accepted that more aggressive clause deletion policies may improve the solvers performance, even when there is no memory limitations. We show that augmenting SAT instances with learnt clauses does not always make them easier for the SAT solver. In fact, it makes worse the solver performance in many cases, specially in satisfiable problems. However, we identify a set of highly useful learnt clauses, and we show that augmenting SAT instances with this set of clauses contributes to improve the solver performance in many cases, especially in satisfiable formulas. These clauses are related to the community structure of the formula, and they can be computed in a fast preprocessing step. This would suggest that the community structure may play an important role in clause deletion policies.

This technique is hold by the following publication:

5. Ansótegui, C., Giráldez-Cru, J., Levy, J., and Simon, L. (2015b). Using community structure to detect relevant learnt clauses. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, pages 238–254.

and (partially) covers the objectives stated in Question 4 ("*How can we use the underlying structure of instances to implement more efficient CDCL SAT solving techniques?*"). In Section 9.2.7, we discuss other possible improvement of CDCL techniques based on the underlying structure of the instance.

The two previous applications are mainly based on our study of the community structure of SAT instances. The reason of using this structural features is discussed in Section 9.2.2.

Finally, we study how the underlying structure of industrial SAT instances can be used to classify families of benchmarks. In particular, we build some classifiers of industrial SAT families using the scale-free structure, the community structure and the self-similar structure, and we measure their effectiveness by comparing them to the same classifiers built using other sets of SAT instances features commonly used in portfolio approaches. Also, we evaluate the performance of this set of structure features when it is used in a real portfolio SAT solver, and we measure their relevance w.r.t. other known features commonly used in these approaches.

This study is hold by the following publication:

6. Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2015a). On the classification of industrial SAT families. In *Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence (CCIA'15)*, pages 163–172.

7. Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2016). Structure features for SAT instances classification. *Journal of Applied Logics.* Submitted.

and (partially) covers the objectives stated in Question 5 ("*What is the impact of the underlying structure on the classification of SAT instances?*").

See Section 1.6 for a detailed description of the organization of the rest of this dissertation.

1.5 Additional remarks

In this dissertation, all experiments are performed over the set of SAT instances publicly available for the yearly SAT races or competitions[3]. These events contribute to evaluate new solvers and new benchmarks periodically. The evaluation of the solvers is performed dividing the set of SAT instances into three main categories or tracks: *random*, *industrial* (a.k.a. *application* or *real-world*) and *crafted* (or *hard-combinatorial*). The nature of random and industrial instances is already explained. In the case of crafted formulas, this category contains problems with particular singularities, which are expected to result into a high cost for the SAT solver. These categories are sometimes sub-divided into SAT and UNSAT sub-tracks, where the satisfiability of the instances is known *a priori*. See [Järvisalo et al., 2012a] for a survey about these competitions.

In this work, we do not make any distinction between SAT and UNSAT instances, except when explicitly indicated. Also, in the *industrial* and the *crafted* tracks, SAT instances are usually grouped into families. These families contain problems from the same origin, which is usually the same domain and/or author. Common families in the industrial track of these competitions are, for instance, *crypto* (cryptography) or *bmc* (bounded model checking). In our study, we often refer to these families.

Although the benchmarks selected for the competition may vary each year, the set of yearly selected benchmarks is supposed to have similar features. For this reason, the set of benchmarks used in each experiment may vary. However, the conclusions drawn from our experimental analysis are *general*, in the sense that such conclusions remain even if the experiments are repeated with another competition benchmarks set.

All experiments were executed in a cluster of 9 nodes IBM dx360 M2, each of them with 32GB of RAM and 2 processors Intel(R) Xeon(R) CPU L55202.27 GHz, limiting all experiments to a single core and to a maximum of 4GB of RAM (except when otherwise indicated).

All software tools used to compute the results presented in this thesis are publicly available in: `http://www.iiia.csic.es/~jgiraldez/software` under the terms of the GNU General Public License (GNU GPL) or GNU Lesser General Public License (GNU LGPL).

1.6 Structure of the thesis

The rest of this thesis is organized as follows.

Some preliminaries are defined in Chapter 2. In particular, we introduce the SAT problem, as well as some important notations in the contexts of SAT and

[3]`http://www.satcompetition.org`

graphs, which are used in the rest of this work.

In Chapter 3, we reference some related works. In Section 3.1, we review those of them that analyze the underlying structure of real-world search problems, with particular interest in SAT instances, and other works on the generation of pseudo-industrial random SAT instances are reviewed in Section 3.2. Two important related works are reviewed in Sections 3.3 and 3.4: the scale-free structure of industrial SAT instances, and a power-law SAT instances generator, respectively. The approaches used in these two works are the inspiration of this thesis. For this reason, we consider interesting to dedicate a separated section to introduce them.

The analysis of the community structure in industrial SAT instances is presented in Chapter 4. We introduce the definition of modularity for graphs in Section 4.2. Then, we analyze in Section 4.3 this feature in real-wold SAT instances, representing them as graph. Finally, we investigate the effect of clause learning in the community structure in Section 4.4.

In Chapter 5, we study the self-similar structure of industrial SAT instances. We first define the fractal dimension in Section 5.2, and we analyze the relation between the fractal dimension and the diameter of a graph in Subsection 5.2.1. An exhaustive analysis of the self-similarity of real-world SAT instances and the effects of clause learning is presented in Sections 5.3 and 5.4, respectively.

We present a random SAT instances generator, called *Community Attachment*, based on the community structure in Chapter 6. The model is presented and validated in Sections 6.2 and 6.3, respectively. The phase transition phenomenon of this model is studied in Section 6.4. Finally, we analyze the performance of some SAT solvers on this model in Section 6.5, and some components of a CDCL SAT solver using this generator in Section 6.6.

In Chapter 7, we introduce a new technique to detect relevant learnt clauses. First, we analyze how the original community structure is destroyed by the effect of clause learning, and we present an analysis of the relevance of learnt clauses in Sections 7.2 and 7.3, respectively. Our technique is presented in Section 7.4, and evaluated in Section 7.5.

The analysis of the classification of SAT instances using the underlying structure is presented in Chapter 8. In particular, we study the effectiveness of these structure feature to correctly classify industrial SAT instances into families in Section 8.2. Then, we analyze the impact of using them in a real port-folio SAT solver in Section 8.3. Finally, we measure the relevance of these structure features in Section 8.4.

Finally, we conclude in Chapter 9 with a summary of the work presented in this thesis, a discussion of some future works, and some final remarks, in Sections 9.1, 9.2, and **??**, respectively.

Chapter 2

Preliminaries

In this chapter, we introduce some notations and formal definitions used in the rest of this dissertation.

2.1 The Boolean Satisfiability Problem (SAT)

A *Boolean variable* is a variable having two possible values: `true` or `false`. A *Boolean formula* is a formula composed of a combination of Boolean variables and the logical operators of conjunction, disjunction, and negation ($\{\wedge, \vee, \neg\}$). From now on, we use the term *formula* (or *instance*) to refer a Boolean formula, unless otherwise indicated.

A *literal* is a Boolean variable or its negation, and a *clause* is a disjunction of literals. The length, or size, of a clause is the number of literals such clause contains. A Boolean formula is said to be in *Conjunctive Normal Form* (*CNF*) when it is written as a conjunction of clauses, i.e., a conjunction of disjunction of literals. Notice that any formula can be translated into CNF in linear time. A k-CNF is a formula whose clauses have exactly length k.

An *interpretation* of a formula is an assignment of a value to the variables of such formula. An interpretation satisfies a formula when its evaluation is `true`. The *Boolean Satisfiability Problem* (*SAT*) is the problem of deciding if there exists an interpretation that satisfies a given formula.

2.2 Graph representations of SAT instances

An undirected weighted graph is a pair (V, w) where V is a set of vertexes and $w : V \times V \to \mathbb{R}^+ \cup \{0\}$ is the edge weight function that satisfies $w(x, y) = w(y, x)$. This definition generalizes the classical notion of graph (V, E), where $E \subseteq V \times V$, by taking $w(x, y) = 1$ if $(x, y) \in E$ and $w(x, y) = 0$ otherwise. The degree of a vertex x is defined as $deg(x) = \sum_{y \in V} w(x, y)$. A bipartite graph is a tuple (V_1, V_2, w) where V_1 and V_2 are two disjoint sets of vertexes, and $w : V_1 \times V_2 \to \mathbb{R}^+ \cup \{0\}$ is the edge weight function.

Given a SAT instance, we construct two graphs, following two models. In the *Variable Incidence Graph* model (*VIG*, for short), vertexes 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. Notice also that there can be more than one clause relating two given variables. To preserve this information we put a higher weight on edges connecting variables related by more clauses. Moreover, to give the same relevance to all clauses, we ponder the contribution of a clause to an edge by $1/\binom{n}{2}$. This way, the sum of the weights of the edges generated by a clause is always 1.

Definition 2.1 (Variable Incidence Graph (VIG))**.** Given a SAT instance Γ over the set of variables X, its variable incidence graph is a graph (X, w) with set of vertexes the set of Boolean variables, and weight function:

$$w(x, y) = \sum_{\substack{c \in \Gamma \\ x, y \in c}} \frac{1}{\binom{|c|}{2}}$$

where $|c|$ is the length of the clause c.

In the *Clause-Variable Incidence Graph* model (*CVIG*, for short), vertexes represent either variables or clauses, and edges represent the occurrence of a variable in a clause. Like in the VIG model, we try to give the same relevance to all clauses, thus every edge connecting a variable x with a clause c containing it has weight $1/|c|$. This way, the sum of the weights of the edges generated by a clause is also 1 in this model.

Definition 2.2 (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 $(X, \{c \mid c \in \Gamma\}, w)$, with vertexes the set of variables and the set of clauses, and weight function:

$$w(x, c) = \begin{cases} 1/|c| & \text{if } x \in c \\ 0 & \text{otherwise} \end{cases}$$

Other graph representations of SAT instances, which may be useful in future works, are discussed in Section 9.2.1.

Chapter 3

Related Work

In this chapter, we summarize some related works on the underlying structure of real-world problems, with special emphasis on SAT instances, and its relations to the cost of solving such problems, and also on the generation of random SAT instances with properties of real-world benchmarks. First, we introduce general background about structure in search problems in Section 3.1. Then, we review some related works on the generation of pseudo-industrial random SAT instances in Section 3.2. Finally, we dedicate two separate sections to review two interesting works that inspired this thesis. On the one hand, the analysis of the scale-free structure of industrial SAT formulas [Ansótegui et al., 2009a] is summarized in Section 3.3. In that work, it is studied whether the number of variable occurrences and clause size follow power-law distributions. It is worth noting that all the tools needed to compute these structure features were re-implemented and integrated in our graph structure features software. Moreover, empirical results have been evaluated in a set of benchmarks distinct from the one used in [Ansótegui et al., 2009a], complementing the conclusions drawn in that work. On the other hand, we review the pseudo-industrial random SAT instances generator based on the scale-free structure of real-world problems [Ansótegui et al., 2009b] in Section 3.4. This generator produces random SAT instances with a power-law distribution in the number of variable occurrences. Also, authors use this generator to analyze the performance of CDCL SAT solving techniques, with the aim of better understanding their success on application benchmarks. We use this approach to propose a new pseudo-industrial random SAT instances generator based on the notions of community structure and modularity (instead of scale-free structure). This generator is presented in Chapter 6.

3.1 Structure in search problems

The topology of graphs have a major impact on the cost of solving search problems on these graphs. Gent et al. [1999] analyze the impact of a small-world topology on the cost of finding solutions to graph coloring problems. Walsh

[2001] does the same in the case of scale-free graphs. Walsh [1999] analyzes the small world topology of many graphs associated with search problems in Artificial Intelligence. He also shows that the cost of solving these search problems can have a *heavy-tailed distribution*. Therefore, we can expect that SAT solving, viewed as a search process on a graph (the formula), will be affected by the topology of this graph.

As we mentioned before, Ansótegui et al. [2009a] studied the scale-free structure of real-world SAT instances, analyzing whether the number of variable occurrences and clause size follow power-law distributions. We review this work in Section **??**.

The power-law distributions also appear in other aspects of SAT solving. Gomes et al. [2004] show that the CPU time of the different executions (with different random variable selection) of a solver on a formula follows a power-law distribution. In [Gomes and Selman, 1997] and [Gomes et al., 1998], it is proposed the use of *randomization* and *rapid restart* techniques to prevent solvers from falling on the long tail of power-law distributions.

Biere and Sinz [2006] show that many SAT instances can be decomposed into *connected components*, and how to handle them within a SAT solver. They discuss how this structure into *connected components* can be used to improve the performance of SAT solvers. Since our notion of community structure is more general, it might be more practical to analyze and improve the performance of SAT solvers.

Our work on community structure of SAT instances [Ansótegui et al., 2012] has already had impact on the SAT community. A pioneering work on using community structure to speed-up solvers was presented by Martins et al. [2013]. In particular, they propose to solve Maximum Satisfiability formulas by partitioning them according to the community structure and adding incrementally to the MaxSAT solver the sets of clauses related to communities. This solution is impoved in [Neves et al., 2015]. Sonobe et al. [2014] use the partition obtained with the community structure to improve the performance of a parallel SAT solver. Newsham et al. [2014] show that the community structure is correlated with the runtime of CDCL SAT solvers. In [Newsham et al., 2015], a tool for visualizing the community structure of SAT instances is presented, and some empirical observations about the evolution of such structure regarding CDCL SAT solvers are enumerated. Ganian and Szeider [2015] use the community structure as inspiration to define a structural parameter of CNF instances, and they define tractable algorithms to solve SAT and #SAT fixing such structural parameter.

Other structural features of SAT instances have also been studied in other papers. For instance, Katsirelos and Simon [2012] study the centrality of variables picked by a CDCL solver. Simon [2014] use observations from the SAT solver performance on industrial problems to better understand some of the unsatisfiability proof characteristics. Other notions of structure have been also used to better understand the branching heuristics of modern SAT solvers and improve them [Liang et al., 2015a,b].

3.2 Pseudo-industrial random SAT generation

There already exist some works facing the problem of generating *pseudo-industrial* random instances. We distinguish between those works focused on particular problems domains from those others that generalize common properties of industrial instances.

In the case of particular problems domains, we have the following works. Gomes and Selman [1997] proposed a new model of SAT benchmarks resulting from introducing random perturbations into structured problems. These benchmarks encode instances of the partial latin square completion problem, with some randomly selected pre-assigned values. Achlioptas et al. [2000] refined the previous model to guarantee the generation of satisfiable instances. Gent et al. [1999] introduced a method, called *morphing*, for introducing randomness into structured problems, resulting into problems with small-world topology. Järvisalo et al. [2012c] proposed an instance generator based on finding optimal circuits of Boolean functions. In general, these methods do not explicitly generate SAT instances with community structure. However, this feature may appear for certain input parameters.

On the other hand, other works focus on general properties shared by the majority of real-world problems. Burg et al. [2012] presented a generator that combines subparts of real-world instances to create new ones. Ansótegui et al. [2009b] used the notion of scale-free graph to generate SAT instances where the number of variable occurrences follows a power-law distribution. We review this generator in Section 3.4. Newsham et al. [2014] proposed a random 3-CNF generator which divides the set of variables into groups, and uses a certain probability to decide for each clause whether a literal is selected from the same group than the previous one or not (randomly selected in the case of the first literal of the clause). This way, a high value in this probability generates very modular instances. Slater [2002] proposed a method to generate SAT instances with *small-world* topology. This feature is also very common in many real-world networks, and it is characterized by a small typical distance and a high clustering coefficient. The proposed model is characterized by n variables, m clauses, c clusters and a parameter p. They generate c independent 3-SAT formulas, each one having n/c variables and $(1-p)\,m/c$ clauses, and then add $p\,m$ link clauses using the entire set of variables, in the spirit of the techniques used traditionally to generate small-world graphs.

Notice that the previous 4 generators are able to create random SAT instances with community structure under certain circumstances. In the first method, this happens when the input formulas have such structure, which is the case in real application benchmarks. The scale-freeness of the second method is very characteristic in real-world instances, but it is not a sufficient condition to guarantee community structure. In the third method, a high probability produces very dense groups, i.e., a very modular structure. And this is also the case in the fourth method, since the small-world topology implies, in general, a clear community structure. However, none of these solutions allow to control the value of the modularity in the resulting formula (i.e., how clear is its community structure). For this reason, we present a new generator in Chapter 6.

3.3 The scale-free structure of SAT instances

In this section, we introduce the study of the scale-free structure of industrial SAT instances presented in [Ansótegui et al., 2009a]. In particular, they analyze whether the number of variable occurrences and clause size follow power-law distributions. It is worth noting that all the software needed to compute this structure feature was re-implemented and integrated in our graph structure features tool. Moreover, empirical results have been evaluated in a set of industrial SAT benchmarks different from the one used in [Ansótegui et al., 2009a], complementing the conclusions drawn in that work.

In scale-free graphs, the degree K of nodes follows a power-law probability distribution $f^{pow}(k) = P(K{=}k) = c\,k^{-\alpha}$, at least asymptotically. The exponent α has typically values inside $[2,3]$, although occasionally it may lie outside these bounds. This distribution diverges at zero. Since the degree of nodes has to follow this distribution *asymptotically*, we assume that there is a lower bound k_{min} for the values of k, also called *heavy tail*, from where we get this behavior. In the discrete case (the one that concerns us), the normalizing constant is $c = 1/\zeta(\alpha, k_{min}) = 1/\sum_{i=0}^{\infty}(i + k_{min})^{-\alpha}$, where ζ is the Hurwitz zeta function. For big values of k_{min} we can approximate this distribution using the continuous version. In this case the probability density function is $f^{pow}(k) = \frac{\alpha-1}{k_{min}}\left(\frac{k}{k_{min}}\right)^{-\alpha}$, and the complementary cumulative distribution is $F^{pow}(k) = \int_{k'=k}^{\infty} f^{pow}(k')\,\mathrm{d}k' = \left(\frac{k}{k_{min}}\right)^{-\alpha+1}$.

In the (unweighted) CVIG model, the degree of a variable-node corresponds to the number of occurrences of this variable, and the degree of a clause-node to the size of this clause.

Definition 3.1. Given a formula with n variables and m clauses, let $f_v^{real}(k)$ be the number of variables that have a number of occurrences equal to k, divided by n. Similarly, $f_c^{real}(k)$ is the number of clauses of size k divided by m.
We can also define the (complementary) cumulative versions of these functions as $F_v^{real}(k) = \sum_{i \geq k} f_v^{real}(i)$ and $F_c^{real}(k) = \sum_{i \geq k} f_c^{real}(i)$.

Notice that in the previous definition, the label *real* is added to emphasize that these functions come from empirical data. Notice also that, assuming that there are no empty clauses and all variables occur somewhere, hence $F_v^{real}(1) = F_c^{real}(1) = 1$.

The purpose is to check if, for some values of α_v and α_c, we have $f_v^{real}(k) \approx c\,k^{-\alpha_v}$ and $f_c^{real}(k) \approx c\,k^{-\alpha_c}$. Notice that, applying logarithms to both sides, we get $\log f(k) = \log c - \alpha \log k$. Therefore, if $f_v^{real}(k)$ and $f_c^{real}(k)$ are power-law, representing them as a function of k with double-logarithmic axes, we should get closed to a straight line with slope $-\alpha$.

Some papers estimate the value α by linear regression of $\log f(k)$ or $\log F(k)$ [Li et al., 2005]. However, the most reliable estimation techniques for the exponent α are based on the method of maximum likelihood [Clauset et al., 2009]. In this method, to estimate the value of α for a collection of empirical data $k_1, \ldots, k_n$, we compute the value of α that maximizes the probability that the data were drawn from the model. For the continuous case, this probability is:

$$P(k_1, \ldots, k_n \mid \alpha) = \prod_{i=1}^{n} \frac{\alpha - 1}{k_{min}} \left(\frac{k_i}{k_{min}} \right)^{-\alpha} \tag{3.1}$$

Taking logarithms (since the maximum will be in the same place), and then taking derivatives and making the function equal to zero, we can compute in the discrete case a good approximation of α for big values of k_{min}:

$$\hat{\alpha} = 1 + \frac{n}{\sum_{i=1}^{n} \log \frac{k_i}{k_{min} - 1/2}} \tag{3.2}$$

Notice that the estimated α depends on k_{min}. Moreover, $\hat{k}_{min}$ corresponds to the estimation of the point where we assume that the distribution should start an asymptotic power-law behavior. In order to estimate this value, we define the maximal distance $d(k_{min})$ between the real and the cumulative distribution functions, only considering values of k greater than k_{min}. Therefore, we select the value of k_{min} that minimizes this maximal distance $d(k_{min})$[1]. Additionally, we also set an upper bound on the values of k_{min}. Otherwise, we could consider power-law some distributions that only exhibit this behavior for a very few big values of k.

When it is said that arity of nodes *seems* to follow a power-law distribution, it is emphasized the *seems* because it is obvious that SAT formulas, as well as the WWW and other scale-free graphs, are not randomly generated. Therefore, we do not expect the arity of nodes to follow *exactly* any distribution. However, we want to check if some distribution fits the data better than others. In particular, we will check if the tail of the empirical data fits better a power-law distribution (i.e., a polynomial decreasing tail) than an exponential distribution (i.e., an exponential decreasing tail), as an indicator of a scale-free graph structure.

The probability density function for an exponential distribution has the form $c\, e^{-\beta\, x}$. Calculating the constant, for the discrete case, we get $f^{exp}(k; \beta, k_{min}) = (1 - e^{-\beta})\, e^{-\beta\,(k - k_{min})}$ and its cumulative function $F^{exp}(k) = e^{-\beta(k - k_{min})}$. In this case the estimation of the β parameter by the method of maximum likelihood gives:

$$\hat{\beta} = \log \left(\frac{n}{\sum_{i=1}^{n} (k_i - k_{min})} + 1 \right) \tag{3.3}$$

and the value of $\widehat{k_{min}}$ is estimated as in the case of power-law distributions.

We address the reader to the original work of Ansótegui et al. [2009a] for more details about the computation of these estimations and their corresponding distributions.

[1]We use the distribution function after normalization. Therefore, the distance $d(k_{min})$ does not necessarily decrease for bigger values of k_{min}.

Variables (v)							
		Power-law			Exponential		
Family	#inst	α_v	$k^{pow}_{min,v}$	d^{pow}_v	β_v	$k^{exp}_{min,v}$	d^{exp}_v
2d-strip-packing	5	1.213	25	0.295	0.0001	29	0.155
bio	5	**2.614**	5	**0.139**	0.0036	12	0.310
crypto-aes	11	**1.782**	9	**0.116**	0.0172	13	0.154
crypto-des	9	**3.962**	5	**0.116**	0.0547	12	0.203
crypto-gos	30	1.482	6	0.254	0.0141	1	0.226
crypto-md5	11	2.119	4	0.235	0.0697	6	0.231
crypto-sha	30	1.264	4	0.492	0.0049	8	0.357
crypto-vmpc	8	10.65	388	0.215	0.0232	388	0.208
diagnosis	26	2.650	7	0.154	**0.1271**	4	**0.130**
hardware-bmc	3	**2.934**	11	**0.109**	0.0925	8	0.112
hardware-bmc-ibm	4	**2.644**	9	**0.020**	0.0915	7	0.213
hardware-cec	30	**2.647**	9	**0.107**	0.2796	4	0.313
hardware-velev	21	**1.884**	10	**0.057**	0.0134	3	0.436
planning	25	2.054	4	0.211	0.0742	3	0.363
scheduling	30	1.890	11	0.193	**0.0335**	11	**0.056**
scheduling-pesp	30	**2.224**	11	**0.085**	**0.0551**	6	**0.085**
software-bit-verif	14	**4.021**	7	**0.117**	0.2961	4	0.236
software-bmc	3	3.756	9	0.158	0.1833	2	0.368
termination	5	2.584	5	0.180	0.1236	4	0.204
SAT Comp. 2013	300	**2.303**	7	**0.094**	0.0606	1	0.334
random	50	4.391	11	0.148	**0.262**	11	**0.110**

Table 3.1: Most likelihood values of α and β estimated for a power-law and an exponential distribution, and their respective parameters k_{min} and d, that best fit the number of variable occurrences v. In bold we remark the smallest distance between the real and the fitted distributions.

3.3.1 Experimental analysis of the scale-free structure

In this section, we analyze the scale-free structure of some industrial SAT instances. To this purpose, we compute $f_v^{real}(k)$ and $f_c^{real}(k)$, as well as their cumulative functions, of some industrial benchmarks. First, we observe that functions $f_v^{real}(k)$ and $f_c^{real}(k)$ are similar for different instances of the same family. Thus, we decide to group the instances by families, assuming that all formulas of the same family follow the same probability distribution. We define $f^{real}(k)$ for a given a family as the sum of the functions $f^{real}(k)$ for each formula of the family, conveniently normalized. Therefore, we treat a family of formulas as a unique formula[2].

Later, we compute the power-law and exponential distributions that best fit the functions $f^{real}(k)$, and we calculate the distance d^{pow} between $F^{real}(k)$

[2]Notice that, under the assumption that all formulas follow the same probability distribution, the sum of $f^{real}(k)$ also follows the same distribution.

Clauses (c)								
		Power-law			Exponential			
Family	#inst	α_c	$k_{min,c}^{pow}$	d_c^{pow}	β_c	$k_{min,c}^{exp}$	d_c^{exp}
2d-strip-packing	5	**4.435**	8	**0.115**	0.2568	6	0.141
bio	5	6.478	4	0.088	**2.3446**	3	**0.010**
crypto-aes	11	**4.875**	6	**0.079**	1.0281	2	0.095
crypto-gos	30	1.788	2	0.281	0.2346	2	0.345
crypto-sha	30	2.059	3	0.364	0.2452	3	0.362
crypto-vmpc	8	17.599	33	0.152	**4.0090**	3	**0.018**
diagnosis	26	4.346	5	0.078	**1.3122**	2	**0.034**
hardware-bmc-ibm	4	**3.124**	4	**0.026**	0.3238	5	0.042
hardware-cec	30	7.829	4	0.148	**7.6735**	3	**0.000**
hardware-velev	21	**2.149**	10	**0.088**	0.0494	4	0.114
planning	25	4.221	2	0.198	**1.8074**	2	**0.144**
scheduling	30	4.345	11	0.128	**2.5820**	3	**0.069**
software-bmc	3	2.241	6	0.108	**6.4061**	3	**0.001**
termination	5	6.538	10	0.113	**0.5181**	10	**0.093**
SAT Comp. 2013	300	**2.548**	10	**0.067**	0.8819	2	0.144

Table 3.2: Most likelihood values of α and β estimated for a power-law and an exponential distribution, and their respective parameters k_{min} and d, that best fit the clause size c. In bold we remark the smallest distance between the real and the fitted distributions.

and the estimated $F^{pow}(k;\alpha,k_{min})$, and the distance d^{exp} between $F^{real}(k)$ and the estimated $F^{exp}(k;\beta,k_{min})$. When $d^{pow} < d^{exp}$, we say that the power-law distribution fits better than the exponential distribution. If the distance d^{pow} between the observed data and the distribution is smaller than 0.15, then it is plausible that the data follows such power-law distribution. We use the two previous criteria to state that a family of formulas has a scale-free structure. It is also important to compare the value of k_{min} obtained in each estimation, noted k_{min}^{pow} and k_{min}^{exp}. A big value of k_{min} means that we need to discard a lot of values of $F^{real}(k)$ to fit the distribution, and it must be taken as a point against the fitted distribution. For this reason, we allow to disregard at most 10 distinct values of the distribution. Also, a value of α far away from the interval $[2,3]$ must be read as a point against the scale-free structure.

In Tables 3.1 and 3.2, we present the estimations of the parameters of the power-law and exponential distributions for number of variables occurrences and clause size of the industrial SAT instances of the SAT Competition 2013. We have also extended the study to a family of 50 random 3-CNF instances of 10^3 variables close to the phase transition point (clause/variable ratio $m/n = 4.25$), and to the *heterogeneous* family composed by the 300 instances used in this competition. Families whose clause size is regular are not reported in Table 3.2.

In Figure 3.1, we plot the distributions $F_v(k)$ and $F_c(k)$ versus k of some

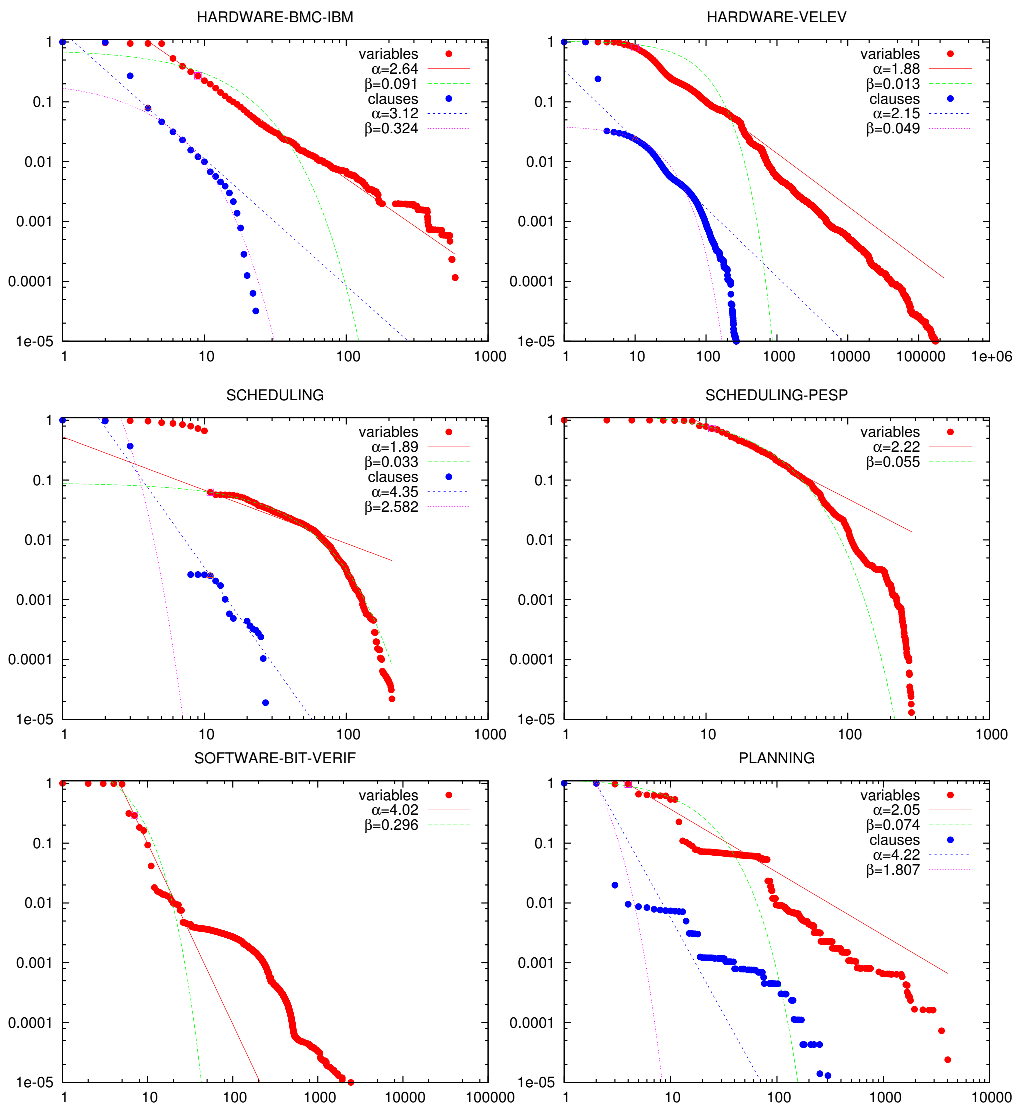


Figure 3.1: Plot of $F_v^{real}(k)$ and $F_c^{real}(k)$, and their respective power-law and exponential estimations (characterized by α and β, respectively), for some industrial SAT families of formulas. In families where all clauses are small, we have avoided the representation of $F_c^{real}(k)$.

representative industrial families, as well as the estimated power-law and exponential distributions that best fit them. In Figure 3.2, we also plot the distributions for the heterogeneous family of the SAT Competition 2013, and the

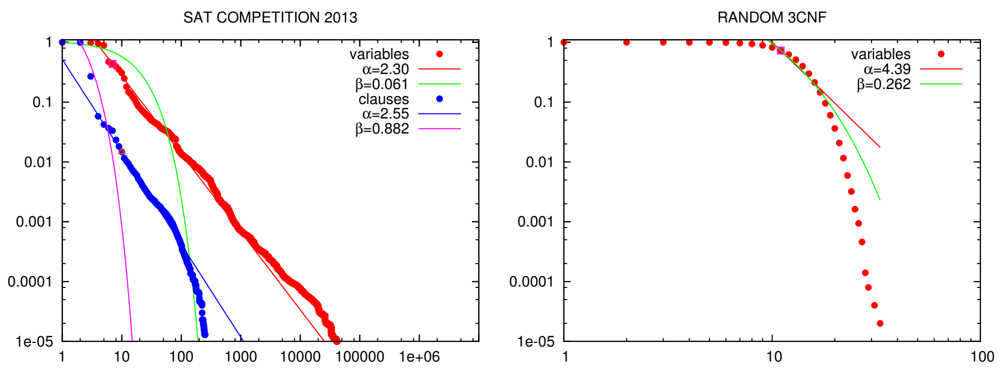


Figure 3.2: Plot of $F_v^{real}(k)$ and $F_c^{real}(k)$, and their respective power-law and exponential estimations (characterized by α and β, respectively), for the formulas of the SAT Competition 2013 and random 3-CNF instances.

random 3-CNF formulas. We use double-logarithmic axes in all plots. Thus, in this representation, power-law functions become straight lines with slope $-\alpha+1$, and exponential functions are concave curves multiplied by $-\beta$. Notice that this representation creates an undesirable *visual effect* in the tail, since $F^{real}(k)$ and $F^{pow}(k)$ *seem* to be further away than they actually are. For instance, for the family *hardware-velev*, $F_v^{real}(k)$ *seems* to fit very well a power-law distribution (F_v^{pow}) for small values of K ($k \leq 500$), but it *seems* to diverge from this distribution for bigger values. However, the maximum distance d_v^{pow} between F_v^{real} and F_v^{pow} is found in $k = 30$ ($d_v^{pow} = 0.057$). Therefore, even when the tail does not *seem* to fit this power-law distribution, this is just an undesirable visual effect produced by this kind of representations.

We can conclude that for the families *bio*, *crypto-aes*, *crypto-des*, *hardware-bmc*, *hardware-bmc-ibm*, *hardware-cec*, *hardware-velev* and *software-bit-verif* the number of variable occurrences follows a power-law distribution. For the family *scheduling-pesp*, we obtain $d^{pow} = d^{exp}$, thus we cannot assure that a power-law distribution fits better these data. In the case of clause size, only the families *2d-strip-packing*, *crypto-aes*, *hardware-bmc-ibm* and *hardware-velev* seem to follow power-law distributions. Therefore, in general, the variable occurrences follow a power-law distribution in more families than the clause size. The value of α_v for variables is also smaller than the α_c for clauses, and in some cases they fall out of the interval $[2, 3]$. We think that the explanation for this phenomena is that, when the formulas are encoded, people try to avoid the use of very long clauses, since they decrease the propagation power in SAT solvers. We also observe that some families, like *crypto-gos*, do not seem to follow any particular distribution.

In the random 3-CNF formulas, the exponential distribution fits better than the power-law, although the distance d^{pow} is surprisingly small. Looking at the plot of the SAT Competition 2013 heterogeneous family, we see that the data fits better the power-law distribution than other homogeneous families. In this

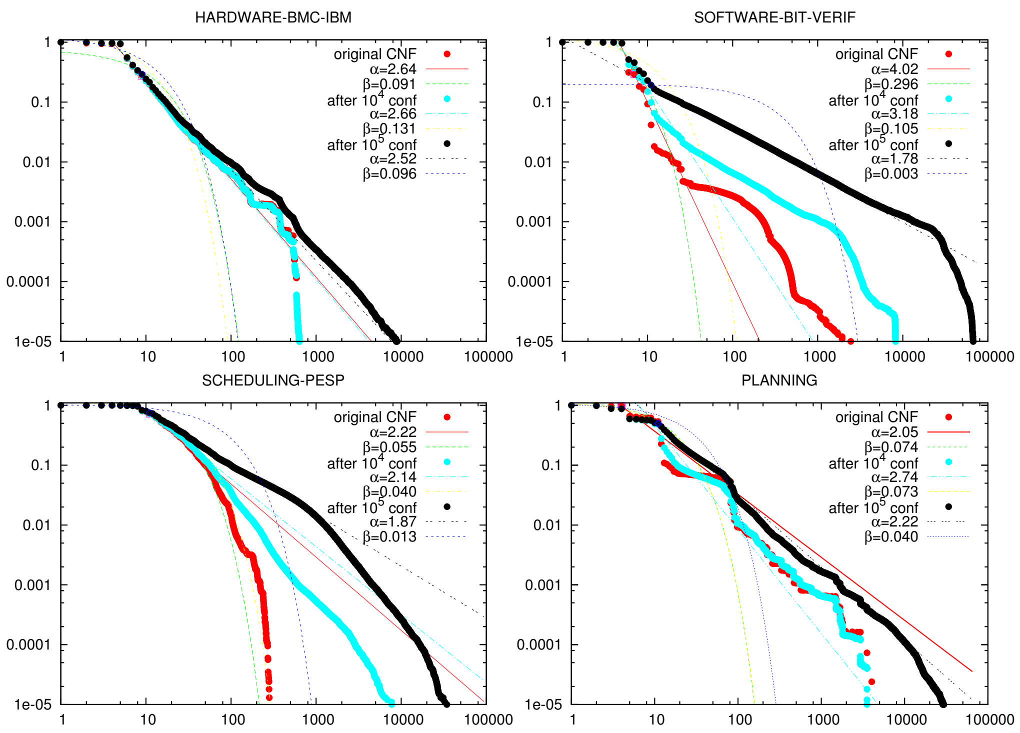


Figure 3.3: Plot of $F_v^{real}(k)$, and their respective power-law and exponential estimations (characterized by α), for some industrial families of formulas, using original CNFs and formulas after 10^4 and 10^5 conflicts.

case, we have to take into account that the addition of so many instances, by a kind of law of the big numbers, tends to make distributions smoother. The values of α that we get are $\alpha_v = 2.303$ for variable occurrences and $\alpha_c = 2.548$ for clause size. As in some homogeneous families, we observe that the value of α_c in the case of clause size is bigger than the value of α_v for variable occurrences, and they both fall in the limits of the interval $[2, 3]$.

3.3.2 The scale-free structure during SAT solver search

During the search, CDCL solvers augment the original formula by adding new (learnt) clauses, due to the learning mechanism they incorporate. We want to answer the question of what kind of formula the solver *sees* during the search. The question is important because, assuming industrial SAT instances have a kind of *structure*, during the execution of the CDCL, the original structure can change, and it can turn into a totally different one.

We conduct some experiments to answer the previous question, using the

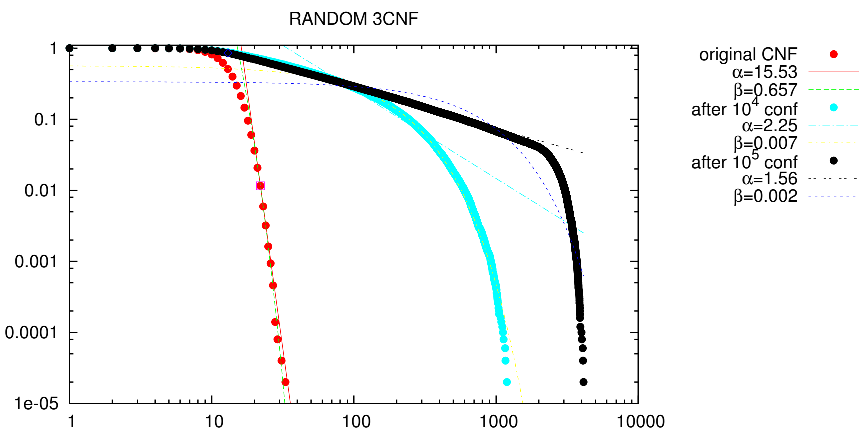


Figure 3.4: Plot of $F_v^{real}(k)$, and their respective power-law and exponential estimations (characterized by α and β, respectively), for some 3-CNF formulas, using original CNFs and formulas after 10^4 and 10^5 conflicts.

CDCL solver MiniSAT [Eén and Sörensson, 2003], and analyzing the formula at certain depths in the search. In particular, we keep the formula the solver is seeing after some conflicts, and we repeat the analysis of the scale-free structure presented in the previous section in such formulas. Notice that each conflict generates a new learnt clause, so these new formulas are, in general[3], the original formula plus learnt clauses from conflicts.

In Fig. 3.3, we plot the distributions of number of variable occurrences $F_v(k)$ of the resultant formulas after 10^4 and 10^5 conflicts for the industrial families *hardware-bmc-ibm* (top left), *software-bit-verif* (top right), *scheduling-pesp* (bottom left), and *planning* (bottom right). Analyzing the evolution of these distributions, we observe two phenomena. First, families that originally fit better power-law distributions (e.g., *hardware-bmc-ibm* or *software-bit-verif*) still fit better these distributions during the search, but the estimated exponent α_v^X is smaller after a big number of conflicts X. For instance, in the family *hardware-bmc-ibm*, the exponent is originally $\alpha_v^0 = 2.64$, and it turns into $\alpha_v^{10^5} = 2.52$ after 10^5 conflicts. Moreover, the error d^{pow} almost does not change: $d^{pow,0} = 0.020$ and $d^{pow,10^5} = 0.022$. Since $d^{pow,10^5} < d^{exp,10^5}$, we conclude that the new formula is still scale-free. Second, families that originally does not clearly fit a power-law distribution (e.g., *scheduling-pesp*) start to fit better this distribution after a certain number of conflicts. For the family *scheduling-pesp*, the original error $d^{pow,0} = 0.085$ turns into $d^{pow,10^5} = 0.038$ after 10^5 conflicts, while $d^{exp,10^5} = 0.388$.

We can conclude that CDCL solvers preserves the scale-free structure in formulas that already had this structure, whereas they turn non-scale-free formulas

[3]In MiniSAT, unary learnt clauses are propagated and this can lead to remove original clauses.

into scale-free ones. Moreover, the estimated exponent α_v that characterizes power-law distributions becomes smaller as new learnt clauses are added. To explain this phenomenon recall that solvers like MiniSAT decide on the most active variables. Hence, they are more likely to appear in new learnt clauses. This creates an effect of *rich get richer* that has been proposed as a mechanism for creation of scale-free networks [Barabási and Albert, 1999].

Finally, we want to check if the same effect can be observed also in random formulas. In Fig. 3.4, we repeat the same experiment for the family of random 3-CNF instances. We observe the same phenomenon described before. The original random formulas do not fit a power-law distribution because: i) the error is bigger than for an exponential distribution $d^{pow,0} = 0.148 > d^{exp,0} = 0.110$, and ii) the value of the exponent $\alpha^0 = 4.39$ is too big to consider a power-law distribution. On the contrary, it fits quite well an exponential distribution, as the abrupt decay in the function $F_v^{real}(k)$ suggests. This behavior is still observable after 10^4 conflicts ($d^{pow,10^4} = 0.101 > d^{exp,10^4} = 0.028$), but in this case, the abrupt decay is less clear for small values of k. After 10^5 conflicts, the function $F_v^{real}(k)$ fits better a power-law distribution because $d^{pow,10^5} = 0.041 < d^{exp,10^5} = 0.216$, and the value of the exponent $\alpha^{10^5} = 1.563$ is small enough to consider this kind of distribution. Therefore, we conclude that CDCL induces the emergence of a scale-free structure, even in the cases where it was not present at all in the original formula.

3.4 Power-law SAT instances generators

In this section, we introduce the pseudo-industrial random SAT instances generators presented by Ansótegui et al. [2009b]. These generators produce random SAT instances that have computational features shared by the majority of real-wold benchmarks. In particular, they propose several generators that produce random SAT formulas whose number of variable occurrences follows a power-law distribution. This is a very common property in application benchmarks, as we have seen from the analysis of the scale-free structure of industrial SAT formulas (presented in [Ansótegui et al., 2009a] and described in the previous section).

Let us define first the two classical random models of generating k-CNF instances. Namely, the are the uniform and the regular models [Boufkhad et al., 2005]. Both models have as parameters the number of variables n, the number of clauses m, and the clause size k. In the uniform model $F_k(n, m)$, the m clauses are uniformly and independently chosen from the set of $2^k\binom{n}{k}$ non-trivial clauses of size k. The regular model $F_k^{reg}(n, m)$ consists in selecting uniformly a formula from the set of formulas with m non-trivial clauses of size k, and n variables where all literals have nearly the same number of occurrences, i.e., either $\lfloor \frac{km}{2n} \rfloor$ or $\lfloor \frac{km}{2n} \rfloor + 1$. Therefore, while in the uniform model one cannot know the exact number of occurrences of a certain variable, this number is fixed in the regular model.

Given a continuous probability distribution ϕ with domain $[0, 1]$, we can generate a family of probability distributions $P(X = i; n)$, with discrete do-

Algorithm 1: Power-law k-CNF generator

```
Input: int $n$, $m$, $k$; real $\alpha$;
Output: $k$-CNF $\Gamma$, with $n$ variables and $m$ clauses
$\Gamma := \emptyset$;
for $j = 1$ to $m$ do
    $C_j := \emptyset$;
    repeat
        $C_j := \emptyset$;
        for $i = 1$ to $k$ do
            $p := rand()$;                              // rand: real in $[0, 1)$
            $x := 1$;
            while $p > P(X = x; \alpha, n)$ do          // See Equation 3.7
                $p = p - P(X = x; \alpha, n)$;
                $x := x + 1$;
            $C_j := C_j \vee (-1)^{rand(2)} \cdot x$;   // rand: int in $[0, 2)$
    until $C_j$ is neither tautology nor simplifiable;
    $\Gamma := \Gamma \wedge C_j$;
return $\Gamma$;
```

main $i = \{1, \ldots, n\}$, as follows. We obtain n points the interval $\big[0, 1\big]$ (i.e., $1/n, 2/n, \ldots, 1$), and we define $P(X = i; n) = \phi(i/n)$. With the appropriated normalization, this results into:

$$P(X = i; n) = \frac{\phi(i/n)}{\sum_{j=1}^{n} \phi(j/n)} \tag{3.4}$$

Since $\lim_{n\to\infty} \sum_{j=1}^{n} \phi(j/n) \frac{1}{n} = \int_0^1 \phi(x)\, dx$, for big values of n we have:

$$P(X = i; n) = \frac{\phi(i/n)}{\sum_{j=1}^{n} \phi(j/n)} \approx_{n\to\infty} \frac{\phi(i/n)}{n \int_0^1 \phi(x)\, dx} = \frac{\phi(i/n)}{n} \tag{3.5}$$

Ansótegui et al. [2009b] generalize the uniform and regular models for any probability distribution. In this section, we focus on the case of power-law distributions $P(k) = c\, k^{-\alpha}$. Notice that this distribution generalizes the classical random model when $\alpha = 0$. On the other hand, as the value of α increases, some variables occur much more frequently than others. In the power-law model, we use the continuous probability distribution $\phi^{pow}(x; \alpha) = (1 - \alpha)\, x^{-\alpha}$. But this function is not defined in $x = 0$. Therefore, we use the interval $\big[0 + \epsilon, 1 + \epsilon\big]$, with a small value of ϵ. Equivalently, this is:

$$\phi^{pow}(x; \alpha) = \frac{1 - \alpha}{(1 + \epsilon)^{1-\alpha} - \epsilon^{1-\alpha}}\, (x + \epsilon)^{-\alpha} \tag{3.6}$$

Using ϕ^{pow} and normalizing, we obtain the following discrete power-law prob-

Algorithm 2: Regular Power-law k-CNF generator

```
   Input: int $n$, $m$, $k$; real $\alpha$;
   Output: $k$-CNF $\Gamma$, with $n$ variables and $m$ clauses
1  $bag := \emptyset$;
2  for $x = 1$ to $n$ do
3      $bag := bag \cup \{\lfloor P(X = x; \alpha, n) \frac{k\,m}{2} \rfloor$ copies of $x\}$;
4      $bag := bag \cup \{\lfloor P(X = x; \alpha, n) \frac{k\,m}{2} \rfloor$ copies of $\neg x\}$;
5  $S :=$ subset of $k\,m - |bag|$ random literals from $\{1, \ldots, n, \neg 1, \ldots, \neg n\}$
6      maximizing $P(X = x; \alpha, n) \frac{k\,m}{2} - \lfloor P(X = x; \alpha, n) \frac{k\,m}{2} \rfloor$;
7  $bag := bag \cup S$;
8  repeat
9      $\Gamma := \emptyset$;
10     for $j = 1$ to $m$ do
11         $C_j := \bigvee$(random multi-subset of $k$ literals from $bag$);
12         $bag := bag \setminus C_j$;
13         $\Gamma := \Gamma \wedge C_j$;
14 until $\Gamma$ contains neither tautologies nor simplifiable clauses;
15 return $\Gamma$;
```

ability distribution $P(X; \alpha, n)$:

$$P(X = i; \alpha, n) = \frac{(i + \epsilon \cdot n)^{-\alpha}}{\sum_{j=1}^{n} (j + \epsilon \cdot n)^{-\alpha}} \tag{3.7}$$

The uniform model is generalized as described in Algorithm 1. This generalization let us to generate random SAT instances whose number of variable occurrences follows a power-law distribution. The algorithm proceeds as follows. It iterates for generating the m clauses with k variables each (line 2). For choosing each of these variables, a real random number p between 0 and 1 is generated (line 7), and the variable x is selected if and only if $P(X = x; \alpha, n) \geq p$ and $P(X = x + 1; \alpha, n) < p$ (line 9), receiving a random polarity (line 12). Finally, a clause is accepted if and only if it is neither simplifiable (i.e., it has no repeated literals) nor tautology (no variable occurs in the clause with both polarities). Otherwise such clause is discarded, and a new clause is generated instead (line 13).

In the regular model, the number of occurrences of each variable is fixed *a priori*. In Algorithm 2, this model is generalized for power-law probability distributions. Therefore, we want all variables to occur with a frequency given by $P(X = x; \alpha, n)$. To do so, we construct a multi-set[4] *bag* with $k\,m$ literals. Notice that $k\,m$ is the total number of (repeated) literals in the resulting formula. This multi-set contains approximately $P(X = x; \alpha, n) \frac{k\,m}{2}$ copies of each literal. As this number is not probably an integer, we first add $\lfloor P(X = x; \alpha, n) \frac{k\,m}{2} \rfloor$

[4]A multi-set is a set with repeated elements.

copies of each literal (lines 2-4). Then, we decide which literals are added one more time, in order to assure that *bag* has a size equal to $k\,m$. To do so, it selects those literals maximizing the difference $P(X = x; \alpha, n)\frac{k\,m}{2} - \lfloor P(X = x; \alpha, n)\frac{k\,m}{2}\rfloor$ (line 5-7). Once this multi-set is created, it iterates to create the m clauses of the resulting formula (line 10). Each clause is generated choosing k random elements from *bag* (line 11), and they are removed from this multi-set (line 12). Finally, it is checked that none of the clauses are either simplifiable or tautologies (line 14). Otherwise all the m clauses are discarded, and m new clauses are generated instead. Notice that this is the only way to assure that the formula is selected uniformly among the set of formulas with m clauses of size k and n variables.

Part I

The underlying structure of SAT instances

Chapter 4

The community structure of SAT instances

4.1 Introduction

In this chapter, we introduce the notions of community structure and modularity in the context of SAT instances, in order to better characterize their underlying structure.

Many real-world networks have a clear *community structure* (or high *modularity*). This means that nodes can be grouped into communities such that most of the edges connect nodes of the same community. In contrast, Erdös-Rényi graphs do not have community structure, thus the modularity is very low.

We show that this feature is also very common in most of industrial SAT instances, whereas random SAT formulas do not have community structure at all. We analyze the impact of learning clauses during the search, and we observe that they slightly decrease the modularity, but they completely destroys the original partition of the formula. This analysis extends the study of the scale-free structure of SAT instances, previously presented, giving a more complete picture about their structure, with especial emphasis on industrial SAT benchmarks.

Questions addressed in this chapter:

Question 1. *What is the underlying structure of industrial SAT instances?*
Question 2. *How is this structure affected by CDCL SAT solving techniques?*

Related publications:

- Ansótegui, C., Giráldez-Cru, J., and Levy, J. (2012). The community structure of SAT formulas. In *Proceedings of the 15st International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 410–423.

It is worth noting that this analysis of the community structure has been already used to improve the performance of some solvers, including core SAT solvers, MaxSAT solvers or parallel SAT solvers [Ansótegui et al., 2015b; Martins et al., 2013; Neves et al., 2015; Sonobe et al., 2014].

The rest of the chapter is organized as follows. We formally introduce the definitions of community structure and modularity in the context of graphs in Section 4.2, and we present some algorithms to compute it. Then, we analyze the community structure of SAT instances in Section 4.3, and the effect of learning clauses on this structure in Section 4.4. Finally, we summarize some conclusion in Section 4.5.

In Section 9.2.3, we discuss the notion of overlapping communities, which may be useful for future works and applications related to the community structure of SAT instances. We also discuss in Section 9.2.2 the relation between the community structure and the two other structure features described in this thesis, and in Section 9.2.4 we mention other interesting structure features that may be analyzed to complement this study. Recall some related work in Section **??** about the underlying structure of real-world problems. Finally, remark that the applications presented in this dissertation (see Chapters 6, 7 and 8) directly uses the community structure of SAT instances.

4.2 The community structure of graphs

The notion of *modularity* was introduced by Newman and Girvan [2004]. This property is defined for a graph and a specific *partition* of its vertexes into *communities*, and it measures the adequacy of the partition in the sense that most of the edges are within a community and few of them connect vertexes of distinct communities. The modularity of a graph is then the maximal modularity for all possible partitions of its vertexes. Obviously, measured this way, the maximal modularity would be obtained putting all vertexes in a single community (i.e., all edges connect vertexes of the same community). To avoid this problem, modularity is defined as the fraction of edges connecting vertexes of the same community minus the *expected* fraction of edges for a random graph with the same number of vertexes and the same node degrees.

Definition 4.1 (Modularity of a graph)**.** Given a graph $G = (V, w)$ and a partition $P = \{P_1, \dots, P_n\}$ of its vertexes, we define its *modularity* as

$$Q(G,P) = \sum_{P_i \in P} \frac{\displaystyle\sum_{x,y \in P_i} w(x,y)}{\displaystyle\sum_{x,y \in V} w(x,y)} - \left(\frac{\displaystyle\sum_{x \in P_i} deg(x)}{\displaystyle\sum_{x \in V} deg(x)} \right)^2 \tag{4.1}$$

We call the first term of this formula the *inner edges fraction*, IEF for short, and the second term the *expected inner edges fraction*, IEF^e for short. Then, $Q = IEF - IEF^e$.

The *(optimal) modularity* of a graph is the maximal modularity, for any possible partition of its vertexes:

$$Q(G) = \max\{Q(G,P) \mid P\} \tag{4.2}$$

Since the IEF and the IEF^e of a graph are both in the range $[0,1]$, and, for the partition given by a single community, both have value 1, the optimal modularity of graph will be in the range $[0,1]$.

There has not been an agreement on the definition of modularity for bipartite graphs. Here we will use the notion proposed by Barber [2007] that extends Newman and Girvan's definition by restricting the random graph used in the computation of the IEF^e to be bipartite. In this definition, communities may contain vertexes of V_1 and of V_2.

Definition 4.2 (Modularity of a bipartite graph)**.** Given a bipartite graph $G=(V_1,V_2,w)$ and a partition $P=\{P_1,\dots,P_n\}$ of its vertexes, we define its *modularity* as

$$Q(G,P)=\sum_{P_i\in P}\frac{\displaystyle\sum_{\substack{x\in P_i\cap V_1\\ y\in P_i\cap V_2}} w(x,y)}{\displaystyle\sum_{\substack{x\in V_1\\ y\in V_2}} w(x,y)} - \frac{\displaystyle\sum_{x\in P_i\cap V_1} deg(x)}{\displaystyle\sum_{x\in V_1} deg(x)}\cdot\frac{\displaystyle\sum_{y\in P_i\cap V_2} deg(y)}{\displaystyle\sum_{y\in V_2} deg(y)} \tag{4.3}$$

There exist a wide variety of algorithms for computing the modularity of a graph. Moreover, there exist alternative notions and definitions of modularity for analyzing the community structure of a network. See [Fortunato, 2010] for a survey in the field. The decision version of modularity maximization is NP-complete [Brandes et al., 2008]. Therefore, most of the modularity-based algorithms proposed in the literature return an approximated lower bound for the modularity. They include greedy methods, methods based on simulated annealing, on spectral analysis of graphs, etc. Most of them have a complexity that make them inadequate to study the structure of very large graphs (as industrial SAT instances). There are algorithms specially designed to deal with large-scale networks, like the greedy algorithms for modularity optimization [Clauset et al., 2004; Newman and Girvan, 2004], the label propagation-based algorithm [Raghavan et al., 2007] and the method based on graph folding [Blondel et al., 2008].

The first algorithm for modularity maximization was described by Newman and Girvan [2004]. This algorithm starts by assigning every vertex to a distinct community. Then, it proceeds by joining the pair of communities that result in a bigger increase of the modularity value. The algorithm finishes when no community joining results in an increase of the modularity. In other words, it is a greedy gradient-guided optimization algorithm. The algorithm may also return a dendrogram of the successive partitions found. Obviously, the obtained partition may be a local maximum. Clauset et al. [2004] optimize the data structures used in this basic algorithm, using among others, data structures for sparse matrices. The complexity of this refined algorithm is $\mathcal{O}(m\,d\,\log n)$, where d is the depth of the dendrogram (i.e. the number of joining steps), m the number of edges and n the number of vertexes. They argue that d may be approximated by $\log n$, assuming that the dendrogram is a balanced tree, and the sizes of the communities are similar. However, this is not true for the graphs we have analyzed, where the sizes of the communities are not homogeneous.

Algorithm 3: Graph Folding Algorithm (GFA)

```
Input: Graph $G = (X, w)$
Output: Partition $P_1$
foreach $i \in X$ do $P_1[i] := i$;
$P_2 := OneLevel(G)$;
while $Q(G, P_1) < Q(G, P_2)$ do
    $P_1 := P_1 \circ P_2$;
    $G = Fold(G, P_2)$;
    $P_2 := OneLevel(G)$;
function $OneLevel$(Graph $G$) : Partition $P$
    foreach $i \in X$ do $P[i] := i$;
    repeat
        $changes := false$;
        foreach $i \in X$ do
            $bestinc := 0$;
            foreach $c \in \{c \mid \exists j.w(i,j) \neq 0 \wedge P[j] = c\}$ do
                $inc := \sum_{P[j]=c} w(i,j) - deg(i) \cdot \sum_{P[j]=c} deg(j) / \sum_{j \in X} deg(j)$;
                if $inc > bestinc$ then
                    $P[i] := c$; $bestinc := inc$; $changes := true$;
    until $\neg changes$;
    return $L$
function $Fold$(Graph $G_1$, Partition $P$) : Graph $G_2$
    $X_2 = \{c \mid \exists i \in X_1.P[i] = c\}$;
    $w_2(c_1, c_2) = \sum_{i,j \in X_1 \mid P[i]=c_1 \wedge P[j]=c_2} w_1(i,j)$;
    return $G_2 = (X_2, w_2)$;
```

This algorithm has not been able to finish, for none of our SAT instances, with a runtime limit of one hour.

An alternative algorithm is the *Label Propagation Algorithm* (*LPA*) proposed by Raghavan et al. [2007]. Initially, all vertexes are assigned to a distinct label (e.g., its identifier). Then, the algorithm proceeds by re-assigning to every vertex the most frequent label among its neighbors. The procedure ends when every vertex is assigned a label that is maximal among its neighbors. In case of a tie between most frequent labels, the assigned label is chosen randomly. The algorithm returns the partition defined by the vertexes sharing the same label. The Label Propagation algorithm has a near linear complexity. However, it has been shown experimentally that the partitions it computes have a worse modularity than the partitions computed by the Newman's greedy algorithm.

The *Graph Folding Algorithm* (*GFA*) (a.k.a. *Louvain Method*) proposed by Blondel et al. [2008] (see Alg. 3) improves the Label Propagation algorithm in two directions. The idea of moving one node from one community to another following a greedy strategy is the same, but, instead of selecting the most frequent community among the neighbors of a node, it selects the community which would most increase the modularity. The increase of the modularity ΔQ produced by the node $i \in V$ of a graph $G = (V, w)$ changing its community to $P_c \in P$ can be measured as expressed in Equation 4.4:

$$\Delta Q(i, P_c) = \sum_{j \in P_c} w(i,j) - \frac{deg(i) \cdot \sum\limits_{j \in P_c} deg(j)}{\sum\limits_{j \in V} deg(j)} \tag{4.4}$$

which can be directly derived from Equation 4.1. This greedy strategy corresponds to the function *OneLevel* (lines 7-18), and such increase of the modularity is computed in line 14. A second improvement is performed once no movement of node from community to community can increase the modularity (i.e., we have reached a local modularity maximum). In this step, nodes of the same community are merged. For this purpose, we construct a new graph. The nodes of this new graph are the communities of the previous graph, and its edges have a weight equal to the sum of the weights of the edges of the previous graph that connect nodes of both communities. Notice that edges in this new graph connect two communities of the previous graph. Notice also that this new graph may contain self-loops[1]. This folding step corresponds to the function *Fold* (lines 19-22). Then, we apply again the greedy function to the new graph, and its communities are merged again. These two steps are repeated till no modularity increase is possible (lines 3-6).

4.3 Experimental analysis of the community structure

In this section, we present the analysis of the community structure of some SAT formulas. We use the industrial instances of the SAT Competition 2013 (excluding the 3 instances of the family *software-bmc*, due to their large size), and some families of randomly generated problems.

In order to analyze the community structure of a SAT instance, we represent it using the VIG and CVIG graph models. Then, we run GFA (see Alg. 3) on these graphs. Recall that this algorithm returns a partition P into communities of the nodes of the graph. Therefore, the value of the modularity Q associated to such partition is computed according to Equation 4.1, by fixing the parameter P. Notice that GFA is a greedy algorithm, hence the optimal modularity is not guaranteed. Instead, we obtain a lower bound of this optimal value. Nevertheless, a high value of Q returned by GFA is enough to ensure a clear community structure[2]. Moreover, as we have two graph models, we can use the highest

[1]A self-loop is an edge that connects a node to itself.

[2]The modularity Q ranges in the interval $[0, 1]$.

modularity as a lower bound of the modularity of the instance. Hereafter, we refer as *modularity* to the lower bound returned by GFA.

First, we compute the community structure of each industrial SAT instance. We observe that all instances of the same family have a similar community structure (measured with the modularity Q, the number of communities $|P|$, etc.). For instance, the maximal dispersion of the modularity Q is found in the family *hardawre-velev* in the VIG model, with an standard deviation $SD[Q] = 0.0081$. This means that the modularity Q of the 21 *hardware-velev* instances is very similar. Therefore, we group results by families, and we show all results in average for each family.

In Table 4.1, we show the analysis of the community structure of industrial SAT instances, grouped by families. In particular, we report the modularity Q_o of the original instances for each industrial family and each graph model (VIG and CVIG), and the modularity Q_p after preprocessing the instances. We use the preprocessor Satelite [Eén and Biere, 2005], without expensive simplifications (see its documentation for further details). We remark in bold the modularity values greater than 0.3, as a representative value of a clear community structure. We also report the number of communities $|P|$ and the percentage of nodes l belonging to the largest community. Finally, we also study the connected components, as suggested by Biere and Sinz [2006]. In this case, $|P|$ and l stand for the number of connected components and the percentage of nodes in the largest component, respectively.

We observe that *all* families show a very clear community structure, with values of Q_o greater than 0.8 in many cases. We remark again that the GFA algorithm returns a lower bound on the modularity. In other kind of networks, values greater than 0.7 are rare, therefore the values obtained for SAT instances can be considered as exceptionally high. Moreover, GFA gives better bounds in both models (VIG and CVIG) than other algorithms. See [Ansótegui et al., 2012] for a comparative between GFA and LPA on industrial SAT benchmarks.

If we compare the modularity for the VIG model Q_o^{VIG} with the same values for the CVIG model Q_o^{CVIG}, we can conclude that, in general, these values are higher for the VIG model. As GFA is not able to run on bipartite graphs, we have re-implemented this algorithm to be used in this kind of graphs, modifying the function *Fold* in order to preserve the bipartite structure in the folding step. However, even with this modification, GFA is not able to find a good partition (and therefore, a good modularity) on bipartite graphs. After the first *folding*, GFA is not (almost) able to do any change in the bipartite structure of the resulting graph, and it finishes. Therefore, as GFA does not group many communities and it finishes before for the CVIG, the number of communities $|P|^{CVIG}$ is bigger than $|P|^{VIG}$, and the biggest community l^{CVIG} is smaller than the one obtained for the VIG model (l^{VIG}). Hence, the modularity is also smaller: $Q_o^{CVIG} < Q_o^{VIG}$.

We also compare the values of the modularity before and after prepossessing the instances (Q_o and Q_p, respectively). We see that in most cases, Q_p is slightly smaller than Q_o, and in some *crypto* families, it is even bigger. However, both

		VIG				CVIG				CC	
Family	#inst.	Q_o	Q_p	$\|P\|$	l	Q_o	Q_p	$\|P\|$	l	$\|P\|$	l
2d-strip-packing	5	**0.94**	**0.94**	40.2	4.8	**0.93**	**0.93**	$>10^3$	3.4	1	100
bio	5	**0.61**	**0.55**	42.4	7.9	**0.37**	**0.36**	$>10^3$	0.2	1.4	99.9
crypto-aes	11	**0.80**	**0.75**	23.3	12.7	**0.61**	**0.56**	$>10^3$	4.1	1	100
crypto-des	9	**0.95**	**0.93**	82.4	2.9	**0.50**	**0.43**	$>10^4$	0.0	1	100
crypto-gos	30	**0.64**	**0.64**	39.6	16.3	**0.63**	**0.62**	$>10^2$	10.5	1	100
crypto-md5	11	**0.78**	**0.78**	33.1	6.1	**0.51**	**0.54**	$>10^4$	0.0	1	100
crypto-sha	30	**0.56**	**0.64**	13.7	11.6	**0.56**	**0.58**	$>10^3$	0.2	1	100
crypto-vmpc	8	0.24	0.24	9.5	16.0	**0.40**	**0.40**	$>10^3$	0.3	1	100
diagnosis	26	**0.93**	**0.93**	56.8	4.5	**0.48**	**0.44**	$>10^5$	0.0	1	100
hardware-bmc-ibm	4	**0.97**	**0.96**	76.0	2.5	**0.50**	**0.47**	$>10^5$	0.0	1	100
hardware-bmc	3	**0.92**	**0.89**	20.7	7.7	**0.50**	**0.43**	$>10^4$	0.1	1	100
hardware-cec	30	**0.86**	**0.79**	29.2	14.9	**0.48**	**0.46**	$>10^4$	1.1	1.1	99.9
hardware-velev	21	**0.68**	**0.68**	16.4	36.3	**0.49**	**0.49**	$>10^5$	2.9	1	100
planning	25	**0.87**	**0.85**	22.6	9.9	**0.50**	**0.50**	$>10^5$	0.0	1	100
scheduling-pesp	30	**0.78**	**0.78**	14.7	17.0	**0.36**	**0.36**	$>10^4$	0.0	2.4	95.3
scheduling	30	**0.89**	**0.89**	45.7	6.1	**0.47**	**0.46**	$>10^5$	0.0	1	100
software-bit-verif	12	**0.88**	**0.80**	21.0	9.9	**0.51**	**0.57**	$>10^4$	2.5	1	100
termination	5	**0.78**	**0.70**	38.4	14.0	**0.53**	**0.53**	$>10^4$	1.0	1	100

Table 4.1: Modularity of industrial SAT instances, computed with GFA, before and after prepossessing (Q_o and Q_p, respectively). $|P|$ stands for number of communities (or connected components), and l for fraction of nodes in the largest community (or component). Results are reported in average for each industrial family.

values are very close. Therefore, we can conclude that the default prepossessing techniques applied by Satelite almost do not affect the community structure of the formula.

If all communities have a similar size, then $l \approx 1/|P|$. In many cases in Table 4.1, we have $|P| \gg 1/l$. This means that the community structure has a big variability in the sizes of the communities obtained.

Finally, we have also studied the *connected components* of these instances (after prepossessing). Almost all instances have a single connected component with almost all variables included in it. Hence, the rest of connected components contain just an insignificant subset of variables. Therefore, the modularity gives us much more information about the structure of the formula than connected components. Notice that a connected component can be structured into several communities. We also found a large number of very small connected components in some industrial formulas before preprocessing (not shown in Table 4.1). However, these components have very few variables, and they are easily removed by unit propagation in the preprocessing step.

We have also conducted a study of the modularity of some families of random 3-CNF SAT instances, varying the clause/variable ratio m/n. Each family is

n	m/n	Q_o	$\lvert P\rvert$	l
10^4	1	**0.486**	545	3.8
10^4	1.5	**0.353**	146	5.1
10^4	2	0.280	53	6.8
10^4	3	0.217	14	15.5
10^4	4	0.178	11	14.8
10^4	4.25	0.170	11	14.6
10^4	4.5	0.163	11	14.7
10^4	5	0.152	11	14.3
10^4	6	0.133	12	13.9
10^4	7	0.120	10	15.0
10^4	8	0.138	6	25.0
10^4	9	0.130	6	24.3
10^4	10	0.123	6	24.4

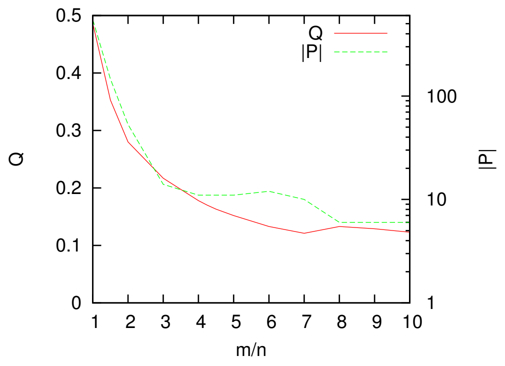


Table 4.2: Modularity of random 3-CNF formulas varying the clause/variable ratio m/n, for families of 100 instances and $n = 10^4$ variables.

m/n	n	Q_o	$\lvert P\rvert$	l
4.25	10^2	0.177	6.0	14.5
4.25	10^3	0.187	10.5	11.4
4.25	10^4	0.170	11.0	12.2
4.25	10^5	0.151	14.0	6.8
4.25	10^6	0.151	14.0	5.7

Table 4.3: Modularity of random 3-CNF formulas at the peak transition region (clause/variable ratio m/n=4.25), varying the number of variables n.

composed by 100 SAT instances, and a fixed number of variables $n = 10^4$. In this experiment we run the GFA algorithm on the VIG model only. Table 4.2 shows the results. As we can see, the modularity of random instances is only significant for very low clause/variable ratios, i.e., on the leftist SAT easy side. This is due to the presence of a large quantity of very small communities. Notice, that as m/n increases, variables get more connected but without following any particular structure, and the number of communities highly decreases. Moreover, even for low values of m/n, the modularity is not as high as for industrial instances, confirming their distinct nature. We do not observe any abrupt change in the phase transition point.

In a second experiment with random formulas, we want to investigate the modularity at the peak transition region for an increasing number of variables. Table 4.3 shows these results. As we can see, the modularity is very low and it tends to slightly decrease as the number of variables increases, and seems to tend to a particular value (0.15 for the phase transition point).

	VIG				
Family	Q_o	Q_p	Q_{10^3}	Q_{10^4}	Q_{10^5}
2d-strip-packing	**0.942**	**0.942**	**0.942**	**0.932**	**0.884**
bio	**0.607**	**0.549**	**0.621**	**0.619**	**0.590**
crypto-aes	**0.804**	**0.752**	**0.777**	**0.737**	**0.627**
crypto-des	**0.952**	**0.929**	**0.945**	**0.929**	**0.717**
crypto-gos	**0.639**	**0.641**	**0.621**	**0.522**	**0.424**
crypto-md5	**0.784**	**0.780**	**0.850**	**0.847**	**0.825**
crypto-sha	**0.558**	**0.641**	**0.644**	**0.641**	**0.577**
diagnosis	**0.932**	**0.927**	**0.932**	**0.926**	**0.871**
hardware-bmc	**0.922**	**0.956**	**0.923**	**0.920**	**0.835**
hardware-bmc-ibm	**0.971**	**0.886**	**0.970**	**0.969**	**0.962**
hardware-cec	**0.857**	**0.785**	**0.853**	**0.825**	**0.765**
hardware-velev	**0.679**	**0.678**	**0.678**	**0.677**	**0.676**
planning	**0.865**	**0.850**	**0.856**	**0.853**	**0.834**
scheduling	**0.894**	**0.781**	**0.896**	**0.885**	**0.817**
scheduling-pesp	**0.780**	**0.892**	**0.780**	**0.772**	**0.662**
software-bit-verif	**0.878**	**0.801**	**0.872**	**0.845**	**0.728**
termination	**0.775**	**0.695**	**0.764**	**0.674**	**0.619**

	CVIG				
Family	Q_o	Q_p	Q_{10^3}	Q_{10^4}	Q_{10^5}
crypto-vmpc	**0.398**	**0.398**	**0.397**	**0.397**	0.241

Table 4.4: Modularity Q_X of industrial SAT families after X conflicts, for VIG and CVIG models.

4.4 The community structure during SAT solver search

In this section, we want to investigate how CDCL techniques affect the community structure of the formula. The natural question is: even if the original formula shows a clear community structure, could it be the case that this structure is quickly destroyed during the search process? Moreover, learning techniques introduce new learnt clauses to the original formula. Therefore, a second natural question is: how these new clauses affect to the community structure of the formula? Finally, even if the value of the modularity is not altered, can it be the case that the partition of the formula into communities is changed?

To answer these questions, we analyze the community structure of the formulas the solver is seeing after learning clauses. In particular, we use the instances that the solver MiniSAT is seeing after after 10^3, 10^4 and 10^5 conflicts, and we compare these results to the community structure of original formulas.

In Table 4.4, we show the values of the modularities Q_o and Q_p of the original and preprocessed formulas, and the modularities Q_X of the formulas after $X =$

	VIG			
Family	Q_p	$Q^{part}_{10^3}$	$Q^{part}_{10^4}$	$Q^{part}_{10^5}$
2d-strip-packing	**0.942**	0.272	0.209	0.132
bio	**0.549**	0.026	0.028	0.029
crypto-aes	**0.752**	**0.346**	**0.324**	0.250
crypto-des	**0.929**	**0.361**	**0.351**	0.245
crypto-gos	**0.641**	0.122	0.097	0.059
crypto-md5	**0.780**	0.277	0.272	0.250
crypto-sha	**0.641**	0.121	0.122	0.107
crypto-vmpc	0.239	0.076	0.057	0.046
diagnosis	**0.927**	**0.308**	**0.327**	**0.306**
hardware-bmc	**0.886**	**0.715**	**0.702**	**0.632**
hardware-bmc-ibm	**0.956**	**0.661**	**0.635**	**0.630**
hardware-cec	**0.785**	**0.469**	**0.440**	**0.407**
hardware-velev	**0.678**	**0.328**	**0.326**	**0.319**
planning	**0.850**	**0.535**	**0.534**	**0.423**
scheduling	**0.892**	**0.758**	**0.746**	**0.665**
scheduling-pesp	**0.781**	**0.755**	**0.748**	**0.626**
software-bit-verif	**0.801**	**0.569**	**0.547**	**0.449**
termination	**0.695**	**0.428**	**0.372**	**0.313**

Table 4.5: Modularity Q^{part}_X of the formulas after X conflicts (for VIG), and using the partition of the original formula.

$10^3, 10^4, 10^5$ conflicts. We only report results for the VIG model, except for the family *crypto-vmpc*, for which the obtained modularity is greater when using the CVIG. We can observe that the modularity weakly decreases as we add learnt clauses, but it is still meaningful. Therefore, learning does not completely destroy the organization of the formula into weakly connected communities.

The question now is, even if the modularity does not decreases very much, could it be the case that the communities have changed? In other words, can it be the case that there is still a clear community structure but the partition of the formula into communities has totally changed? Remark that in the previous experiment, the GFA algorithm is re-run for each instance after learning. Therefore, it can be the case that the partition returned by this algorithm is completely different.

If a considerable part of learning is performed locally inside each community, then the communities will not change. In the VIG model, the set of nodes is always the same (even with the addition of learnt clauses). Notice that in this model, nodes represent only variables, so no learnt clause creates new nodes. However, these learnt clauses do create new edges between the existent nodes, or modify the weight of existing edges. Therefore, we can use modularity as a *quality measure* to see how *internal* a learnt clause is. Notice that modularity is a function of two parameters: a graph, and a partition of it. For a given

n	m/n	Q_o	Q_p	Q_l
300	1	**0.459**	0	**0.453**
300	2	0.291	0.235	0.291
300	4	0.190	0.188	0.073
300	4.25	0.183	0.182	0.041
300	4.5	0.177	0.177	0.045
300	6	0.150	0.150	0.120
300	10	0.112	0.112	0.171

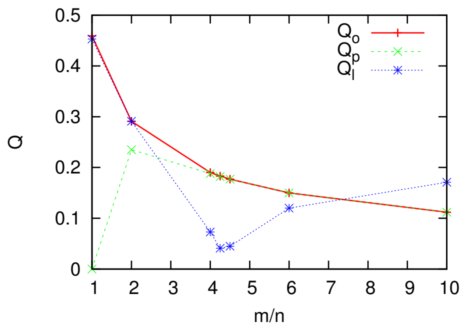


Table 4.6: Modularity of random 3-CNF formulas varying the clause/variable ratio m/n, for original formulas (Q_o), preprocessed formulas (Q_p), and formulas after adding all learnt clauses needed to solve them (Q_l).

partition of a graph, a new edge will increase the modularity iff it connects two nodes of the same community, otherwise modularity will decrease. Thus, using the partition of the original formulas, we can see if learning acts *internally* (i.e., connecting variables of the same community), or if it tends to connect variables of different communities.

We conduct another experiment to see how learning changes such partition. We use the same formulas than before (after 10^3, 10^4 and 10^5 conflicts), and the partition of the VIG obtained from the original formulas, to compute the modularity Q^{part}. Notice that in the case we do not run the GFA to compute a (possibly) new partition, but we give explicitly that partition. Moreover, we can only use the VIG since the set of nodes is the same in both original formulas and formulas after learning (recall that using the CVIG, each new learnt clause adds a new clause-node to the graph).

In Table 4.5, we show the result of the modularity Q^{part}. The analysis of the tables shows us that there is a drop-off in the modularity as we incorporate more learnt clauses. In other words, the partition of the formula is changing. This means that, if we use explicitly the community structure to improve the efficiency of a SAT solver, to overcome this problem, we would have to recompute the partition (after some number of conflicts) to adjust it to the modified formula.

Finally, we wanted to evaluate the impact on modularity of the prepossessing, and the effect of adding all the learnt clauses in random formulas. Table 4.6 shows the results. We compare the modularity Q_o of the original formulas to the modularity Q_p of the preprocessed formulas, and the modularity Q_l of the formulas after adding all learnt clauses needed to solve the formula. In general, prepossessing has almost no impact on the modularity of the formula ($Q_o \approx Q_p$), except for small values of the clause/variable ratio m/n. For instance, for $m/n = 1$ there is an abrupt change on Q because prepossessing already solves the formula. With respect to the addition of learnt clauses, it is interesting to observe that closer to the peak transition region $m/n = 4.25$, lower the modularity is.

A possible explanation is that at the peak region we find the hardest instances, and harder an instance is, more clauses connecting distinct communities have to be learnt, thus lower the modularity is.

4.5 Conclusions

We show that most industrial SAT instances exhibit a clear community structure. This means that we can find a partition of the formula into communities in which variables are highly interconnected. In general, industrial formulas have a modularity of around 0.8. Notice that in other kind of networks, values greater than 0.7 are rare. On the contrary, random formulas do not have this kind of structure (as expected). Moreover, we check that modularity slightly decreases with the addition of learnt clauses, but it is still high. Thus, a partition that shows a good community structure can be found. However, we also observe that such partition change during the search.

Chapter 5

The self-similar structure of SAT instances

5.1 Introduction

In this chapter, we introduce the notion of fractal dimension or self-similar structure, and we analyze this structure in the context of SAT instances.

The fractal dimension is another very common feature in many real-world networks. This means that the *shape* of the network is the same at different scales (i.e., grouping sets of nodes into a single one).

In the case of industrial SAT problems, we show that this property is also shared by the majority of the instances, while random formulas are not self-similar at all. Also, we analyze the effects of learning clauses on the self-similar structure, and we observe that these clauses very slowly increase the fractal dimension. This means that, in general, learning does not contribute to connect distant parts of the formula. This study complements the other structure features described before (i.e., the scale-free structure and the community structure).

Questions addressed in this chapter:

Question 1. *What is the underlying structure of industrial SAT instances?*
Question 2. *How is this structure affected by CDCL SAT solving techniques?*

Related publications:

- Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2014). The fractal dimension of SAT formulas. In *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14)*, pages 107–121.

The rest of the chapter is organized as follows. We formally introduce the definitions of fractal dimension and self-similarity in the context of graphs in

Section 5.2, including an analysis about the relations between it and the diameter (see Subsection 5.2.1). Then, we analyze the self-similar structure of SAT instances in Section 5.3, and the effect of learning clauses on this structure in Section 5.4. Finally, we summarize some conclusion in Section 5.5.

In Section 9.2.2, we discuss the relation between the self-similar structure and the other structure features described before. Also, in Section 9.2.4, we discuss other interesting structure features that may be analyzed to complement this analysis. Recall some related work in Section **??** about the underlying structure of real-world problems. Finally, remark that the application presented in Chapter 8 directly uses the self-similar structure (as well as the scale-free structure and the community structure).

5.2 The self-similar structure of graphs

We can define a notion of fractal dimension of a graph following the principle of self-similarity. We will use the definition of box covering by Hausdorff [Mandelbrot, 1983]. This definition is for unweighted graphs (i.e., edges have no weight). A weighted graph is a generalization of an unweighted one (see Chapter 2). Therefore, we can just simply use (weighted) graphs without considering the weights of their edges.

Definition 5.1 (Diameter of a graph)**.** The **distance** between two nodes is the minimum number of edges we need to follow to go from one node to the other.

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

Definition 5.2 (Fractal dimension of a graph)**.** Given a graph G, a **box** B of size l is a subset of nodes such that the distance between any pair of them is strictly smaller than l.

We say that a set of boxes covers a graph if every node of the graph is in some box. Let $N(l)$ be the minimum number of boxes of size l required to *cover* the graph.

We say that a graph has the **self-similarity** property if the function $N(l)$ decreases polynomially, i.e. $N(l) \sim l^{-d}$, for some value d. In this case, we call d the **fractal dimension** of the graph.

Notice that $N(1)$ is equal to the number of nodes of G, and $N(d^{max}+1)$ is the number of connected components of the graph.

Lemma 5.1. Computing the function $N(l)$ is NP-hard[1].

Proof. We prove that computing $N(2)$ is already NP-hard by reducing the graph coloring problem to the computation of $N(2)$. 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

[1]Song et al. [2007] state the same result, but they prove the wrong reduction. They reduce the computation of $N(2)$ to the graph coloring problem.

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. $N(2)$. □

There are several efficient algorithms that approximate $N(l)$. They compute upper bounds of $N(l)$. They are called *burning* algorithms (see [Song et al., 2007]). 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 5.3 (Circle)**.** A **circle** of *radius* r and *center* c is a subset of nodes of a graph G such that the distance between any of them and the node c is strictly smaller that r.

Let $N(r)$ be the minimum number of circles of radius r required to cover a graph.

Notice that any circle of radius r is inside of a box of size $2\,r-1$ (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 with fractal dimension d. We can approximate this fractal dimension using the *Maximum-Excluded-Mass-Burning* (*MEMB*) algorithm [Song et al., 2007], 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 nodes covered by this circle. This is repeated till all nodes are burnt. The function $N(r)$ is computed for all values of $1 < r < d^{max}$, till for some r' we obtain $N(r') = k$, where k is the number of connected components of G.

The MEMB algorithm is still too costly for our purposes in very large networks (as SAT instances). We apply the following strategy to make the algorithm more efficient. We order the nodes according to their degree: $\langle v_1, \ldots, v_n \rangle$ such that $degree(v_i) \geq degree(v_j)$, when $i < j$. Now, for $i = 1$ to n, if c_i is not burned, then select the circle of center c_i and radius r (even if it does not maximizes the number of unburned covered nodes), and burn all unburned nodes belonging to this circle. We call this algorithm *Burning by Node Degree (BND)*, and describe it in Alg. 4. We will compare the accuracy and efficiency of algorithms MEMB and BND in Section 5.3 to justify the use of algorithm BND in our experimentation.

In our study, we analyze the function $N(r)$ for the graphs obtained from a SAT instance following unweighted version of the VIG and CVIG models. We

Algorithm 4: Burning by Node Degree (**BND**)

```
Input: Graph $G = (V, E)$
Output: vector[int] $N$
$N[1] := |V|$;
int $i := 2$;
while $N[i-1] > connectedComponents(G)$ do
    vector[bool] $burned(|V|)$;
    $N[i] := 0$;
    $burned := \{\textbf{false}, \ldots, \textbf{false}\}$;
    while existsUnburnedNode(burned) do
        $c := highestDegreeUnburnedNode(G, burned)$;
        $S := circle(c, i)$; // circle with center $c$ and radius $i$;
        foreach $x \in S$ do
            $burned[x] :=$ true;
        $N[i]++$;
    i := i+1;
```

denote these two functions as $N(r)$ and $N^b(r)$, respectively, and they relate to each other as follows.

Lemma 5.2. If $N(r) \sim r^{-d}$ then $N^b(r) \sim r^{-d}$.
If $N(r) \sim e^{-\beta\, r}$ then $N^b(r) \sim e^{-\frac{\beta}{2}\, r}$.

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

Conversely, given a circle of radius $2\,r$ 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 $N(r+1) \leq N^b(2\,r)$.

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

Previous lemma states that if a SAT formula is (fully) self-similar, then in both models, VIG and CVIG, the fractal dimension is the same. In such case, if we plot $N(r)$ as a function of r in double-logarithmic axes, we obtain a line with slope $-d$. If $N(r)$ decays exponentially, then the decay factor in the CVIG model is half of the decay factor in the VIG model. In such case, if we plot $N(r)$ in semi-logarithmic axes, we obtain a line with slope $-\beta$. We will always plot $N(r)$ in double-logarithmic axes. Thus, when $N(r)$ decays exponentially, we will observe a concave curve.

5.2.1 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 $r^{max} \leq d^{max} \leq 2\,r^{max}$. From these relations we can conclude the following.

Lemma 5.3. For self-similar graphs, or SAT formulas, (i.e., $N(r) \sim r^{-d}$), the diameter is $d^{max} \approx n^{1/d}$, where d is the fractal dimension.
In graphs, or SAT formulas, 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*[2] L of a graph, have been widely used in the characterization of graphs. For instance, *small world graphs* [Walsh, 1999] are characterized as those graphs with a small typical distance $L \sim \log n$ and a large clustering coefficient. This definition works well for *families* of graphs because then we can quantify the typical distance as a function on the number of nodes. But it is quite imprecise in the case of individual graphs, because it is difficult to decide what is a "small" distance and a "large" clustering coefficient, for a concrete graph. Moreover, the diameter and the typical distance of a graph are measures quite expensive to compute in practice (for huge graphs, as the ones representing many industrial SAT formulas), even though there is a quadratic algorithm. In fact, our approximation to the fractal dimension can be computed more efficiently than the diameter.

Since we are interested in characterizing the structure of formulas, the fractal dimension is a better measure because it is independent of the size. Thus, formulas of the same family (and similar structure), but very distinct size, will have similar dimension and $N(r)$ function shape.

5.3 Experimental analysis of the self-similar structure

In this section, we present an exhaustive analysis of the 300 industrial SAT instances of the SAT Competition 2013, and 90 random 3-CNF formulas of $n = 10^5$ variables at different clause/variable ratios. We will see that most industrial

[2]The typical distance of a graph is the average of the distances between any two nodes.

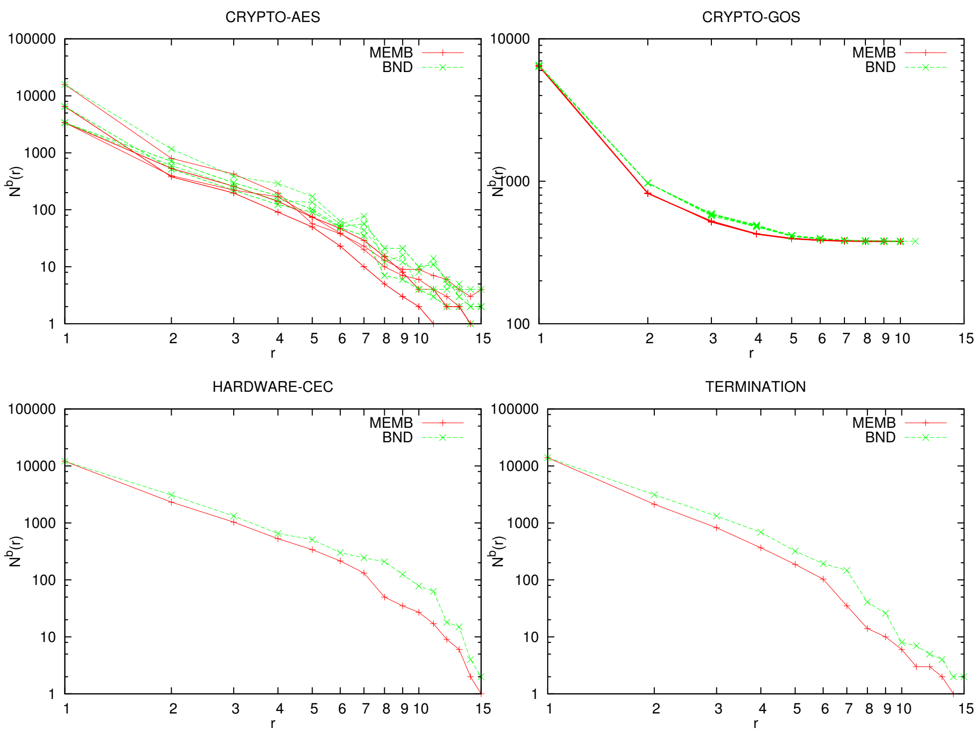


Figure 5.1: Upper bounds for $N^b(r)$ obtained with MEMB and BND algorithms, for the 17 industrial instances that MEMB is able to compute in 30 minutes, grouped by families.

instances 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. $N(r) \sim e^{-\beta\, r}$.

Before presenting the results of this evaluation, let us justify the use of the BND algorithm to calculate the fractal dimension, instead of the MEMB algorithm [Song et al., 2007]. We compare both algorithms in order to evaluate how accurate BND is.

We run both algorithms for the set of 300 industrial instances of the SAT Competition 2013 with a timeout of 30 minutes. While the BND algorithm finishes for all the 300 instances, MEMB is only able to approximate $N^b(r)$ in 17 instances. Moreover, while the average runtime of BND for these instances is 0.11 seconds, MEMB takes an average of 10 minutes and 7.2 seconds to compute them. On the other hand, the approximations of $N^b(r)$ computed by MEMB and BND are very similar. In Figure 5.1, we represent these upper bounds for these 17 industrial SAT instances.

Since the MEMB algorithm is more accurate than the BND algorithm, the upper bounds of $N^b(r)$ that MEMB calculates are below the ones calculated by BND. The real values of $N^b(r)$ are probably even lower in the final points (where

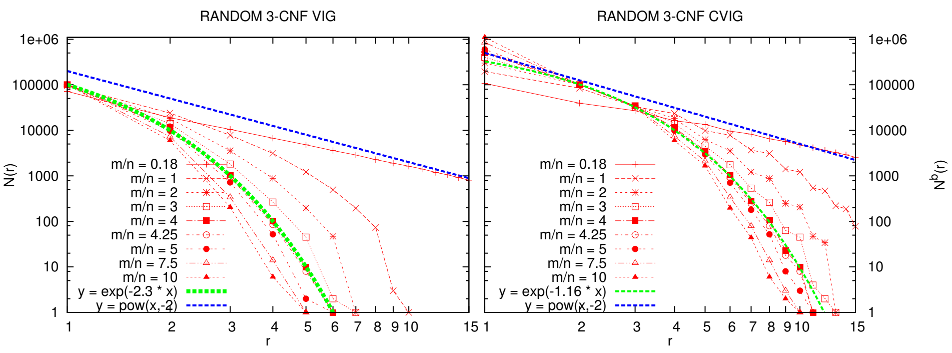


Figure 5.2: Functions $N(r)$ for VIG (left), and $N^b(r)$ for CVIG (right), for 3-CNF random formulas with distinct values of m/n. Formulas are generated using $n = 10^5$ variables and taking the major connected component, except for $m/n = 0.18$, where $n = 10^6$.

the approximation is less accurate).

Random 2-SAT formulas in the VIG model correspond exactly to Erdös-Rényi graphs. It is known that these formulas have a phase transition point at $m/n = 1$ where formulas pass from satisfiable to unsatisfiable with probability one. It is also known that at $m/n = 0.5$ there is a percolation threshold. Formulas below this point have many connected components in the VIG graph, and above this threshold there is a major connected component. In the percolation point the formula is self-similar with a fractal dimension $d = 2$. Above this point $N(r)$ decays exponentially. To the best of our knowledge, a result of this kind is not known for random 3-CNF formulas.

In Figure 5.2 we plot the functions $N(r)$ and $N^b(r)$ for random 3-CNF formulas at distinct clause/variable ratios. Experimentally, we observe that the functions only depends on the clause/variable ratio m/n, and not on the number of variables (this is not shown in figures). In the phase transition point $m/n = 4.25$, the function $N(r)$ has the form $N(r) \sim e^{-2.3\,r}$, i.e. it decays exponentially with $\beta = 2.3$. Hence, $r^{max} = \frac{\log n}{2.3} + 1$. For instance, for $n = 10^5$ variables, random formulas have a radius $r^{max} \approx 6$. For bigger values of m/n, the decay factor β is bigger. In the CVIG model, we observe the same behavior. However, in this case, in the phase transition point, the function $N^b(r)$ decays exponentially with exponent $\beta = 1.16 \approx 2.3/2$. Hence, the decay is just half of the decay of the VIG model, as we expected by Lemma 5.2. Also, we have experimentally found a percolation threshold at $m/n \approx 0.17$. At this point the principal connected component also exhibits a fractal dimension $d = 2$.

In Figure 5.3, we plot the function $N^b(r)$ for some industrial formulas grouped by families. We observe that most of them are self-similar, and most dimensions range between 2 and 4. In many of the industrial families, all instances have the same fractal dimension, being this dimension a characteristic of the family. See, for instance, families *crypto-sha* or *diagnosis*. Notice that the size of the formulas

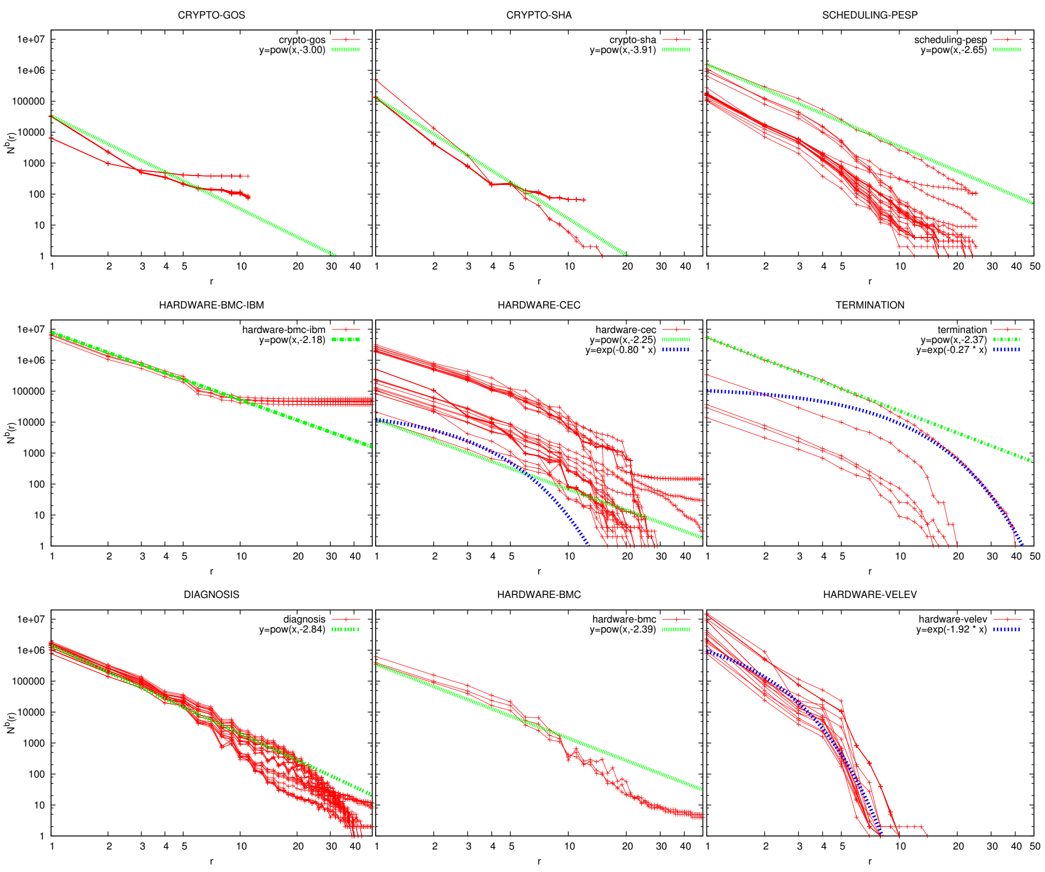


Figure 5.3: Function $N^b(r)$ for some industrial SAT formulas, grouped by families.

does not affect the value of the dimension (in the logarithmic representation the function can be higher or lower, but with the same slope).

In general, the polynomial decay is clearer for small values of r. Moreover, in this area, the slope is very similar for all instances of the same family of formulas.

For big values of r, we must make some considerations. First, the upper bound on $N^b(r)$ that we calculate can be a bad approximation. Second, there are two phenomena that we can identify. In some cases there is an abrupt decay, but the whole function can not be approximated by an exponential function (see some *hardware-cec* or *termination* instances, for example). This decay in the number of required tiles can be due to a small number of edges connecting distant areas of the graph. These edges have no effect for small values of r, but may drop down the number of tiles for big values of r. In some other cases (see *hardware-bmc-ibm*, for instance), there is a long tail. In this case, it is due to the existence of (small) unconnected components in the graph. If we compute

$N(r)$ only for the major component, this tail disappears.[3]

Finally, all instances of the *hardware-velev* family have a $N(r)$ function with exponential decay, i.e. are not self-similar.

5.3.1 Fractal dimension at fine-grained scale

If a graph is self-similar, then it has the same structure at all scales. We could replace groups of nodes tiled by a box by a single node, obtaining another graph with the same structure. In our experiments, we observe that this is the case for small values of r (for small values of r, the function behavior is $N(r) \approx C\, r^{-d}$). However, this is more arguable for big values of r. Perhaps this is because the graph is not self-similar at large scale (coarse-grained), or because our approximation of $N(r)$ is not precise enough.

We think that, more than whether there exists a self-similar structure, what is important, is the value of the fractal dimension at fine-grained, i.e. the slope of the function $N(r)$ for small values of r. Therefore, in our experimentation, we note these fine-grained dimensions as d and d^b 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 $N(r)$ and $N^b(r)$, for $r = 1, \ldots, 6$.

5.4 The self-similar structure during SAT solver search

In this section, we analyze the effects of CDCL techniques on the self-similar structure of SAT instances. In particular, we focus our analysis on the learnt clauses. Notice that the effects produced by a learnt clauses depends on its length. While unitary clauses, in general, simplify the formula (removing all clauses containing such literal, and removing all the occurrences of the opposite literal), clauses of bigger length may create new relations between existing literals. Therefore, their effects on the graph representation are also different.

In general graphs, the addition of edges (preserving the nodes) can only increase its dimension, because tiles may cover more nodes, thus the number $N(r)$ of tiles required to cover the graph may decrease, whereas $N(1)$ does not change. This can only contribute to decrease the slope of function $N(r)$, hence to increase the fractal dimension[4].

In the case of learnt clauses, their addition as well as the simplification of formulas due to unitary clauses modify the VIG and CVIG representations of the formula, and hence may affect its fractal dimension.

In the VIG model, the addition of learnt clauses only introduces new edges, thus as we argued the dimension can only increase. In the CVIG model, the argument is a bit more complicated. Let $N(r)$ be the original number of tiles

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

[4]Notice that the fractal dimension has the same value than the slope of $N(r)$ but opposite sign.

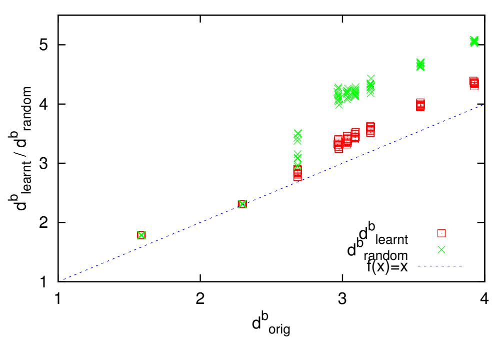


Figure 5.4: Relation between the original fractal dimension d^b_{orig}, and the dimension d^b_{learnt} after adding learnt clauses, or after adding random clauses d^b_{rand}, in random 3-CNF formulas. Learnt clauses are computed after 10^3 conflicts.

and $N'(r)$ the minimum number of tiles needed after adding L learnt clauses. We have $N'(1) = N(1)+L$, since we add L new nodes (clause-nodes). For $r > 1$, we can ensure that the whole graph will be covered using the old tiles and L new tiles to cover new nodes, thus $N'(r) \leq N(r) + L$ in the worst case. Therefore, the dimension can only increase. On the other hand, CDCL SAT solvers do not generally add unitary learnt clauses to the formula, but they propagate them. In both models, this contributes to decrease $N(1)$ by the number of assigned variables, and may contribute to increase $N(r)$, for $r > 1$, due to the elimination and simplification of clauses. Therefore, the 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.

We have conducted some experiments to analyze how the fractal dimension evolves during the execution of the SAT solver. First we show the effect of introducing learnt clauses in random 3-CNF instances with 10^5 variables and distinct clause/variable ratios. In these instances almost all learnt clauses are not unitary, hence we do remove neither nodes nor edges. In Figure 5.4, we plot the dimension d^b_{learnt} after adding learnt clauses w.r.t. the original dimension d^b_{orig}. We observe that the addition of learnt clauses increases the dimension of the formula, as expected theoretically. This increase is bigger for formulas with higher clause/variable ratio. In order to *quantify* the increase in the fractal dimension, we repeat the same experiment replacing learnt clauses by random clauses of the same size, and computing the new dimension d^b_{random} (results are also shown in Fig. 5.4). We observe that in this second experiment the increase in the dimension is bigger than adding learnt clauses: $d^b_{random} \geq d^b_{learnt} \geq d^b_{orig}$. This means that learnt clauses, even in these random formulas, tend to connect variables that were already *close* in the graph. Therefore, their effect in the

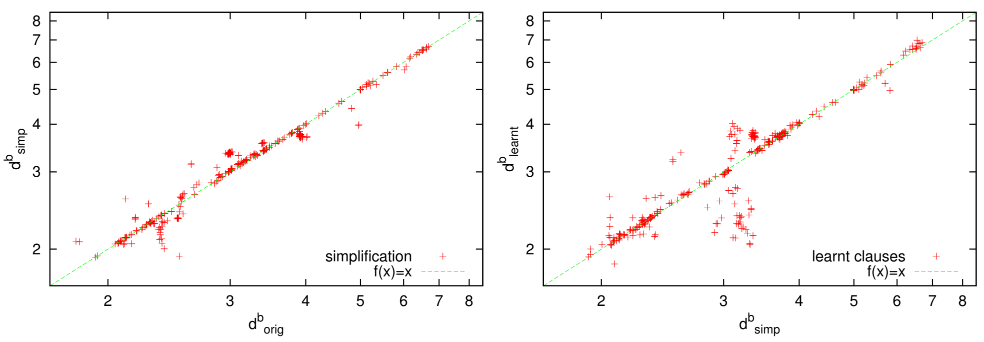


Figure 5.5: Relation between the original fractal dimension d^b_{orig} and the fractal dimension d^b_{simp} after simplifying the formula with the unitary learnt clauses (left), and relation between the fractal dimension d^b_{simp} and the fractal dimension d^b_{learnt} after simplification and adding learnt clauses (right), for all industrial formulas. Learnt clauses are the result of 10^3 conflicts.

fractal dimension is not as important as adding random clauses.

In industrial instances some learnt clauses are unitary. In Figure 5.5 , we analyze separately the effect of simplifying the formula using these unitary clauses (left), and the effect of adding non-unitary learnt clauses (right), after 10^3 conflicts. We observe that, in most of the industrial instances, the fractal dimension after simplifying the formula with unitary learnt clauses d^b_{simp} is, in general, slightly smaller than the original fractal dimension d^b_{orig}, as expected. In some instances, the decrease of d^b_{simp} is quite big. This is the case of families *crypto-md5* and *crypto-sha*. In these families, the effect of propagating these clauses is more impontant. However, we also find few instances for which the dimension d^b_{simp} slightly increases. In the case of learning non-unitary clauses, we observe that the dimension d^b_{learnt} after learning these clauses is slightly greater than the dimension d^b_{simp}, in most of the instances, as expected. This increase is particularly big in families *crypto-aes*, *crypto-gos* and *crypto-vmpc*. This means that learning is connecting distant part of the formula, hence $N^b(r)$ drops off. Again, we also observe some cases where the dimension d^b_{learnt} decreases due to learning.

In Figure 5.6, we show the evoution of the values of the fractal dimension after 10^3, 10^4 and 10^5 conflicts. We observe that, after 10^3 conflicts, dimensions may increase or decrease slightly, depending on the impact of learning unitary clauses (making the fractal dimension decrease) or learning longer clauses (making the fractal dimension increase). However, after 10^5 conflicts, the dimension clearly increases in most of the cases. That means that after a certain number of conflicts, the solver is not learning unitary clauses any more, hence the dimension always increases.

Finally, we want to detect if learnt clauses tend to connect distant parts of the

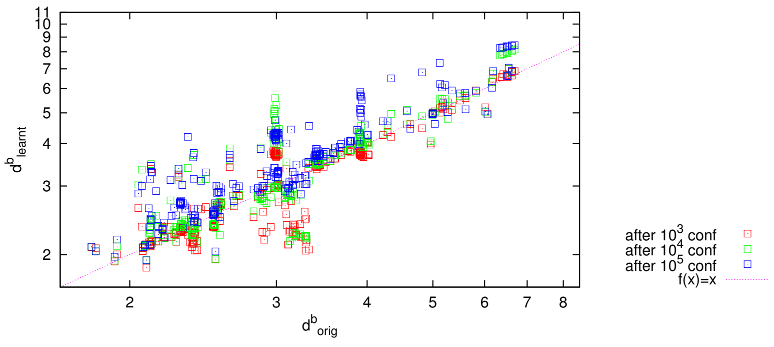


Figure 5.6: Relation between the original fractal dimension d^b_{orig} and the fractal dimensions after learning clauses d^b_{learnt}, in industrial SAT formulas.

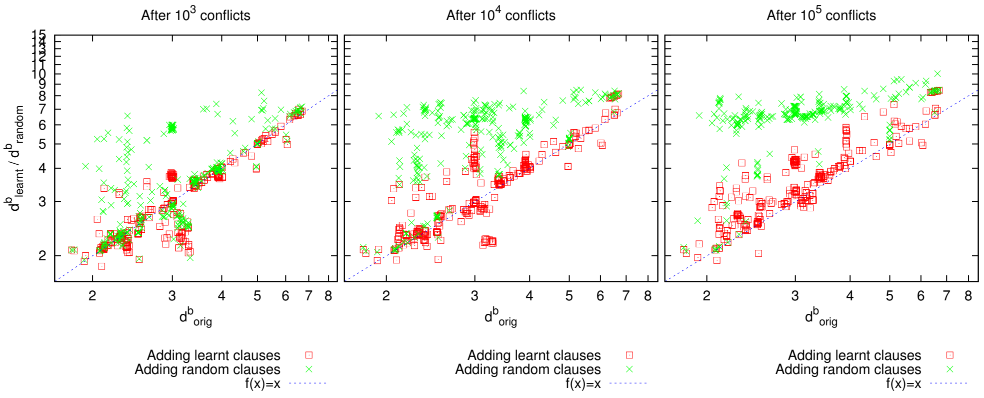


Figure 5.7: Relation between the original fractal dimension d^b_{orig} and the fractal dimensions after adding learnt clauses d^b_{learnt}, or after adding random clauses d^b_{random}, in industrial SAT formulas.

formula in industrial instances. In order to measure it, we *quantify* the variation of the dimension d^b_{learnt} due to the addition of learnt clauses, compared to the dimension d^b_{random} due the addition of the same number of random clauses with the same sizes. In Figure 5.7, we plot the relation between these two dimensions. We observe that the dimension d^b_{random} is considerably bigger than the dimension d^b_{learnt}. This means that the effect of random clauses is much more significant in the fractal dimension, i.e., most of learnt clauses do not contribute to make tiles bigger. In other words, learnt clauses do not reduces the number of tiles needed to cover the graph as randomly generated clauses do. Therefore, they mainly connect nodes inside the tiles, i.e. nodes that where already close or connected. Therefore, learning acts quite locally in the formula.

5.5 Conclusions

This analysis of the fractal dimension of SAT instances shows that most industrial benchmarks exhibit a self-similar structure, with a fractal dimension ranging, in general, between 2 and 4. Fractal dimensions, typical distances and graph diameter are related (small dimension implies big distance and diameter). Hence, industrial SAT instances have a big diameter. Intuitively, this means that we need long chains of implications to propagate a variable instantiation to others. On the other hand, random instances are not self-similar.

We also observe that fractal dimension increases due to learnt clauses. Moreover, the increase is specially abrupt in instances that show exponential decays (for instance, in the family *hardware-velev* or random formulas). Also, this increase is bigger if we substitute learnt clauses by random clauses of the same size. Therefore, learning *does not* contribute very much to connect distant parts of the formula, as one could think.

Part II

Applications

Chapter 6

Pseudo-industrial SAT instances generation

6.1 Introduction

Random SAT formulas can be easily generated on demand. On the contrary, the set of industrial benchmarks, which encode real-world problems, is limited. The problem of generating realistic pseudo-industrial random instances is stated in [Selman et al., 1997] as one of the most important *challenges* for the next few years: *"Challenge 10: Develop a generator for problem instances that have computational properties that are more similar to real world instances"*. This challenge is also stated in [Dechter, 2003; Kautz and Selman, 2003, 2007]. The main motivation of this challenge is improving the process of development and testing of SAT solvers, and their possible specialization.

One important motivation for the development of *pseudo-industrial* SAT instances generators is to *isolate* some known properties of these real-world problems. This allows us to study the impact of these properties on the performance and behavior of SAT solvers. This approach has been already used [Ansótegui et al., 2009b] (see Section 3.4). Using this generator, it was observed that CDCL SAT solvers focus their decisions on the most frequent variables.

In the case of community structure, similar questions also arise. For instance, do SAT solvers concentrate their decisions on variables of the same (or few) communities? Do the conflicts found by the solver relate variables of the same community? How does the activity of each community evolve along the execution of the search? Answering these questions may help to better understand the different ingredients of modern SAT solvers and their impact on the solving process, with the long-term aim of improving them. Recall that, as we have shown in Chapter 4, clear community structure (or high modularity) is a very determining feature of real-world SAT instances.

In this chapter, we present a new model of generation of random SAT instances based on the notion of modularity, called *Community Attachment*. For high values of modularity (i.e., clear community structure), we realistically model pseudo-industrial random SAT formulas. This model also generates SAT instances very similar to classical random formulas using a low value of modularity.

Questions addressed in this chapter:

Question 3. *How can we generate more realistic pseudo-industrial random SAT problems?*

Related publications:

- Giráldez-Cru, J. and Levy, J. (2015). A modularity-based random SAT instances generator. In *Proceedings of the 24st International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1952–1958.
- Giráldez-Cru, J. and Levy, J. (2016). Generating SAT instances with community structure. *Artificial Intelligence.* Accepted with revisions.

The rest of the chapter is organized as follows. We introduce the Community Attachment model in Section 6.2, and we validate it in Section 6.3. This means that this model appropriately generates the expected community structure in the output formulas for different input values of number of variables n and clauses m. In Section 6.4, we prove that the phase transition point is independent on the modularity. In Section 6.5, we give empirical evidence that the performance of SAT solvers is consistent with the expected properties of the generated formulas, i.e. SAT solvers *specialized* in industrial problems perform better in high modular instances than SAT solvers *specialized* in random formulas, and vice versa. We also use this generator to study some components of a CDCL SAT solver in Section 6.6, and we conclude in Section 6.7.

In Section 9.2.5, we discuss some future work about extending the *Community Attachment* model in several ways. Recall the related work on generation of pseudo-industrial random SAT instances in Section **??**, and the scale-free random formulas generator proposed by Ansótegui et al. [2009b] and described in Section 3.4.

6.2 Community Attachment model

In the classical random k-SAT model, a random formula $F_k(n, m)$ is a set of m clauses over n variables, where clauses are chosen uniformly and independently among all $2^k\binom{n}{k}$ non-trivial clauses of length k. A non-trivial clause of length k contains k distinct, non-complementary literals. We present a new model of random formulas: the *Community Attachment* (CA) model, which is parametric in a probability p and a number of communities c of the set of variables. It allows the generation of formulas of any desired modularity.

Algorithm 5: Community Attachment

```
   Input: int $n$, $m$, $c$, $k$; real $Q$;
   Output: $k$-CNF SAT Instance $\Gamma$
1  $\Gamma := \emptyset$;
2  $p := Q + 1/c$;
3  for $j \in \{1, \dots, m\}$ do
       // Select the community $c_i$ of each literal
4      if $rand([0,1]) \leq p$ then        // all literals in the same community
5          $r := rand(\{1, \dots, c\})$;
6          for $i \in \{1, \dots, k\}$ do
7              $c_i := r$;
8      else                                // all literals in distinct communities
9          for $i \in \{1, \dots, k\}$ do
10             repeat
11                 $c_i := rand(\{1, \dots, c\})$;
12             until $\forall i' < i (c_{i'} \neq c_i)$;
       // Create the clause $C$
13     $C := \emptyset$;
14     for $i \in \{1, \dots, k\}$ do
15         repeat
16             $x_i := rand(\{\lfloor (c_i - 1)\,\frac{n}{c} \rfloor + 1, \dots, \lfloor c_i\,\frac{n}{c} \rfloor\})$;  // $x_i$ random var of
               community $c_i$
17         until $\forall i' < i (x_{i'} \neq x_i)$;          // Avoid trivial clauses;
18         $C := C \bigvee rand(\{-1, 1\}) \cdot x_i$;          // random polarity
19     $\Gamma := \Gamma \bigwedge C$;
20 return $\Gamma$;
```

Definition 6.1. Community Attachment. Let N be a set of n variables, a partition P of N into c pairwise disjoint communities of the same size n/c, with $k \leq c \leq n/k$, and a real value $0 \leq p \leq 1$. A random formula $F_k(n, m, c, p)$ is a set of m non-trivial clauses with k literals over the n variables, selected independently as follows. With probability p, choose a clause uniformly among all $c\,2^k \binom{n/c}{k}$ clauses with all literals in the same community; and with probability $1 - p$, a clause uniformly among all $\left(\frac{2\,n}{c}\right)^k \binom{c}{k}$ clauses with all literals in distinct communities.

Notice that in the previous definition we need to impose the restriction:

$$k \leq c \leq n/k$$

to ensure that there always exists at least one possible clause to select. Notice also that for $k = 2$ and $p = \frac{n/c-1}{n-1}$ we have the classical 2-SAT model: $F_2(n, m) =$

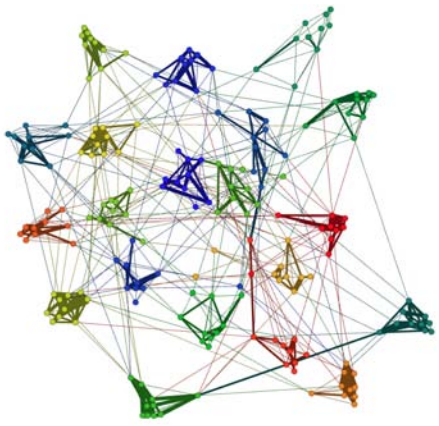


Figure 6.1: Variable Incidence Graph (VIG) of a random instance from $F_k(n,m,c,p)$ generated with $n=200$ variables, $m=425$ clauses of length $k=3$, $c=40$ communities, and modularity $Q=0.8$ ($p=Q+1/c$). Variables (nodes) of the same community are plotted with the same color. Edges are scaled according to their weight.

$F_2(n,m,c,\frac{n/c-1}{n-1})$, but for bigger k the Community Attachment model does not subsume the classical model.

Given a SAT formula Γ with n variables and m clauses, consider the VIG G of Γ. Our model ensures a lower-bound for the modularity of this graph.

Theorem 6.1. Given a formula $\Gamma \in F_k(n,m,c,p)$, let G be its VIG. The average modularity of G is bounded as:

$$E[Q(G)] \geq p - \frac{1}{c}$$

Proof. Recall that modularity is defined as the maximal modularity for all possible partitions of the nodes into communities. Here we consider the partition used to generate the formula. For this particular partition P, when we select a clause with all variables in the same community (with probability p), we get $\binom{k}{2}$ internal edges. The sum of the weights of the edges generated by a single clause is always 1. Therefore, the fraction of internal edges is, on average, $\frac{p\,m}{m}$. The sum of nodes degrees is $2m$, thus $2m/n$ is the expected node degree. Since n/c is the number of nodes per community, the sum of nodes degrees in one community is on average $\frac{n}{c}\frac{2m}{n}$.

Summarizing, for this partition P, we get

$$E[Q(G,P)] = \frac{p\,m}{m} - c\left(\frac{\frac{n}{c}\frac{2m}{n}}{n\frac{2m}{n}}\right)^2 = p - \frac{1}{c}$$

that is a lower-bound for the expected modularity $E[Q(G)]$. □

When p close to 1, the expected modularity $E[Q(G)]$ is very close to this lower-bound $p - 1/c$, because the partition used in the formula generation is very similar to the optimal. Therefore, we can use the previous theorem to generate formulas with a desired modularity Q. We simply take:

$$p = Q + \frac{1}{c} \tag{6.1}$$

which ensures at least a modularity Q. In practice, as we will see in Section 6.3, the formulas we obtain have a modularity $Q \approx p - 1/c$, except when p and m/n are small.

Notice that the previous lower-bound of the modularity depends of the graph model used to represent the formula. For instance, consider the CVIG model of that instance and the following partition: all variable-nodes and all internal clause-nodes in their corresponding community, and an extra community with all external clause-nodes. If, instead of the VIG model, we use the CVIG G', we can get a lower-bound $E[Q(G')] \geq p - p/c$. Notice, however, that in the case we use the definition of modularity for bi-partite graphs, which is slightly different. Nevertheless, both approximations are very similar. Since most of the methods in the literature to compute the community structure are efficient for non bi-partite graphs (as the VIG model), we use the approximation of Theorem 6.1.

In Algorithm 5, there is an implementation of the Community Attachment random formulas generator from $F_k(n, m, c, p)$. Using $p = Q + 1/c$ these formulas will have an expected modularity close to Q.

In Figure 7.1, we represent the VIG of an instance generated with $n = 200$ variables, $m = 425$ clauses, modularity $Q = 0.8$ and $c = 40$ communities. In this plot, variables (nodes) of the same community are plotted with the same color, and edges are scaled according to their weight. As the value of the modularity is high, it is more likely that a clause relates variables of the same community. Therefore, the weight of the edges connecting nodes (variables) of the same community is higher, as expected.

6.3 Validation of the model

In order to analyze the community structure of the SAT instances obtained with our model, we have generated some sets of random formulas for different values of $Q \in \{0.9, 0.8, 0.7, 0.5, 0.3\}$ (hence $p = Q + 1/c$), and for different values of number of communities $c \in \{10, 20, 40, 80\}$. Each set contains 50 random instances. Remark that the modularity Q of (real) industrial SAT instances is

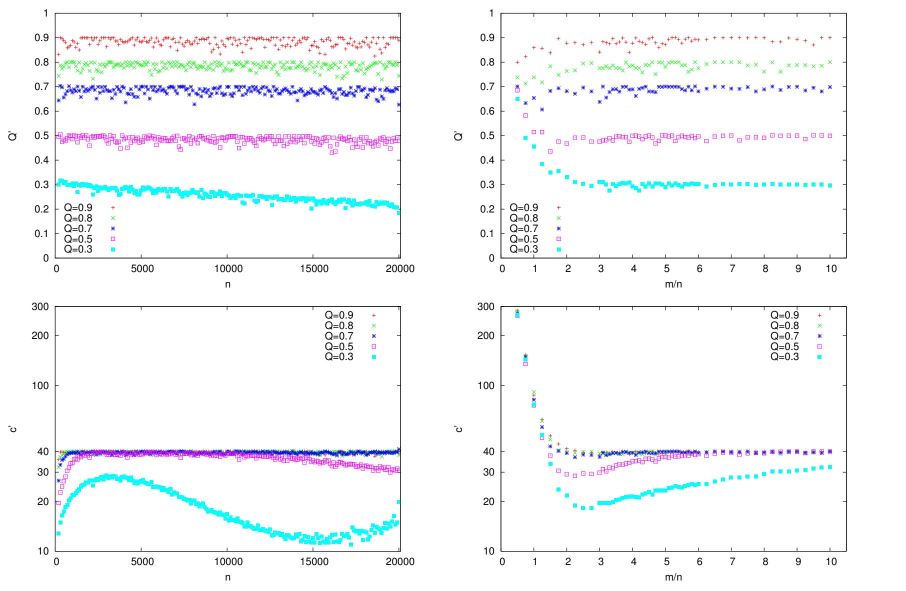


Figure 6.2: Approximations of modularity Q' (top) and number of communities c' (bottom) of some sets of random SAT formulas from $F_k(n, m, c, p)$, varying the number of variables n with $m/n = 4$ (left), and varying the clause/variable ratio m/n with $n = 1000$ (right), with $k = 3$, $c = 40$, and $p = Q + 1/c$. Each plotted data is the average of 50 instances.

usually greater than 0.7 [Ansótegui et al., 2012], while no modularity greater than 0.3 is found for classical random k-CNF formulas. Moreover, the number of communities c is usually in the interval $[10, 100]$ [Ansótegui et al., 2012].

In Figure 6.2, we analyze their modularity Q' (top) and their number of communities c' (bottom), varying the number of variables n, for a fixed clause/variable ratio $m/n = 4$ (left), and varying the clause/variable ratio m/n, for a fixed number of variables $n = 1000$ (right). In this experiment, the number of communities c is fixed to $c = 40$. Notice that the main goal of this experiment is to show that our model generates instances with modularity Q' and number of communities c' similar to the input parameters Q and c, for any number of variables n and clauses m. We use the GFA algorithm described in Alg. 3 to compute an approximation of Q' and c'. In fact, this algorithm computes a lower-bound of the modularity. The dispersions of the approximated Q' and c' are very small, so they are not shown in the plots.

We observe that the modularity Q' and the number of communities c' are almost unaffected by these variations of n and m/n. In general, the approximation computed for Q' is slightly smaller than expected, and the partition into communities is also very similar to the partition used in the generation. For small values of the clause/variable ratio m/n and the probability p, the num-

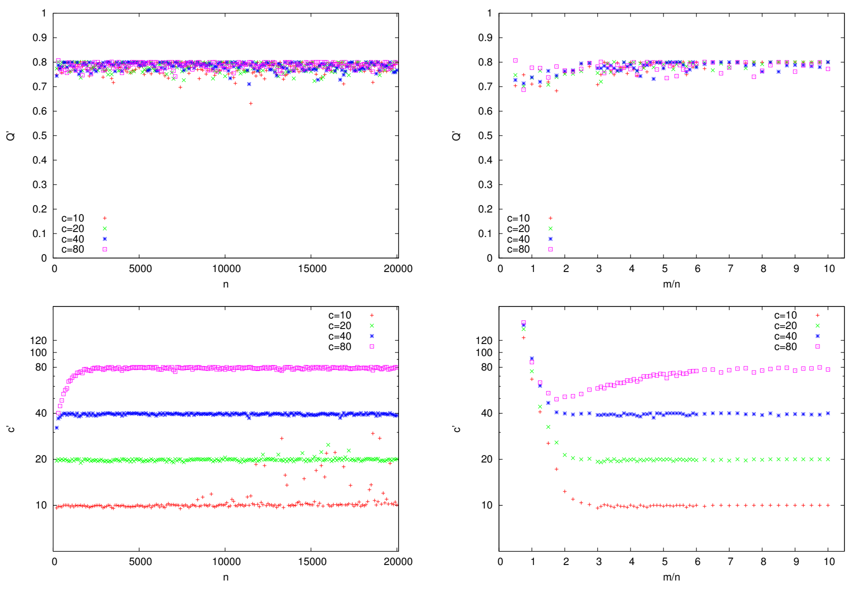


Figure 6.3: Approximations of modularity Q' (top) and number of communities c' (bottom) of some sets of random SAT formulas from $F_k(n, m, c, p)$, varying the number of variables n with $m/n = 4$ (left), and varying the clause/variable ratio m/n with $n = 1000$ (right), with $k = 3$, $Q = 0.8$, and $p = Q + 1/c$. Each plotted data is the average of 50 instances.

ber of clauses relating variables of the same expected community is very small. This produces the existence of some unconnected sub-communities within each expected community. Hence, Q' and c' are much greater than expected, and Q' cannot be estimated as $p - 1/c$. When we generate formulas with small values of p, e.g. $Q = 0.3$ and $c = 40$, we observe that, although the formulas have a guaranteed lower-bound of $Q \geq 0.3$, the computed approximation of Q' is smaller (close to 0.2 when $n \approx 20000$). The number of communities c' is also smaller than the expected c. In this case, this error is not produced by our model. It is due to the greediness of the algorithm used to approximate Q', which is not able to find a similar partition to the one used in the generation.

In a second experiment, we generate families of instances with the same number of variables n and the same clause/variable ratios m/n than in the previous experiment, and varying the number of communities c with a fixed modularity $Q = 0.8$. In Figure 6.3, we represent the results. We observe that the expected modularity Q' as well as the expected number of communities c' is almost unaffected in these families, except, as expected, for small values of n and m/n. In fact, when the value of the number of communities c is high enough (as actual industrial SAT instances have), its relevance in our model is very small, as expressed by Equation 6.1. Hereinafter we use in our experiments a fixed value

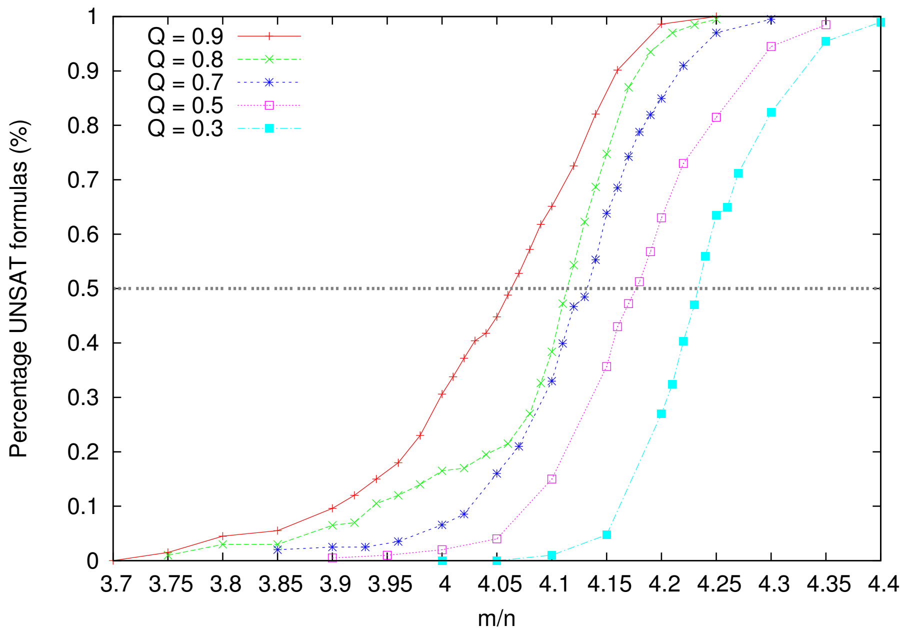


Figure 6.4: Fraction of UNSAT formulas for some sets of 200 random SAT formulas from $F_k(n, m, c, p)$, with $k = 3$, $c = 40$ and $p = Q + 1/c$, and varying the clause/variable ratio m/n (see the number of variables n of each family in Table 6.1).

of number of communities $c = 40$, assuming that this value is representative for real application problems, and therefore without altering general conclusions observed in the empirical results.

6.4 Phase transition

In classical random k-CNF instances, some interesting properties, as the satisfiability or the hardness, are correlated to the clause/variable ratio m/n [Mitchell et al., 1992]. The Satisfiability Threshold Conjecture suggested that it may exist a critical ratio r, such that below this point all formulas from $F_k(n, m)$ are SAT (under-constrained) and above it they all are UNSAT (over-constrained) with uniformly positive probability, when n tends to infinity. Moreover, the hardness of these instances is also characterized by this parameter: closer to this ratio, harder the instance.

Experimentally, this phase transition point has been shown to be around $r \approx 4.26$ for $k = 3$. Mertens et al. [2006] used the *cavity method* to derive the exact value of this threshold, predicting the value $r = 4.267$. This hypothesis has been recently proven by Coja-Oghlan [2014].

Q	r'	n	solver	$\bar{R}$	$S[R]$
0.9	**4.06**	5000	Glucose	80.86	125.28
0.8	**4.11**	2000	Glucose	291.87	1217.23
0.7	**4.13**	1200	Glucose	211.64	791.76
0.5	**4.18**	600	March	544.19	1051.81
0.3	**4.24**	600	March	3492.36	3117.23

Table 6.1: Approximations of the phase transition point r' of some families of 200 random SAT instances from $F_k(n, m, c, p)$, with $k = 3$, $c = 40$ and varying Q. We also report the number of variables n, the solver and runtime R (average $\bar{R}$ and standard deviation $S[R]$) needed to solve them.

In this section we check if this phenomenon also exists in the random SAT instances generated with our model, and if the new transition point, noted r', differs from the classical $r' \neq r$. In Figure 6.4, we represent the fraction of UNSAT instances for some sets of random formulas with distinct Q, varying the clause/variable ratio m/n. We observe that the fraction of UNSAT formulas increases with m/n. Therefore, for small (big) values of m/n, nearly all formulas are SAT (UNSAT). When Q is small, the value r' is close to the classical $r \approx 4.26$. Recall that when $p \approx 1/c$, our model is quite similar to the classical random k-SAT model. However, we also observe that, when Q increases, r' decreases.

In our experimentation, for each family of instances we use the biggest value of number of variables n allowing us to solve each instance in less than 3 hours. Therefore, this value may change for each family of instances. In Table 6.1 we report the phase transition point r' we found for some families of formulas with $k = 3$, varying the modularity Q. We also report the solver we used to solve the formulas, as well as the average and standard deviation of the runtime used by this solver. Remark that as the number of variables n of each family is different, their runtimes cannot be used to compare their hardness. We observe that the phase transition point r' decreases as the modularity Q increases. In Table 6.2, we report the phase transition point r' of some families of instances with $k = 4$. Again, the phase transition point r' of these families tends to decrease as the modularity Q increases. Notice that the generated formulas with a high modularity are hard, compared to industrial formulas. This is because industrial instances have other properties, like the scale-free structure, that contribute to make them easier.

The natural question is if this decrease in r' is also valid for n tending to infinity. In order to explain this decrease in the phase transition point r', and predict the behavior when n tends to infinity, we will consider the extreme case with $p = 1$. In these formulas, clauses only contain variables of the same community. Therefore, the formula is composed by c unconnected sub-formulas, and the whole formula is UNSAT if, and only if, at least one of the sub-formulas is UNSAT. Moreover, in this extreme case, all sub-formulas follow the classical model $F_k(n/c, m')$, for some m'. On average, all sub-formulas contain $E[m'] = m/c$

Q	r'	n	solver	$\bar{R}$	$S[R]$
0.9	**9.37**	2000	Glucose	10.14	10.45
0.8	**9.50**	1000	Glucose	2244.24	4122.35
0.7	**9.55**	600	Glucose	2239.89	3649.96
0.5	**9.60**	250	March	1524.22	1550.52
0.3	**9.79**	200	March	566.83	581.20

Table 6.2: Approximations of the phase transition point r' of some families of 20 random SAT instances from $F_k(n, m, c, p)$, with $k = 4$, $c = 40$ and varying Q. We also report the number of variables n, the solver and runtime R (average $\bar{R}$ and standard deviation $S[R]$) needed to solve them.

clauses; and all of them contain n/c variables. Hence, the average clause/variable fraction in sub-formulas is also $E[\frac{m'}{n/c}] = \frac{m/c}{n/c} = m/n$. However, even when the fraction m/n is smaller than the classical r (and so the expected clause/variable ratio of the formula), with some probability, some of the sub-formulas may get a large portion of clauses m' such that $\frac{m'}{n/c} > r$. This makes that sub-formula UNSAT with high probability. This has the effect of decreasing the phase transition point for finite n and c.

When n/c tends to infinity, the situation is completely different as the following theorem states.

Theorem 6.2. The set of formulas $F_k(n, m, c, p)$, with $p = 1$ and any value of c satisfying $n/c \to \infty$, has a phase transition point r' at the same clause/variable ratio r of the classical formulas $F_k(n, m)$.

Proof. Let $r' = m/n$ and r be the classical phase transition point. The minimal r' such that the probability that some of the sub-formulas has more than $r\,n/c$ clauses (hence it is UNSAT with high probability when n/c tends to infinity), will be the phase transition point for this special case $p = 1$ of our model.

The probability that a given community P_i contains $r\,n/c$ clauses, when the formulas has m clauses, is

$$P(P_i \text{ is UNSAT}) = \frac{\binom{m}{r\,n/c}\,(c-1)^{m-r\,n/c}}{c^m}$$

Let $m = r'\,n$ be the number of clauses of the whole formula.

There are two cases:

First, we assume that the number of communities c tends to ∞ (but slower than n, hence n/c also tends to ∞).

When $m, n \to \infty$, and $m/n \to \infty$, the binomials $\binom{m}{n}$, may be approximated as:

$$\begin{aligned}\binom{m}{n} &= \frac{m\cdot(m-1)\cdots(m-n+1)}{n!} \approx \frac{(m-n/2)^n}{\sqrt{2\pi n}(n/e)^n} = \\ &= \frac{m^n\left(1-\frac{1}{2m/n}\right)^{\frac{2m}{n}\cdot\frac{n^2}{2m}}}{\sqrt{2\pi n}(n/e)^n} \approx \frac{m^n(1/e)^{n^2/2m}}{\sqrt{2\pi n}(n/e)^n} = \\ &= \frac{1}{\sqrt{2\pi n}}\left(\frac{m\,e}{n\,e^{n/2m}}\right)^n\end{aligned}$$

using the middle value in the numerator, and the Stirling approximation in the denominator.

When $c \to \infty$, we can also approximate

$$(c-1)^{m-r\,n/c} = c^{m-r\,n/c}\,(1-1/c)^{c\frac{m-r\,n/c}{c}} \approx \frac{c^{m-r\,n/c}}{e^{\frac{m-rn/c}{c}}}$$

Replacing these two approximations, and $m = r'\,n$ we get

$$P(P_i \text{ is UNSAT}) \approx \frac{1}{\sqrt{2\pi r n/c}}\left(\frac{r'}{r}\exp\left(1-\frac{r'}{r}+\frac{1}{c}(1-\frac{r}{2r'})\right)\right)^{r\,n/c}$$

For $n/c, c \to \infty$, this function is dominated by the exponential factor

$$P(P_i \text{ is UNSAT}) = \mathcal{O}\left(\left(\frac{r'}{r}\exp\left(1-\frac{r'}{r}\right)\right)^{r\,n/c}\right)$$

The base of the exponentiation is strictly smaller than one except for $r = r'$. Therefore, when the number of communities and their size both tend to infinity, even in the extreme case $p = 1$, the probability that the formula is UNSAT is zero, for $r' < r$, i.e. the phase transition point is the same as for the classical random formulas.

In the second case, we assume that c is finite. Then, the approximation we have used for the binomial is not correct. When k is constant and $n \to \infty$, we may use

$$\binom{kn}{n} \approx \frac{1}{\sqrt{2\pi n\frac{k-1}{k}}}\left(\frac{k^k}{(k-1)^{k-1}}\right)^n$$

In this case we get

$$P(P_i \text{ is UNSAT}) = \mathcal{O}\left(\left(\frac{(\frac{r'}{r}c)^{\frac{r'}{r}c}}{(\frac{r'}{r}c-1)^{\frac{r'}{r}c-1}}\;\frac{(c-1)^{\frac{r'}{r}c-1}}{c^{\frac{r'}{r}c}}\right)^{r\,n/c}\right)$$

As in the previous case, the base of the exponentiation is one only when $r' = r$. Therefore, the phase transition point is also just the same as for classical random formulas. □

In the classical model, we recall that the phase transition point is only *applicable* when the size of the formula goes to infinity. In the context of SAT instances, the phase transition point r is, by definition, the clause/variable ratio such that *all* formulas below (equivalently, above) such ratio are satisfiable (equiv., unsatisfiable). In other words, the probability that an instance is UNSAT (equiv., SAT) has a value tending to 0 (equiv., 1) for any clause/variable ratio $r - \epsilon$ (equiv. $r + \epsilon$), for any $\epsilon > 0$. This abrupt change in the probability is represented as a vertical line in the point r, and this would only occur in extremely large instances, as suggested in the Satisfiability Threshold Conjecture. In the case of formulas of finite size, we observe that this abrupt change does not exist. On the contrary, we find an interval $(r - \epsilon, r + \epsilon)$ where the probability that an instance is UNSAT smoothly goes from 0 to 1, and this change becomes more abrupt as the size of the formula grows.

In the case of the Community Attachment model, we observe the same behavior. Therefore, both empirical observations of Figure 6.4 and the formal proof expressed in Theorem 6.2 match with the behavior expected from the classical model.

6.5 SAT solvers performance

In this section we show that *industrial-specialized* SAT solvers exploit the community structure of the formula, whereas *random-specialized* solvers do not.

In Figure 6.5 we compare the performance of the SAT solvers Glucose [Audemard and Simon, 2009] (version 3.0) and March [Heule et al., 2004] (version br) over some sets of SAT formulas generated with our model, with distinct modularity values. While Glucose is a CDCL SAT solver which has been shown very good for solving industrial problems, March is a Look-ahead SAT solver commonly used to solve random k-CNF instances. We use sets of instances from $F_k(n, m, c, p)$ with a clause/variable ratio m/n in the phase transition point, a number of communities $c = 40$ and a clause length $k = 3$. We adjust the number of variables as in Table 6.1, in order to ensure that any of these solvers solve all formulas in a timeout of 3 hours.

We observe that, for high modularities (see $Q = 0.9$), Glucose solves all the instances, but March is only able to solve few UNSAT instances. More precisely, they are the instances in which there exists a very small unsatisfiability core, composed of variables of only one or few communities. Notice that higher the modularity, more likely to find such instances with small refutations. It is also interesting to remark that Glucose also solves UNSAT formulas faster than SAT formulas when their modularity is high. As Q decreases, March is able to solve more instances (see $Q = 0.7$), and it starts to be as fast as Glucose, if it is not faster, when the modularity is small enough (see $Q = 0.5$). Finally, when Q is very small (see $Q = 0.3$), March is able to solve all the instances but Glucose only solves few of them. Remark that the number of variables is not the same for every family. We can conclude that a high modularity makes formulas easier to be solved by CDCL SAT solvers.

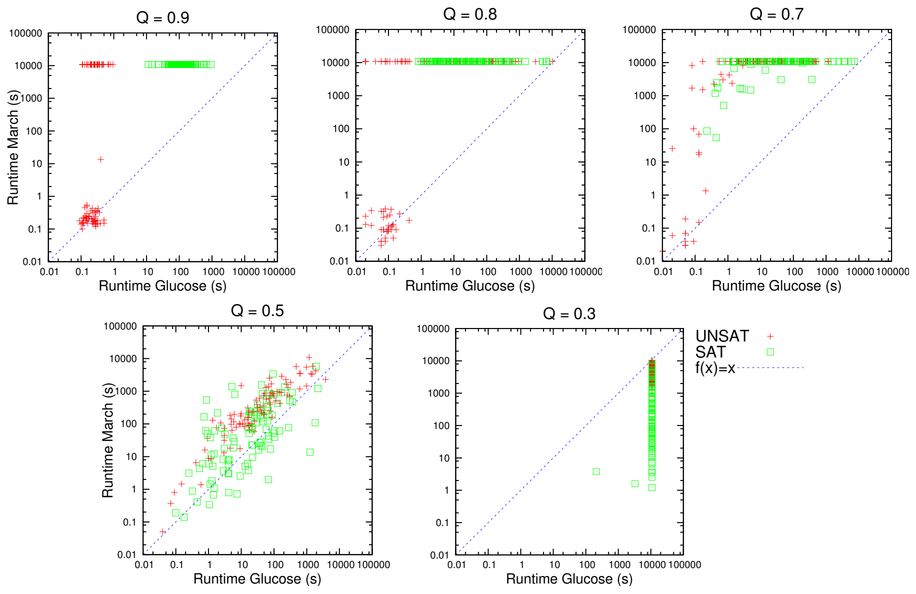


Figure 6.5: Relation between the runtimes (seconds) of Glucose and March, for some sets of 200 random SAT instances with $Q \in \{0.9, 0.8, 0.7, 0.5, 0.3\}$, $k = 3$ and $c = 40$ at the phase transition point (i.e., using families of Table 6.1). The timeout is set to 3 hours.

In Figure 6.6 we compare the performance of these two solvers with the instances of Table 6.2, i.e. $k = 4$. Again, we observe that high modular formulas (see $Q = 0.9$) are easy for Glucose, but March is only able to solve those of them having a small unsatisfiability core. As the modularity decreases, both solvers solve all instances, but March is still some orders of magnitude slower (see $Q = 0.5$). Finally, when the modularity is small, March shows a better performance than Glucose (see $Q = 0.3$).

These experiments suggest that the performance of the solver is affected by the structure of the formula (e.g., its community structure) independently of the lengths of the clauses.

6.6 Analyzing the components of a CDCL SAT solver

One of the most important motivations for the development of this generator is to better understand the connections between the community structure of a SAT instance and the SAT solver components, with the long-term aim of improving them. In this section, we use our generator to study two main components of a CDCL SAT solver: the branching selection heuristics, and the conflict analysis

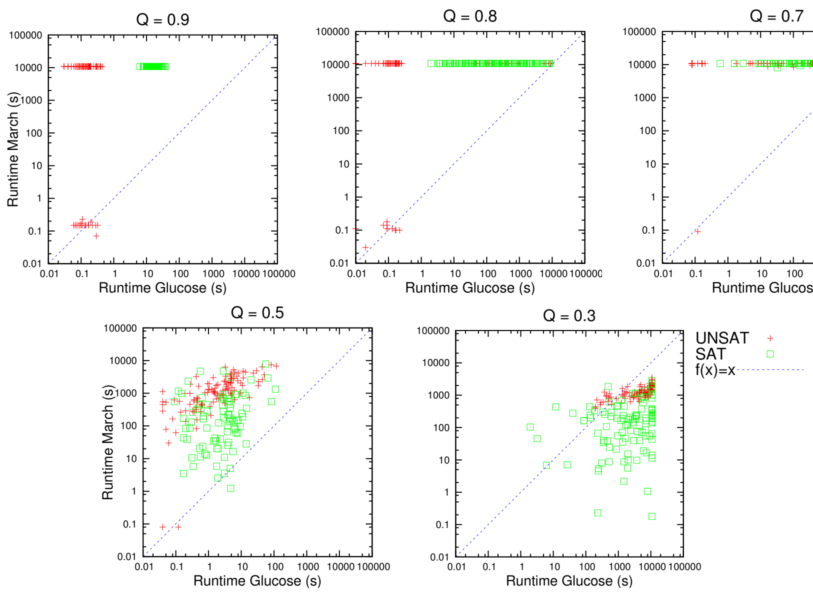


Figure 6.6: Relation between the runtimes (seconds) of Glucose and March, for some sets of 200 random SAT instances with $Q \in \{0.9, 0.8, 0.7, 0.5, 0.3\}$, $k = 4$ and $c = 40$ at the phase transition point (i.e., using families of Table 6.2). The timeout is set to 3 hours.

and clause learning mechanism. Notice that these components are related in general. For instance, the activity of the variables participating in a conflict is increased, and the most active (and unassigned) variable is selected as the next decision variable. Even though, it seems convenient to study these components separately.

In this section, we use MiniSAT [Eén and Sörensson, 2003] (version 2.2) as a representative CDCL SAT solver. This is a very well known SAT solver on which many other SAT solvers, as Glucose, are based. Being a less sophisticated SAT solver allows us to analyze with more precision the impact of certain CDCL techniques on the community structure, without possible *noises* of other components. For instance, MiniSAT uses the Luby series to determine the number of conflicts between two restarts whereas Glucose uses a dynamic LBD-based strategy for this purpose. Therefore, while one can know when restarts are performed by MiniSAT, this is unknown *a priori* in Glucose.

We represent the results for the experiments of a random SAT instance generated with our model, of the family with $n = 1000$ variables, $m = 4200$ clauses, clause length $k = 3$, modularity $Q = 0.8$ and $c = 10$ communities. In this case, we use this reduced number of communities to improve the visualizations of results. However, similar results can be obtained with other typical values (e.g., $c = 40$). We remark that, even when we only plot results for a single

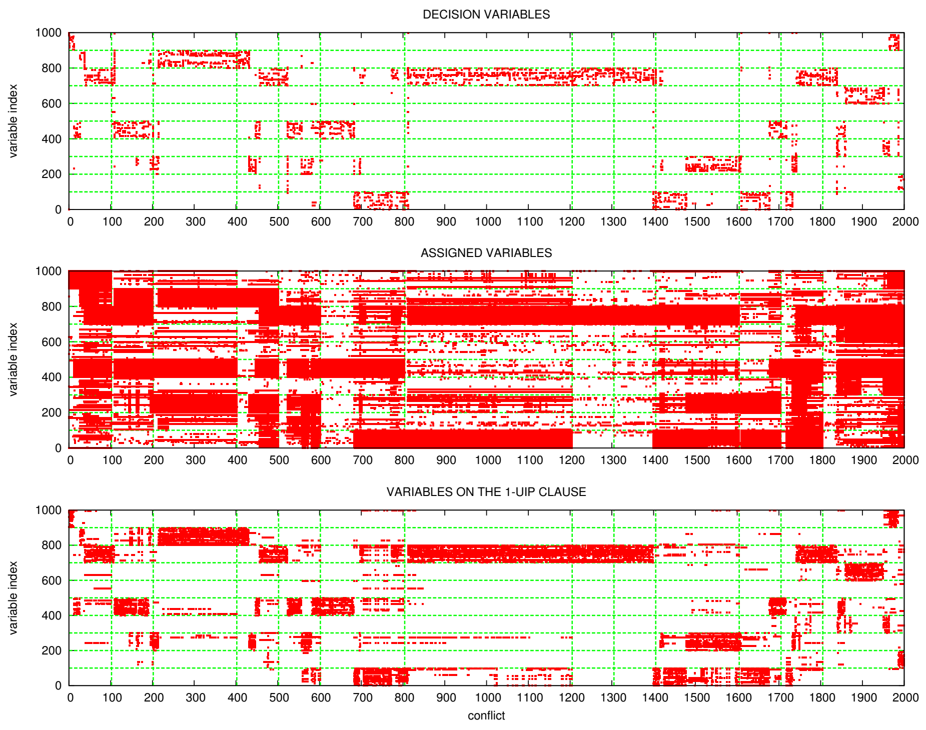


Figure 6.7: Decision variables (top), assigned variables (center) and variables on the learnt clause (bottom) along the first 2000 conflicts of the execution of MiniSAT (i.e., with 1-UIP learning scheme) on a random instances with $n = 1000$, $m = 4200$, $k = 3$, $c = 10$ and $Q = 0.8$. Horizontal lines split the set of variables belonging to each community, and vertical lines represent the restarts.

instance, similar behaviors are observed in all instance of the family. Therefore, the conclusions drawn in this section are *general*. Recall that our generator, for simplicity, assigns n/c consecutive variables to each community. For instance, community c_1 contains variables with indexes from 1 to 100 (both communities and variables are 1-based numbered). MiniSAT solved this instances in 1.9 seconds, taking 18226 decisions and finding 11722 conflicts.

As we want to analyze the trace of the solver, in the following figures (Fig. 6.7 and 6.8), we plot the results using the X-axis to represent the number of conflict, and the Y-axis the index of variables. For clarity, we only plot the first 2000 conflicts. There are horizontal lines to split the set of variables belonging to each community, and vertical lines to represent the restarts along the execution.

First, we want to know if SAT solvers concentrate their decisions on variables of the same (of few) communities along their execution. In Figure 6.7 (top) we represent which variables are used to branch, i.e., the decision variables. As the

X-axis represents the number of conflicts, the Y-axis shows the set of variables decided between two consecutive conflicts. We observe that the solver tends to focus its decisions on variables of the same community, during a period of time. After a while (when all variables of this community are assigned), it changes to another community. This behavior is repeated during the whole execution. The time the solver stays deciding in the same community is indeterminate, and does not depend on the restarts.

Second, in Figure 6.7 (center) we analyze the set of assigned variables. Notice that this set contains, not only decision variables, but also implied variables (i.e., variables whose value is *forced* by constraint propagation). We observe that all variables of the community where the solver has been focusing get assigned, and remain assigned when the solver decides to change to another community. In the next restart, all assigned variables in modules where the solver is not focused, get unassigned. For example, after 400 conflicts, a restart occurs. Before this restart, the solver was focused on community c_9, but all variables of communities c_3 and c_5 were also assigned (because the solver had also been focusing on these communities before c_9). After the restart, only variables of c_9 are assigned again. Therefore, restarting policy has the effect of reinforcing the focus of assigned variables on the same community.

Finally, we want to study if the conflicts found by the solver relate variables of the same community. In Figure 6.7 (bottom) we plot which variables appear in the 1-UIP clause learnt after analyzing the conflict. We observe that, in general, conflicts relate variables of few communities. In fact, this clause mainly contains variables of the community where the solver is focusing its last decisions, and (very) few variables of the rest. Notice that all of these variables had to be previously assigned. We also observe that in the last steps of the search (not shown in the plot), there exist many conflicts relating almost all communities.

In a second experiment, we want to investigate the relation between the clause learning techniques used by the SAT solver and the community structure of the formula. To this purpose, we modify the solver MiniSAT with another learning strategy: the *decision-induced clause learning scheme*. This strategy learns the decision variables that implies the conflict (i.e., it explores the whole implication graph of the conflict till the decision nodes). This is one of the most classical learning strategies, and it was proposed in GRASP [Marques-Silva and Sakallah, 1999].

In Figure 6.8, we solve the same instance of the previous experiment using a modified version of MiniSAT with a *decision-induced clause learning scheme*. As for the 1-UIP schema, we represent the decision variables (top), the assigned variables (center), and the variables belonging to the learnt clause (bottom), for the first 2000 conflicts. Using this learning strategy, the solver require 157726 decisions and 113012 conflicts to solve the instance, spending a total of 23.1 seconds. Notice that this is approximately one order of magnitude slower. The reason of this experiment is to show how CDCL techniques, when used all together, help the solver to focus on particular communities along the search. On the contrary, small changes in these techniques may provoke that community

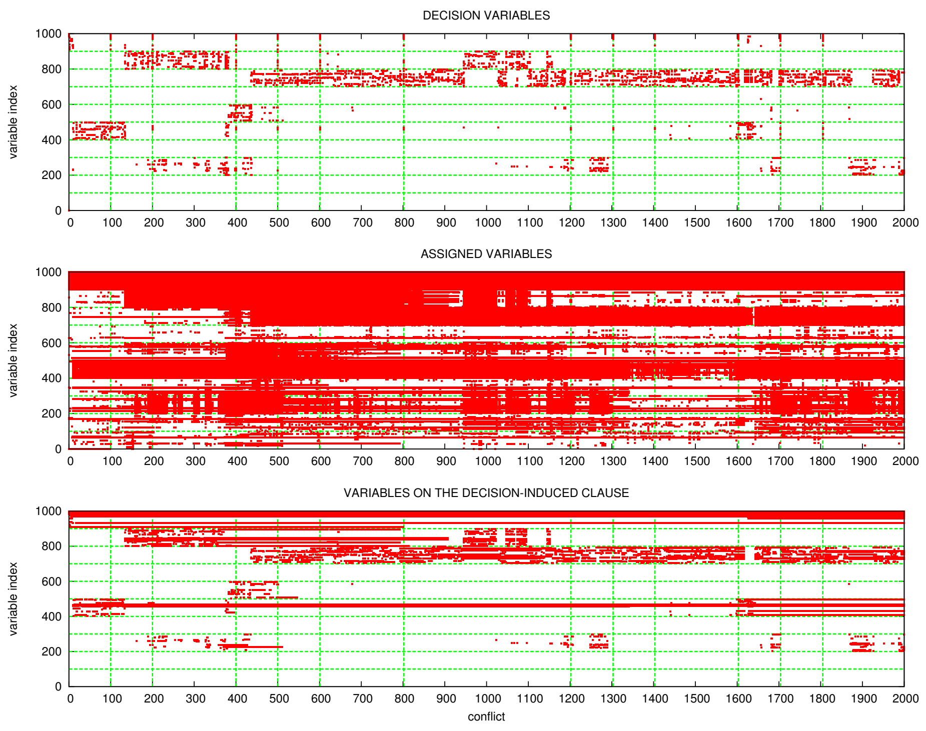


Figure 6.8: Decision variables (top), assigned variables (center) and variables on the learnt clause (bottom) along the first 2000 conflicts of the execution of a modified version of MiniSAT with a *decision-induced clause learning scheme*, on a random instances with $n = 1000$, $m = 4200$, $k = 3$, $c = 10$ and $Q = 0.8$. Horizontal lines split the set of variables belonging to each community, and vertical lines represent the restarts.

structure is not explicitly considered any more, affecting thereby the overall performance of the solver. This may explain the success of these technique on benchmarks with a clear community structure, as industrial SAT instances are.

First, we observe that the solver also tends to focus its decisions on communities with this learning strategy. However, this phenomenon is less clear than in the previous experiment. For instance, between conflicts 200 and 400, the solver is focusing on communities c_3 and c_9 at the same time. This effect can be explained with the next observation. Second, we show that many restarts have no effect on most of the assigned variables. That means that even when restarts remove the value of all assigned variables, they are re-assigned again afterwards. See, for instance, community c_{10}: it is assigned at the beginning of the search, and it remains assigned during most of the execution. Finally, we observe that the variables belonging to the learnt clause correspond in fact to

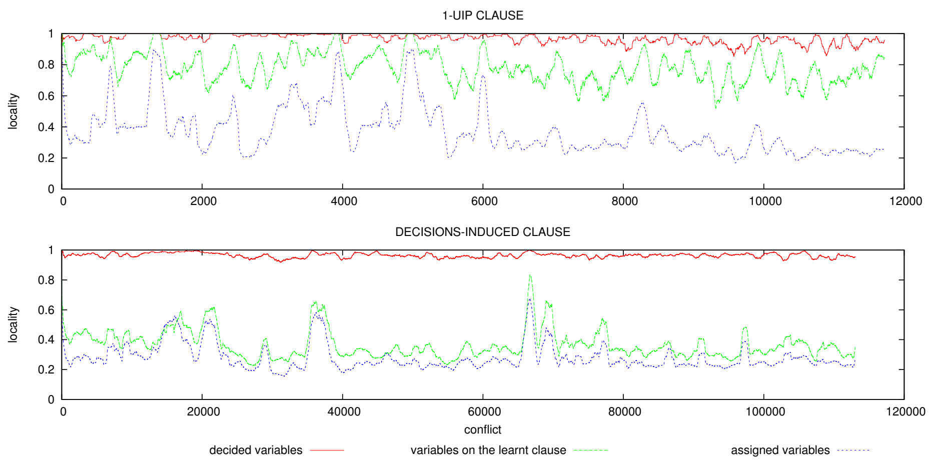


Figure 6.9: Evolution of the locality of the set of decided variables, set of assigned variables and learnt clause along the execution of 1-UIP based solver (top) and a decision-induced learning based solver (bottom).

the variables of the communities where the solver was focused at the beginning of the search. Therefore, the activity of these variables are constantly increased, and hence, they are assigned once they become unassigned (i.e., after a restart). In fact, the presence of these variable in the learnt clause explains why they are re-assigned after each restart. These observations allow us to understand how the 1-UIP, in joint with the activity-based heuristics, help the solver to focus on communities, and this may explain the different performance (e.g., number of decisions and conflicts, runtime) spent to solve an industrial instance. All these effects cannot be observed if we use a formula with a low modularity because the communities are not so well delimited.

In a last experiment, we want to analyze the *locality* of the solver during the search. In other words, we want to *measure* how much the solver focuses its decisions and learnt clauses on variables of the same community. In order to quantify this locality, we introduce the following definition, inspired by the notion of modularity.

Definition 6.2. Given a subset of variables $S \subseteq N$, and a partition P of the variables $N = \cup_{i=1}^{c} P_i$, we define

$$\text{locality}(S, P) = \frac{\sum_{i=1}^{c} |P_i \cap S|^2}{|S|^2}$$

Intuitively, this definition expresses the fraction of edges between nodes of the same community with respect to the total number of edges, if we consider all possible edges between vertexes of S (including self-loops). Notice that, if all variables of S are split into r communities of same size (i.e. $|P_i \cap S| = |S|/r$, for $i = 1, \dots, r$, and $|P_i \cap S| = 0$, for $i = r+1, \dots, c$), then $\text{locality}(S, P) = 1/r$. Therefore, the inverse of locality measures the number of distinct communities involved in a set of variables (when all of these communities contain the same number of variables of S).

In Figure 6.9, we represent the evolution of the locality of the set of decided variables (between two conflicts), the set of assigned variables (between two conflicts), and the set of variables on the learnt clause, along the execution of the solver for the same SAT instance as in previous experiments, using as learnt clause the 1-UIP and the decisions-induced clauses. We only represent the moving average every 100 conflicts for 1-UIP clauses and 1000 conflicts for decisions-induced clauses. We observe that in both cases, the set of decided variables is very local, close to 1. This means that all variables decided between two conflicts almost always belong to the same community. This is not the case when we analyze the set of assigned variables. In this case, the average locality over the whole execution is 0.373 for the 1-UIP strategy and 0.281 for the other learning scheme. The locality of variables on the learnt clauses has a value in between the decision and the assigned variables. However, we clearly notice that, for the 1-UIP scheme this locality (0.782 on average) is much bigger than for decision-induced clauses (0.367 on average). We also notice that when the locality of the set of assigned variables increases (after a restart, for instance), the locality of variables on the learnt clauses also increases. Therefore, it is good for the solvers performance to increase the locality of assigned variables.

Finally, Norbert Manthey [Manthey, 2015] reports good results on the use of our generator to train a configurable SAT solver to improve its performance. He used the pseudo-industrial random SAT instances created by our model and used in the last SAT Race 2015 [Giráldez-Cru and Levy, 2015]. This family of instances contains 44 SAT formulas generated with $n = 2200$ variables, $m = 9086$ clauses, $c = 40$ communities and modularity $Q = 0.8$. He used this set of instances to train the SAT solver Riss 5.1.0 [Kahlert et al., 2015] (using the configuration tool SMAC [Hutter et al., 2011]). Riss is a very configurable SAT solver. Then, this configuration was used in Riss to solve the aggregated set of industrial instances used in all SAT Competitions from 2002 to 2015, and its performance was compared to the performance of the default configuration of this solver on the same set. Interestingly, the resulting configuration (i.e., from training this solver with our pseudo-industrial SAT instances) performed better than its default configuration. This suggests that our model captures a very important feature of industrial SAT problems which is, in fact, crucial in the solving process.

6.7 Conclusions

We present the Community Attachment model, a modularity-based generator which generates random k-CNF SAT instances of any desired modularity. Industrial problems are characterized by a high modularity. Therefore, our model can generate more realistic pseudo-industrial random formulas on demand. We validate the adequacy of this model checking that (i) the community structure of the resulting formulas is the expected, (ii) the phase transition point, dependent on the clause/variable ratio, is independent on the modularity, and (iii) the SAT solvers performances are consistent to the structure of the formulas generated by our model, i.e. SAT solvers *specialized* in industrial (random) problems perform better in *high modular* (*low modular*) instances.

Finally, we use our generator to study how the community structure is affected by some components of the solver. Namely, we study the variables branching heuristics and the clause learning mechanism. We observe that, for a given period of time, the solver tends to focus its decisions on variables of the same community, and learns clauses mostly relating variables of this community. We also show that restarts help to unassign variables belonging to communities where the solver is no longer focused on. Therefore, the community structure of the instance plays an important role in order to explain the success of these techniques, when they are used all together. On the contrary, we see that the solver has a worse performance when it uses instead a learning strategy that does not take into account such structure, as the learning of the *decision-induced clause*.

Chapter 7

Detecting relevant learnt clauses

7.1 Introduction

It has been empirically shown that the four main components of CDCL SAT solvers (i.e., conflict-driven clause learning, activity-based heuristics, lazy data structures used by the propagation engine, and random restarts) contribute to their success, but clause learning is the most important [Katebi et al., 2011]. However, there are some considerations about the utility or *relevance* of learnt clauses to be made: the relevance of learnt clauses is not the same, and it may vary along the search. This means that some clauses are useful and some others are useless, and also that a useful clause may become useless later.

Clause removal policies were initially proposed with the objective of saving memory and speed up propagations by the solver [Eén and Sörensson, 2003; Moskewicz et al., 2001]. However, the irruption of the SAT solver Glucose [Audemard and Simon, 2009], which implements a very aggressive clause removal strategy (more than 95% of the learnt clauses can be removed), demonstrates that this component is also an essential component of CDCL solvers. Notice that the preservation or removal of clauses may completely change the search, hence the performance of the solver may be dramatically altered. Therefore, the initial arguments for clause database managements (unit propagation speed and memory issues) do not completely hold anymore. The intriguing question on how to predict efficiently and effectively the relevance of new learnt clauses is still open.

The measure of the learnt clauses relevance proposed in Glucose (i.e., the Literal Block Distance or *LBD*) has been shown to be strongly related to the community structure of the initial formula [Newsham et al., 2014]. However, this last result was just a one-way observation of the solvers behavior: while LBD is related to the number of communities in a learnt clause, it was not possible until now to exploit this correlation the other way, i.e., by using the community structure to guide the search in a CDCL SAT solver.

In this chapter, we show that community structure can be used to detect relevant learnt clauses. In particular, we present a technique that transforms the original formula adding certain learnt clauses, and hence guiding the search. These clauses are directly obtained from the community structure of the formula. Notice that this causality is much stronger than the previous observed correlation. Although we present our technique as a preprocessor for readability, our contribution is to give empirical evidence that the community structure can be used to generate relevant clauses, which is much stronger than identifying them (e.g., LBD is used to rank existing clauses). This would suggest that the community structure may play an important role in clause deletion policies.

Our preprocessor uses the community structure to split the instance into disjoint subformulas, and augments it with the learnt clauses of solving pairs of such subformulas. Intuitively, these clauses could be related to the notion of *glue clauses* used in Glucose. Our inspiration comes from the observation that clause learning destroys the (original) community structure of the instance. We give empirical evidence about the commonly accepted claim that having more learnt clauses does not always speed up the solving process. However, we show that augmenting the instance with our technique works experimentally. This is the case in several sets of industrial benchmarks and several CDCL SAT solvers. Notice that augmenting a formula with learnt clauses is against the common idea of preprocessing, which generally tries to reduce the instance.

Questions addressed in this chapter:

Question 4. *How can we use the underlying structure of instances to implement more efficient CDCL SAT solving techniques?*

Related publications:

- Ansótegui, C., Giráldez-Cru, J., Levy, J., and Simon, L. (2015b). Using community structure to detect relevant learnt clauses. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, pages 238–254.

The rest of this chapter is structured as follows. We first review in Section 7.2 some observations about the effect of clause learning on the community structure of SAT instances, and we provide some insights on the relevance of clauses learnt by a CDCL solver in Section 7.3. In Section 7.4, we propose an algorithm that exploits the community structure to detect relevant clauses, and evaluate its performance in Section 7.5. We conclude in Section 7.6.

In Section 9.2.6, we discuss some extensions on the use of modularity-based relevant learnt clauses. Moreover, in Section 9.2.7, we also discuss some possible improvements on CDCL SAT solving techniques directly related to the underlying structure of industrial SAT benchmarks. Recall some related work in Section **??** about the underlying structure of real-world problems.

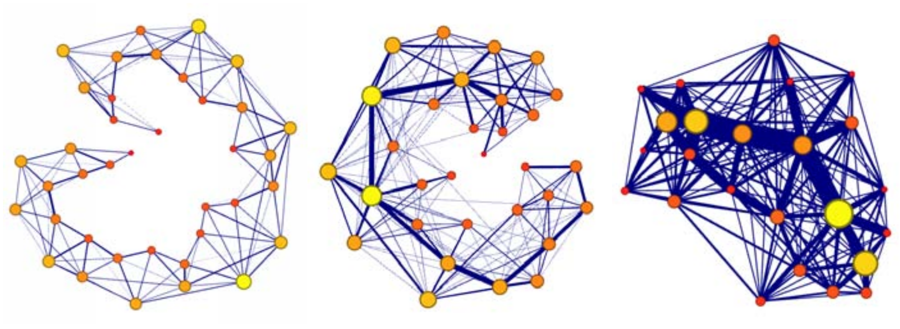


Figure 7.1: Graph of communities of the instance `ibm-2002-22r-k60`: original formula (left), solved formula considering *small* learnt clauses (center), and solved formula considering *small* and *medium-sized* learnt clauses (right). Nodes and edges are accordingly scaled by community size and weight, respectively.

7.2 Clause learning destroys the community structure

We have shown in Chapter 4 that industrial SAT instances have a very clear community structure, with modularity Q in the VIG higher to 0.7 in most of the cases. Recall that the maximum value of Q is 1. This means that we can find a partition of their variables into communities, such that clauses mainly constraint variables of the same community. However, this partition is destroyed by the addition of learnt clauses [Ansótegui et al., 2012] (as shown in Section 4.4). Let us review some important aspects of these results.

In order to represent how this (initial) community structure is destroyed by the effects of clause learning, we can use the graph of communities[1]. This graph is built as follows: all nodes of the VIG (variables) that belong to the same community are merged into a single node in the graph of communities, and weighted edges are updated accordingly.[2] In Figure 7.1 (*left*), we represent the graph of communities of the industrial formula `ibm-2002-22r-k60`. This instance has a modularity $Q = 0.91$ and 35 communities. The SAT solver Glucose [Audemard and Simon, 2009] (version 3.0) solved this formula keeping a total of 504964 learnt clauses. We can recompute the graph of communities after adding some of these learnt clauses to the original instance. In Figure 7.1 (*center* and *right*), we represent the graph of communities after adding *small* learnt clauses (up to 10 literals), and *medium-sized* learnt clauses (up to 50 literals), respectively.[3] In these graphs of communities, the node size is scaled according to the number of variables that belong to each community. Also, edges

[1]We cannot directly represent the VIG due to its large number of nodes (variables).

[2]The weight of the edge connecting communities A and B is the addition of the weights of the edges connecting one node from A and one node from B.

[3]As each clause of length l generates $\binom{l}{2}$ edges, it is hard to compute these graphs using *long* clauses.

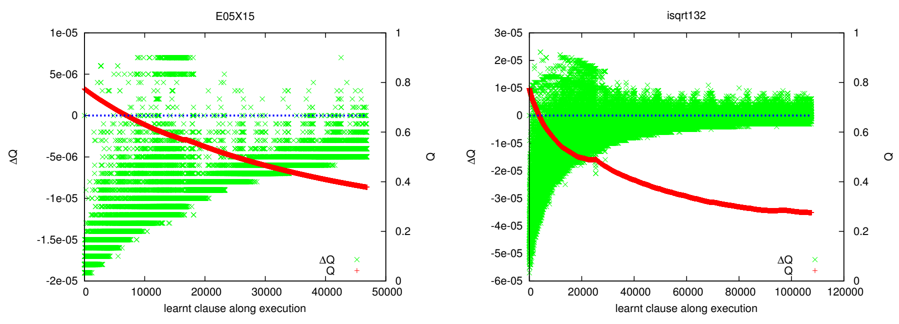


Figure 7.2: Impact of adding learnt clauses on modularity, in instances `E05X15` (left) and `isqrt1_32` (right). Each point (x, y), with y measured in the left Y axis, represents a clause learnt at instant x and increasing Q on y. We also represent the evolution of the modularity Q (using the right Y axis).

are scaled by their weights. Notice that edges weights are computed using the weights of the VIG (i.e., taking into account the length of the clauses). As it is stated by Ansótegui et al. [2012], the community structure is clear in all of these three graphs. However, as we consider more learnt clauses, we can observe two phenomena. First, the number of communities (number of nodes in the graph of communities) decreases. This means that variables that originally belonged to distinct communities are now grouped into the same community. Second, the weight of the inter-communities edges increases. Therefore, from the two previous effects, we observe that the solver prefers to learn clauses containing variables of distinct (original) communities (also stated in [Ansótegui et al., 2012]). This means that, in general, clause learning contributes to decrease the modularity.

A question now is: are there some learnt clauses that contribute to increase the modularity even when most of them do not? In order to answer this question, we can measure the increase of the modularity ΔQ that each learnt clause produces. Notice that ΔQ is positive when most of the new edges generated by such clause connect nodes (variables) of the same community. Otherwise, ΔQ is negative. After an extensive experimentation, we see that, in general, learnt clauses produce a very small decrease of the modularity (i.e., $\Delta Q < 0$, in most cases). In Figure 7.2, we represent this analysis for the industrial instances `E05X15` and `isqrt1_32`. Each point (x, y), with y measured in the left Y axis, represents a clause learnt at instant x and increasing Q on y. We also represent (using the right Y axis) the value of the modularity Q using the original partition of variables, along the execution. We can see that, even when some learnt clauses contribute to increase the value of Q, most of them do not (i.e., $\Delta Q < 0$), and thus Q tends to decrease. Due to space limitations, we only represent this analysis for these two benchmarks. However, we observed similar results in most industrial SAT instances studied. Therefore, we can conclude that, in general, learnt clauses contribute to destroy the (original) community

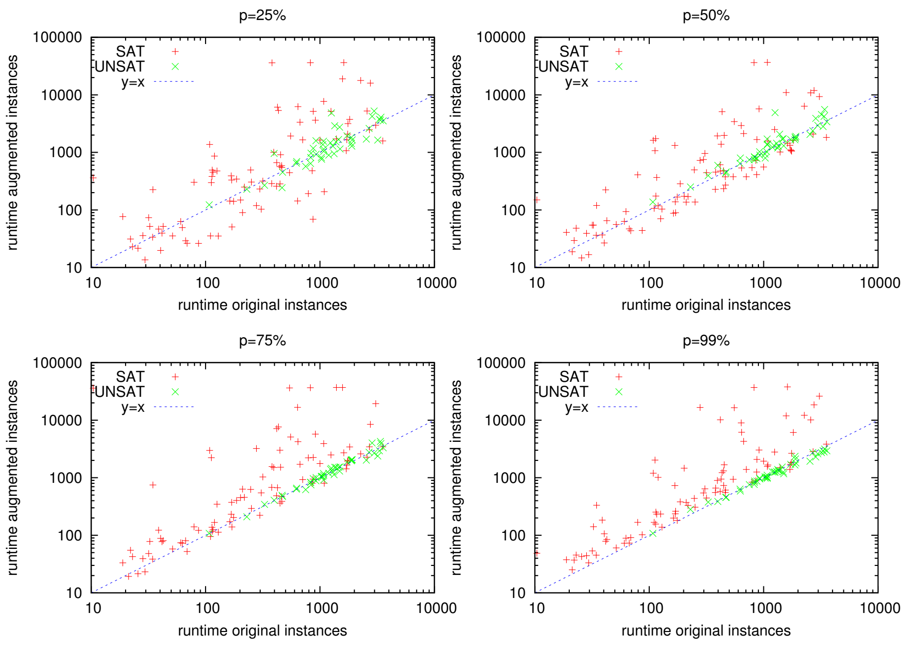


Figure 7.3: Scatter plots of solving original instances (first step) versus generating and solving formulas augmented with learnt clauses (second plus third steps), at $p = 25\%, 50\%, 75\%$ and 99%.

structure of the formula. It is not due to some particular clauses but rather a general phenomenon of the learning mechanism.

7.3 On the relevance of learnt clauses

In this section, we try to answer the following question: if we augment the original formula with a set of learnt clauses obtained from some CDCL solver, will this contribute to solve the formula faster? In order to answer this question, we first introduce the notion of *relevant* clauses.

Definition 7.1. Given a SAT solver S, a formula Γ, and a set of clauses φ, we say that φ is *relevant* for Γ and S, if φ is a logical consequence of Γ and $\Gamma \cup \varphi$ is *easier to solve* for S than Γ.

Notice that in this definition we neglect the time needed to compute φ. Obviously, previous definition is *informal*. In order to experimentally validate if a set of clauses is relevant, we have considered a significant set of industrial instances.

In a first experiment, we select the set of instances of the application track of the SAT Competition 2013 solved in less than one hour. Notice that this set

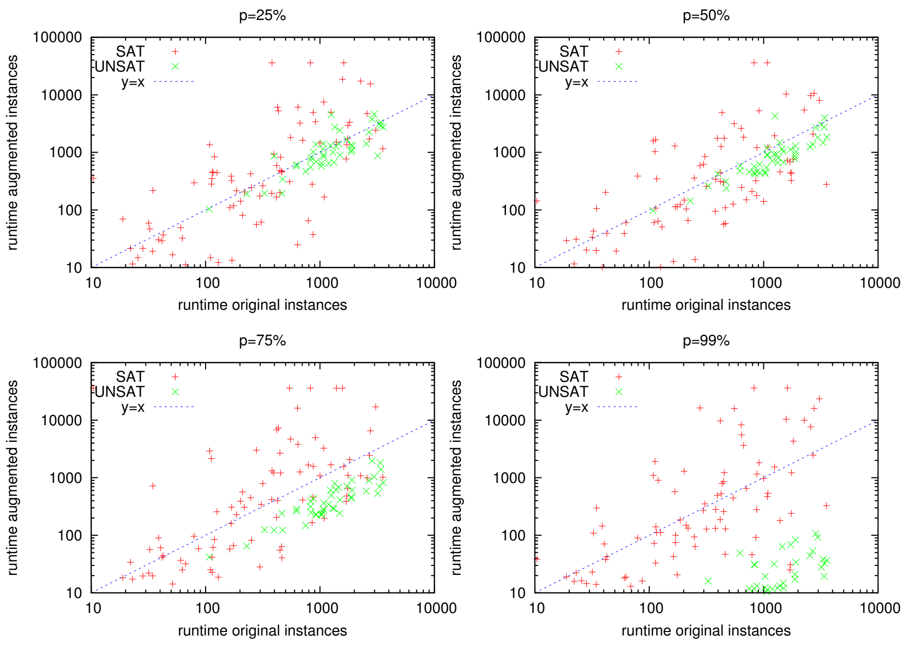


Figure 7.4: Scatter plots of solving original instances (first step) versus solving formulas augmented with learnt clauses (third step), at $p = 25\%, 50\%, 75\%$ and 99%.

contains both satisfiable and unsatisfiable instances. This experiment is divided in three steps. In all of them, we use the CDCL SAT solver MiniSAT [Eén and Sörensson, 2003] (version 2.2).

First step: we compute the number of conflicts c needed to solve the formula in an arbitrary run.

Second step: we repeat the same execution stopping the search after a certain number of conflicts $p \cdot c$ (where $0 < p < 1$), and we generate a new instance augmenting the original formula with the learnt clauses stored in the solver at that instant.

Third step: we solve the *augmented formula* generated in the previous step.

We could think that the third step is just the continuation of the second step due to a restart after $p \cdot c$ conflicts. But this is far from being true. First, CDCL SAT solvers do have more contextual information than learnt clauses, such as activity counters, status of restarts, etc. It is also interesting to notice that the phase caching scheme [Pipatsrisawat and Darwiche, 2007] is not saved in the third step: a learnt clause could have been responsible for a propagation, and thus responsible for setting the phase caching scheme when backtracking, but this learnt clause could have been removed. Second, the learnt clauses used to generate the augmented formula will be treated as original clauses in the third step, i.e., they cannot be removed by the solver.

Since we limited the number of conflicts to $p \cdot c$ in the second step, you could expect to need around $(1-p) \cdot c$ conflicts to complete the search in the third step. Surprisingly, in our experiments, this is true when the instance is unsatisfiable, but not when it is satisfiable. If the formula is satisfiable, the aggregated runtime of generating the augmented formula (second step) and solving it (third step) is usually higher than the runtime required to solve the original instance (first step).

Let us present these observations in detail. In Figure 7.3, we present the scatter plot of the runtime of solving the original formula (first step) versus generating and solving the augmented formula (second plus third steps), with $p = 25\%, 50\%, 75\%$ and 99%, and distinguishing SAT and UNSAT instances. In unsatisfiable instances, there is almost no difference (i.e., almost all points are on the diagonal). On the contrary, in satisfiable formulas the differences are much bigger (almost all points are far from the diagonal). Moreover, as we increase p, solving original instances is faster than generating and solving their corresponding augmented formulas (almost all points are above the diagonal). In Figure 7.4, we present the scatter plots of solving the original formula (first step) versus just solving the augmented formula (third step). Notice that in this case, we do not take into account the runtime needed to generate these augmented instances. However, even in this case, solving some satisfiable augmented instances takes more time than solving their corresponding original formulas.

We have observed that augmenting an instance with learnt clauses does not always contribute to make it easier, when the formula is satisfiable. Let us conjecture why. First, although adding learnt clauses helps to reduce the search space, there are other key components, such as the activity counters and the phase component caching. These heuristics are set to their *optimal*[4] values after a certain number of conflicts. The phase component caching may play a crucial role here, since the solver may use this information to keep the solution to a subproblem. Therefore, even if we have an oracle providing a set of learnt clauses, this does not mean that you will find a satisfying assignment faster. Also notice that the status of the activity counters cannot be reproduced from this set of learnt clauses. These counters depend on all clauses learnt during the execution of a solver, but some of them may have been removed, and therefore they do not belong to the provided set anymore. Second, in [Simon, 2014] it was shown that the runtime of solving unsatisfiable formulas is much more robust than for satisfiable ones. Shuffling the instance may have an important impact on satisfiable problems, but not on unsatisfiable ones: the effort to find the UNSAT answer (and the size its proof) are always of the same order. If we try to link our result to this work, we think a reasonable explanation is the following one. For satisfiable instances, the solver is mostly starting again the whole search, trying to *learn* the correct phase component caching values. In this case, adding learnt clauses can slightly help, but the overall process is dominated by the high discrepancy of CPU time needed for satisfiable problems when shuffling the instance. For unsatisfiable instances, this shows that the solver is *continuing* the

[4]In order to guide the search to a satisfying assignment.

Algorithm 6: Modularity-based SAT Instance Preprocessor (*modprep*)

```
  Input: SAT Instance $\Gamma$
  Output: SAT Instance $\Gamma'$
1 $\Gamma' := \Gamma$;
2 $C := communityStructure(\Gamma)$;
3 foreach pair $(c_i, c_j)$ of connected communities of $C$ do
4     Solver $s$;
5     $s.solve(c_i \cup c_j)$;
6     if $s == UNSAT$ then
7         return $\varnothing$;
8     $\Gamma' := \Gamma' \cup s.learntClauses$
9 return $\Gamma'$;
```

same proof.

Therefore, even when adding learnt clauses does not always help in satisfiable instances, is it possible to find a set of highly useful clauses that makes these formulas easier? In the next section, we will show that we can use the community structure to identify some clauses that are indeed relevant for those instances, i.e., they help to solve satisfiable instances faster.

7.4 Detecting relevant learnt clauses

Learnt clauses are redundant by definition, hence not strictly necessary. However, they can help to prevent exploring the same unsatisfiable subspaces during the search. Moreover, their role could be to *guide* the solver in building the UNSAT proof by resolution. It is essential here to see CDCL SAT solvers as a combination of backtrack search algorithms (where learnt clauses are used to prevent exploring the same search space) and resolution proof engines (where learnt clauses are used to derive new learnt clauses).

In the early versions of CDCL solvers, memory was an important issue [Eén and Sörensson, 2003; Moskewicz et al., 2001]. Therefore, some heuristics were proposed to remove useless clauses. Moreover, it is important to correctly manage the learnt clauses database in order to maintain a good unit propagation speed. More recently [Audemard and Simon, 2009], some clause removal policies have been proposed. They aggressively remove most of the learnt clauses (95% of the learnt clauses can be removed). The proposed strategy is now one of the standards in CDCL engines. Thus, this policy is not only about maintaining good unit propagation rates, but also to guide the solver to some *easier* proofs. In Glucose, it was proposed to consider the number of decision levels occurring in a learnt clause as a measure of its quality (this was called Literal Block Distance, *LBD*, lower is better). The idea was that literals propagated at the same decision level were tightly connected and may often be propagated again and again together. Clauses of LBD 2 (called *glue clauses*) are kept forever in Glucose. Recently [Newsham et al., 2014], it was shown that the LBD value was correlated to the number of communities of the clause. In this section, we show that community structure can be used to detect relevant learnt clauses.

In Algorithm 6, we propose a technique presented as a preprocessing step, called *Modularity-based SAT Instance Preprocessor* (*modprep*). It augments the original formula with some learnt clauses based on its community structure. This algorithm proceeds as follows. First, it computes the community structure of the original formula (line 2). Recall that each community represents a set of clauses of the original instance. Then, for each pair of connected communities[5], it creates a subformula containing both communities, and solves it (line 5). If this subformula is UNSAT, it returns the empty clause. Otherwise, the original instance is augmented with the clauses the solver learnt for solving such subformula (line 8). Finally, it returns the augmented instance.

Notice that the previous algorithm imposes a very strong condition, which is solving *all* subformulas between two connected communities and keeping *all* learnt clauses found in this process. This could be further refined. Moreover, this preprocessing step could be heuristically applied during the search in the flavor of inprocessing approaches [Järvisalo et al., 2012b].

Although we will show in next section that this approach works experimentally, we may wonder why these learnt clauses indeed improve the performance of the solver. It is worth noticing that, by construction, these learnt clauses are usually composed of at most 2 communities, and thus are clearly related to the notion of *glue clauses* aforementioned. In addition, as we showed in Section 7.2, learnt clauses contribute to destroy the original community structure. In order to do this, we first need to connect pairs of communities, then triples of communities, and so forth; since we learn clauses that connect all communities (i.e., the whole formula) and we derive the empty clause. Therefore, we do not want to erase the base of this process (clauses connecting pairs of communities). Notice that a solver not aware of the community structure may remove them, unless, as we do, these clauses are added in a preprocessing step as original clauses. i.e., the solver will not remove them.

In this work, we only consider learnt clauses connecting pairs of communities at the preprocessing step, and not triples or higher degrees. This is because the combinatorial space for pairs can be managed efficiently by the SAT solver. For bigger degrees, we would need some additional filtering criterion, or working on a parallel solver (discussed in Section 9.2.6).

7.5 Experimental evaluation

In this section, we present an experimental evaluation of the modularity-based preprocessor presented in the previous section. We use four representative CDCL SAT solvers: MiniSAT [Eén and Sörensson, 2003], Lingeling [Biere, 2014], Glucose [Audemard and Simon, 2009], and MiniSAT-blbd [Chen, 2014]. MiniSAT

[5]Two communities are connected if there exists at least one variable appearing in both of them.

	CS	SubF	Learnt
Average	12.6	78.0	11243.9
Median	4.3	21.8	512.0
Max	294.5	975.8	794950.0
Min	-	-	1.0

Table 7.1: Statistics about the execution of *modprep* on the set of 300 application instances of the SAT Competition 2011. *CS* stands for the runtime of computing the community structure (in seconds), *SubF* for the runtime of solving *all* subformulas of each instances, and *Learnt* for the number of learnt clauses obtained from solving all subformulas.

is one of the most popular CDCL SAT solvers, while the three others were the best ranked solvers in the application track of the last SAT Competition 2014: Lingeling won both the UNSAT and the SAT+UNSAT tracks, MiniSAT-blbd won the SAT track, and Glucose was the second classified in the UNSAT track.

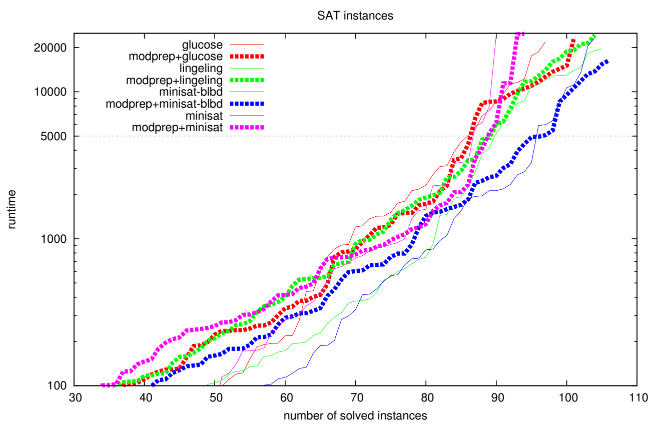
First, we evaluate how expensive is running the preprocessor *modprep* described in Algorithm 6 on a set of industrial SAT instances. We use the 300 application instances of the SAT Competition 2011. Notice that this algorithm can be split into two steps: i) partitioning the input formula into subformulas; and ii) solving them.

We compute the community structure using GFA (see Alg. 3 on Chapter 4). For this set of 300 application instances, this tool is able to correctly compute the community structure of 298 instances. This process is, in general, very fast. The average, median and maximum runtimes are respectively 12.6, 4.3 and 294.5 seconds.

Then, we solve all subformulas using MiniSAT. This step is performed on the 298 industrial formulas, with an average, median and maximum runtime of 78.0, 21.8 and 975.8 seconds, respectively. The average, median, maximum and minimum number of clauses that our preprocessor learnt is 11243.9, 512, 794950 and 1 clauses, respectively. We summarize these results on Table 7.1.

A natural question now is if the number of clauses learnt with this preprocessor depends on the solver used to solve such subformulas. We run again this step using Glucose instead that MiniSAT. Notice that Glucose uses a more aggressive clause removal policy. However, we observe that this solver learns, in general, a similar number of clauses as MiniSAT, and needs a similar runtime to solve these subformulas. This is because the input subformulas are, in general, very easy.

In the next experiment, we evaluate the performance of the mentioned solvers, with and without using the presented preprocessor (referred in the plots as <*solver*> and *modprep*+<*solver*>, respectively). In Figure 7.5, we represent the plots of this evaluation (solvers with and without using the *modprep* preprocessor) for the industrial instances of the SAT Competition 2011, distinguishing between satisfiable and unsatisfiable instances. We represent a cactus plot (i.e.,



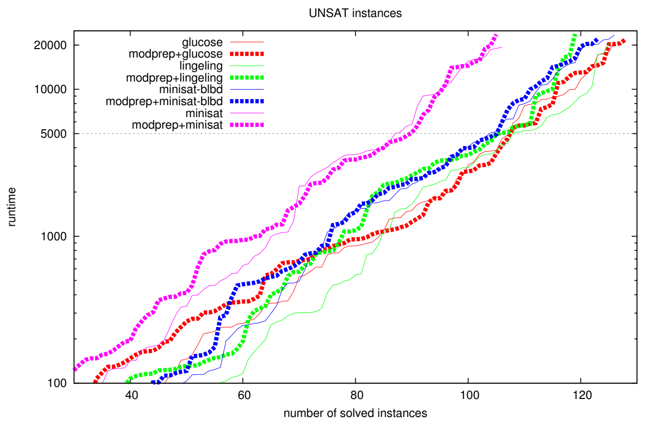


Figure 7.5: Evaluation of application instances of the SAT Competition 2011, distinguishing satisfiable instances (top) and unsatisfiable instances (bottom), for Glucose, Lingering, MiniSAT-blbd, and MiniSAT; with and without using our preprocessor.

the maximum runtime of solving a set of instances) with logarithmic Y axis. The timeout is set to 25000 seconds (the timeout usually used in competitions is 5000 seconds). We remark that the reported runtime when the preprocessor is used include the runtime of computing the community structure and the runtime of solving all subformulas. In Table 7.2 (see 2011), we report the number of solved instances with and without using the modprep algorithm for these two timeouts, and distinguishing between SAT and UNSAT instances.

We observe that using our preprocessor with MiniSAT, Glucose or MiniSAT-blbd improves their performance in satisfiable instances. Moreover, in unsatisfiable instances, Glucose also improves its performance. Interestingly, for this timeout of 25000 seconds, enhancing a solver with our preprocessor results into the best choice for solving satisfiable instances (using MiniSAT-blbd) and unsat-

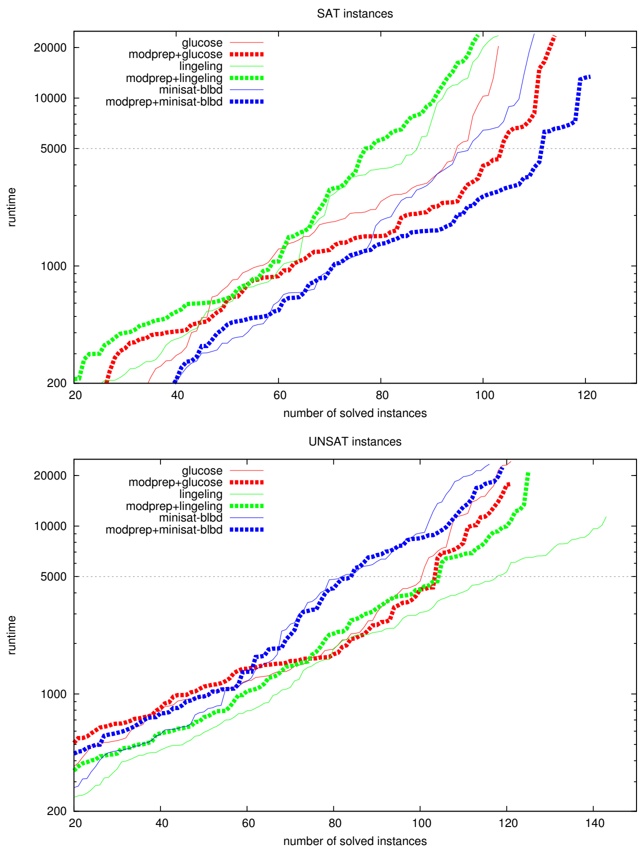


Figure 7.6: Evaluation of application instances of the SAT Competition 2014, distinguishing satisfiable instances (top) and unsatisfiable instances (bottom), for Glucose, Lingering, and MiniSAT-blbd; with and without using our preprocessor.

isfiable instances (using Glucose). More interestingly, the solver MiniSAT-blbd enhanced with our preprocessor also results into the best technique to solve satisfiable instances when a timeout of 5000 seconds is considered (similar to the timeout used in the competition). It is worth noting that, for very easy instances, the overhead of the preprocessor (i.e., computing the community structure and solving all subformulas) does not compensate.

We want to validate if the previous results also hold in a different set of industrial instances. We repeat the same experiment[6] for the set of 300 application instances of the SAT Competition 2014. In Figure 7.6, we represent the cactus plot of this experiment, distinguishing between satisfiable and unsatisfiable instances. We also report the number of solved instances by each SAT solver with

[6]Excluding MiniSAT.

category	SAT				UNSAT			
timeout	5000s		25000s		5000s		25000s	
solver	org	mdp	org	mdp	org	mdp	org	mdp
				SAT Competition 2011				
MiniSAT	86	*88*	90	*94*	87	*89*	*106*	105
Glucose	85	*86*	97	*101*	107	*107*	126	***128***
MiniSAT-blbd	95	***96***	104	***106***	103	*105*	*126*	123
Lingeling	*89*	89	*105*	104	***107***	105	*125*	119
				SAT Competition 2014				
Glucose	94	*103*	103	*114*	100	*103*	121	*121*
MiniSAT-blbd	97	***111***	110	***121***	81	*84*	116	*119*
Lingeling	*87*	77	*103*	99	***117***	104	***143***	125

Table 7.2: Number of solved instances by some SAT solvers without and with using the *modprep* algorithm (see columns *org* and *mdp*, respectively), evaluated on the set of 300 application instances of the SAT Competitions of 2011 and 2014, distinguishing between SAT and UNSAT formulas, and for a timeout of 5000 seconds (typically used in competitions) and a long timeout of 25000 seconds. For each pair of solver with/out modprep, we emphasize the best solution (in case of tie, we select the fastest one). Moreover, for each category (SAT/UNSAT) and timeout, we remark in bold the best solver strategy.

and without using the modprep algorithm in Table 7.2 (see 2014).

Again, we observe that Glucose and MiniSAT-blbd improve their performance in both satisfiable and unsatisfiable instances when the preprocessor is used. In fact, MiniSAT-blbd enhanced with our technique is the best solver in satisfiable instances. Interestingly, these solvers also improve their performance using a shorter timeout of 5000 seconds. For instance, in our cluster MiniSAT-blbd solves 97 SAT instances, while this solver enhanced with our preprocessor solves 111. This difference is significant in the context of competitions. Also, Glucose solves 194 SAT+UNSAT instances, while using our technique with this solver results into a total of 206 SAT+UNSAT solved instances. Again, this difference is significant. However, our preprocessor does not improve the performance of Lingeling.

It is important to remark that the SAT solver MiniSAT-blbd was the winner of the SAT Competition 2014 in the track of satisfiable instances. In the last experiment, we precisely used those instances. As we have seen, using our technique on this solver represents an improvement in its performance from 97 satisfiable instances solved to 111, i.e., a 14.43%, which is an overall improvement remarkable in the context of SAT competitions.

Finally, we want to check if a random partition of the formula would have the same effect as the partition provided by the community structure. For every instance, we compute a random partition of the formula with the same number of components as in the community structure. Also, we compute a sequential

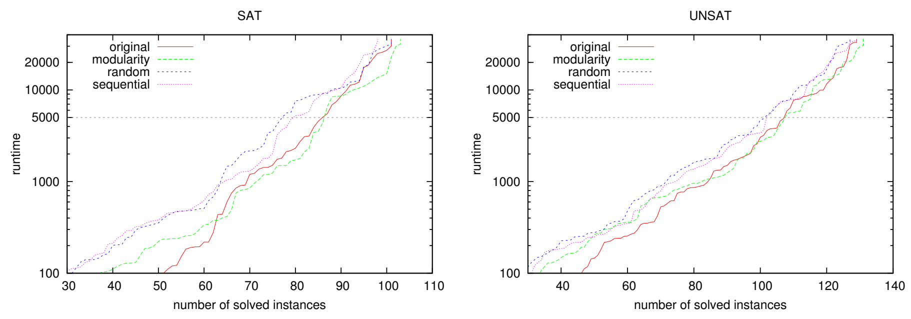


Figure 7.7: Evaluation of random and sequential partitions, distinguishing between satisfiable (left) and unsatisfiable formulas (right), using the set of industrial instances of the SAT Competition 2011, and solved by Glucose.

partition, where all variables of a component have sequential indexes. Then, we repeat all the experimentation with these random and sequential partitions. In Figure 7.7, we show the cactus plot of the results on the set of industrial instances of the SAT Competition 2011. As expected, none of these methods performs better than either solving the original instances or our proposed technique, which is the best strategy of these four.

Notice that in the previous experiment, the average, median, maximum and minimum number of clauses learnt by our preprocessor was respectively 4015.12, 28, 209085 and 0 clauses using the random partition, and 35360, 987, 951839 and 0 clauses using the sequential partition. Recall that using the community structure, our preprocessor learnt in average, median, maximum and minimum a total of 11243.9, 512, 794950 and 1 clauses, respectively. Therefore, with random components the number of learnt clauses is smaller than using the community structure, whereas with sequential components this number is bigger. This suggests that the partition used to create the subformulas is more important than the number of clauses learnt by the preprocessor.

7.6 Conclusions

We use the community structure of industrial SAT instances to identify a set of highly useful learnt clauses. We show that augmenting a SAT instance with clauses learnt by the solver during its execution does not always mean to make the instance easier, especially in the case of satisfiable instances. However, we also show that augmenting the formula with a set of clauses based on the community structure of the formula improves the performance of the solver in many cases. Interestingly, this improvement is especially relevant in satisfiable instances. In particular, we use the set of clauses learnt from solving all subformulas consisting in pairs of connected communities.

We implement this approach as a preprocessor, and we show that it works experimentally on some representative sets of industrial instances, especially in satisfiable formulas. Interestingly, the SAT solver MiniSAT-blbd, which was the winner of the satisfiable track of the last SAT Competition 2014, enhanced with our technique improves its performance. It is also the case of Glucose, which improves its performance when it is enhanced with our technique in both satisfiable and unsatisfiable instances. To the best of our knowledge, this is the first time that community structure has been used to improve the performance of a CDCL SAT solver.

Chapter 8

Classification of industrial SAT families

8.1 Introduction

In the last decades, there have been many works dedicated to improve the efficiency of SAT solving algorithms. One of the most promising approaches is the *portfolio* paradigm. This approach faces the Algorithm Selection Problem [Rice, 1976], which is the problem of choosing, using a prediction model, the best algorithm, from a predefined set, to solve a particular instance of a problem. Such prediction is usually performed using Machine Learning techniques.

In the context of SAT, portfolio solvers *classify* instances using prediction models that have been previously built in an offline process. To this purpose, a set of (predefined) features is computed for a representative set of instances, and the runtimes of a (predefined) set of algorithms are also computed. Then, these instances are grouped into *classes*, according to their features, and the best algorithm for each class is calculated. Finally, in the online step, once the given instance is classified, it is solved by the corresponding *best* solver of the class. Some examples of portfolio approaches to SAT solving are [Gomes and Selman, 2001; Kadioglu et al., 2010; Malitsky et al., 2011; Nikolic et al., 2009, 2013; Silverthorn and Miikkulainen, 2010; Xu et al., 2007, 2008].

The success of portfolio algorithms is due to the observation that different SAT solving techniques perform better on different SAT instances. This has resulted into a *specialization* of SAT solvers. Some examples of this specialization are: Conflict-Driven Clause Learning (CDCL) SAT solvers are the dominant technique for solving industrial SAT instances; Look-Ahead SAT solvers are specially efficient solving random SAT problems; Stochastic Local Search (SLS) SAT solvers exhibit a very good performance on satisfiable random k-SAT.

In this chapter, we show that the three notions of structure previously analyzed (i.e., the scale-free structure, the community structure, and the self-similar structure) can be used to effectively classify industrial SAT families. This classification can be useful for further SAT solvers specializations. In particular, there may exist different techniques exploiting the singularities of different industrial families. We show that using these structure features can result into a classification with similar effectiveness than using other sets of SAT features commonly used in portfolio approaches, independently of the classifier used. Interestingly, for some classifiers, using structure features even improves the performance of the classifier. We evaluate the performance of a portfolio SAT solver using these structure features, observing that its performance is almost unaffected. Finally, we measure the relevance of these structure features w.r.t. the other SAT features commonly used in portfolio approaches.

Questions addressed in this chapter:

Question 5. *What is the impact of the underlying structure on the classification of SAT instances?*

Related publications:

- Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2015a). On the classification of industrial SAT families. In *Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence (CCIA'15)*, pages 163–172.
- Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2016). Structure features for SAT instances classification. *Journal of Applied Logics.* Submitted.

The rest of the chapter proceeds as follows. In Section 8.2, we study how the structure features can be used to classify industrial SAT families. In Section 8.3, we analyze the performance of a portfolio SAT solver trained with this set of structure features w.r.t. other set of SAT features commonly used in portfolio approaches. In Section 8.4, we analyze the relevance of these structure features presented in this paper. Finally, we conclude in Section 8.5.

8.2 Classifying industrial SAT families

As we have shown in the previous chapters, most industrial SAT instances exhibit a power-law distribution in the number of variable occurrences [Ansótegui et al., 2009a], a clear community structure with high modularity [Ansótegui et al., 2012], and a fractal dimension that characterizes their self-similarity [Ansótegui et al., 2014]. Therefore, for each industrial SAT instance we compute the exponent α_v of a power-law distribution that best fits the number of variable occurrences, the modularity Q of its VIG, and the fractal dimensions d and d^b of its VIG and CVIG, respectively. Notice that the number of variable occurrences is exactly the degree of variable-nodes in the CVIG. In the case of the clause length, which corresponds to the degree of clause-nodes in the CVIG, it is not clear if the distribution that best fits these data is, in most of cases, a power-law. Also, the modularity Q^b of the CVIG could be computed, but most methods in

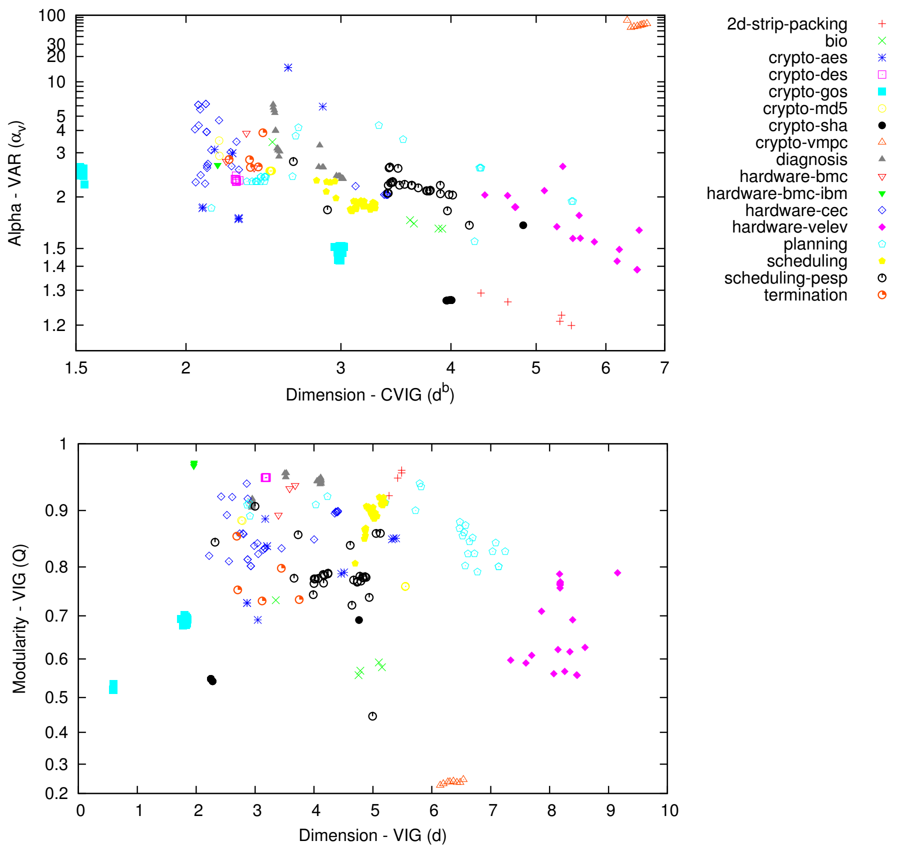


Figure 8.1: Distribution of families according to the exponent of the power-law distribution of variable occurrences (α_v), the fractal dimensions of the VIG and CVIG (d and d^b, respectively), and the modularity (Q). The heterogeneous families *software-bit-verif* (14 instances) and *software-bmc* (3 instances) are not plotted.

the literature are not adapted to be used in bi-partite graphs (they are either not accurate or not fast enough for these graphs).

In this analysis, we use the set of 300 industrial SAT instances of the SAT Competition 2013. These instances are grouped into 19 industrial families, according to their application domain: *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*. All instances are *industrial*, in the sense that they come from a real-world problem.

In a first experiment, we analyze if the classification of industrial SAT instances into families according to their domain corresponds to a classification of families by structure features. In Figure 8.1, we represent the relation between the scale-free structure (α_v), the community structure (Q) and the self-similar structure (d and d^b) for each industrial family. Each industrial SAT instance is represented by a different point, and each industrial family is characterized by a different symbol. In order to facilitate the visualization of this plot, we have omitted the industrial families *software-bit-verif* (14 instances) and *software-bmc* (3 instances), due to their heterogeneity. We have observed that most industrial SAT families are homogeneous, and many of them are clearly characterized by these structure features. For instance, the industrial family *hardware-velev* is characterized by an exponent α_v in the interval $[1.4, 3]$, high fractal dimensions, with $d > 4$ and $d^b > 7$, and a modularity Q in the interval $[0.5, 0.8]$.

Next, we want to determine if this reduced set of 4 structure features plus the clause/variable ratio m/n has similar results classifying industrial SAT families than other sets of SAT features commonly used in portfolio approaches. In particular, we use the set of SAT features used in the portfolio SAT solver SATzilla. The version of SATzilla submitted to the SAT Competition 2012 uses 127 features grouped in several categories: problem size, graphs (including statistics about the degree or clustering coefficient of nodes in the VIG and CVIG, among others), hardness (including DPLL, LP-based, SLS, clause learning and survey propagation statistics), balance, and timing[1] features. See [Xu et al., 2008] for a detailed description of these features. In this analysis, we consider the set of 115 resulting from removing the 12 timing features. We name the set of 5 structure features as *Structure*, and the set of 115 SATzilla features as *SATzilla* in our analysis.

In a second experiment, we build several classifiers using both the *Structure* and the *SATzilla* sets of features, in order to classify the industrial SAT family of a given instance. For each classifier, we measure the number of correctly classified instances (i.e., the number of SAT instances whose industrial family was correctly predicted). Notice that we know *a priori* the family each industrial SAT instance belongs to. Therefore, we use supervised machine learning techniques. In order to evaluate each classifier, we perform a k-folds cross-validation (i.e., dividing the set of instances into k folds such that each fold is evaluated using the classifier built with the other $k-1$ folds), with $k = 10$. Let us introduce the classifiers used in this experiment:

- *C4.5.* This algorithm [Quinlan, 1993] generates a decision tree to determine the category of each element. It is an improved extension of the earlier ID3 algorithm.

- *Random Forest (RF).* This model [Breiman, 2001] builds a combination, or forest, of random decision trees, such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.

[1] In SATzilla, some features represent the runtime needed to compute some categories of features (e.g., the runtime of computing graph features).

	Structure		*SATzilla*	
C4.5	259	(86.33%)	263	(87.67%)
RF	**274**	**(91.33%)**	**288**	**(96.00%)**
NB	254	(84.67%)	256	(85.33%)
MLR	247	(82.33%)	262	(87.33%)
LR	251	(83.67%)	**280**	**(93.33%)**
SMO	153	(51.00%)	241	(80.33%)
IBk	**275**	**(91.67%)**	264	(88.00%)
K*	**273**	**(91.00%)**	199	(66.33%)
JRip	246	(82.00%)	251	(83.67%)

Table 8.1: Number of correctly classified instances (and its percentage over the total set of instances in brackets), using the *Structure* features (α_v, Q, d and d^b plus the clause/variable ratio m/n) or the 115 *SATzilla* features, for some classifiers. In bold, we remark those classifiers whose effectiveness is higher than 90%.

- *Naïve Bayes (NB).* This algorithms [John and Langley, 1995] models a probability distribution with a Bayesian network, and handles continuous variables using statistical methods for non-parametric density estimations.
- *Multi-response Linear Regression (MLR).* This classifier [Frank et al., 1998] transforms the classification problem into a problem of function approximation, and this approximation is performed using regression methods.
- *Logistic Regression (LR).* This algorithm [le Cessie and van Houwelingen, 1992] builds and uses a multinomial logistic regression model with a ridge estimator.
- *Sequential Minimal Optimization (SMO).* This classifier [Platt, 1998] trains a Support Vector Machine (SVM), reducing this training problem into a series of smallest possible quadratic programming problems.
- *IBk.* This method [Aha and Kibler, 1991] implements the instance-based learning k-nearest neighbors algorithm, with a fixed value of k.
- *K*.* This model [Cleary and Trigg, 1995] uses the notion of entropy as a distance measure to determine the similarity between two instances.
- *JRip.* This algorithm [Cohen, 1995] implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER).

In Table 8.1, we represent the number of correctly classified instances by these classifiers, using the features sets *Structure* and *SATzilla*. We run each classifier with their default parameters values used in Weka [Hall et al., 2009]. In bold, we remark those classifiers whose effectiveness is higher than 90%, i.e., they correctly classify the industrial family of more than 90% of the 300 industrial SAT instances. As we observe, most of these classifiers have a very high effectiveness. In general, using the set of features *SATzilla* slightly outperforms the results of the set *Structure*. However, the differences between these two sets are very small. It is worth noting that while the set *SATzilla* contains a total of 115 features, the proposed set *Structure* only contains 5 features, and even so, the obtained classification and its effectiveness is similar. Interestingly, the classifiers K* and IBk improve their performance when using the set *Structure*.

Let us conjecture why this is the case. SATzilla characterizes the structure of SAT instances using a total of 14 graph features. However, these features represent *local* properties of its structure. For instance, the distribution of node degree is analyzed in SATzilla using some statistical features: maximum, average, median, standard deviation and minimum. Even so, these 5 features only characterize some *local* properties of the graph. We say they are *local* in the sense that none of them (used separately) speaks about a common behavior in the whole graph. On the other hand, in our metrics we use the exponent α_v, which characterizes the distribution of degrees, and thus it is a *global* property of the graph. Therefore, the previously mentioned 5 graph features used by SATzilla may be *implied* using the exponent α_v. Similarly, the clustering of variables is analyzed in SATzilla using the clustering coefficient, computing the values maximum, average, median, standard deviation and minimum for each node in the VIG. Again, these metrics only characterize a very *local* community structure. However, the modularity Q is a *global* metric of the graph, and therefore it gives a stronger information about this clustering.

In summary, SATzilla uses many (*local*) features to determine the structure of the formula, but even so, some global characteristics of such structure are not represented. On the other hand, we simply characterize it with 4 (*global*) graph features. Remark that we include the clause/variable ratio m/n in our set of features as a very simple metric about the hardness of the formula[2], while SATzilla analyzes it in a more exhaustive way using a total of 71 hardness features. Therefore, our characterization of their hardness is still weak.

In conclusion, we observe that the proposed set of structure features can be useful for classifying industrial SAT families of instances. This can beneficial for the specialization of SAT solvers, which may exploit the particularities of each industrial SAT family, when used in portfolio SAT solving approaches.

8.3 Evaluation of structure SAT features

In this section, we analyze the performance of a portfolio SAT solver when it is trained with the set of structure features presented in the previous section. In particular, we evaluate the performance of the solver ISAC [Kadioglu et al., 2010], and we compare it when it is trained with the set of SATzilla features.

[2]While the hardness of random k-CNF can be characterized using the clause/variable ratio m/n, this is not the case in industrial SAT instances. However, bigger industrial SAT formulas may be intuitively harder.

	Structure		SATzilla
runtime	VIG+CVIG	VIG	
minimum	0.07	**0.04**	11.71
median	4.9	**3.31**	49.24
average	21.65	**17.70**	170.43
stdev	36.87	**34.11**	362.27
maximum	287.12	**275.11**	3675.28

Table 8.2: Statistics results of the runtime (in seconds) of computing the set of SAT features over the set of 300 industrial SAT instances of the SAT Competition 2013, for the set *Structure*, using only graph features of the VIG, and using graph features of both VIG and CVIG; and for the set *SATzilla*. We remark in bold the fastest method.

For each set of features, the performance is evaluated with a 10-fold cross validation. This means that the set of 300 SAT instances is randomly divided into 10 disjoints subsets, or folds. For each fold, ISAC is trained using the remaining 9 folds (training set) to build a prediction model, and this is used it to predict the best (core) solver to solve each instance of this fold (testing set). We use all solvers submitted to the application track of the SAT Competition 2013 as core solvers. Finally, the reported runtime of solving a SAT instance is the runtime of computing its features plus the runtime of the (core) solver selected by ISAC.

Let us analyze first the cost of computing the set of structure features presented in this paper. Notice that this set contains features of both the VIG and the CVIG. Therefore, it is plausible to consider only computing the features of one of these graphs. In Table 8.2, we present some statistics of the runtime needed to compute these sets of features. For the case of the set *Structure*, we consider two cases: the features of the VIG, and the features of both VIG and CVIG. We observe that computing the structure features is, in general, more than one order of magnitude faster than computing the SATzilla features[3], even when we use our two graphs (VIG and CVIG). As expected, computing the structure features in only one graph (VIG) is faster than using both of them (VIG and CVIG).

In Figure 8.2, we represent the cactus plot of solving the 300 industrial SAT instances of the SAT Competition 2013 by ISAC when it is trained with the set of features *SATzilla* and some combinations of features from the set *Structure*. This plot represents the runtime (in seconds) needed to solve a certain number of instances, i.e., each point (x, y) represents that x instances were solved in at most y seconds (each of them). The combinations of structure features, named as $\mathit{Structure}_X$, uses VIG features (i.e., α_v, Q and d) when 'V' $\in X$, CVIG features (i.e., d^b) when 'C' $\in X$, and the clause/variable ratio when 'r' $\in X$. In Table 8.3, we also report some statistics about the results.

We observe that using the set of 115 *SATzilla* features is the best strategy,

[3]We use the tool provided in the solver SATzilla.

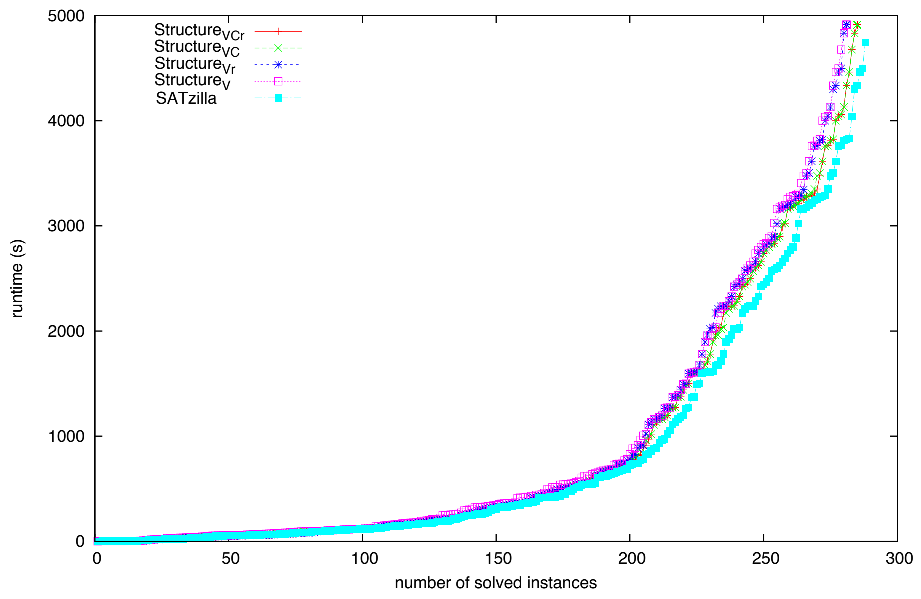


Figure 8.2: Cactus plot for the solver ISAC trained with the set of features *SATzilla* and some subsets of features from the set *Structure*. Specifically, *Structure*$_{VCr}$ uses α_v, Q, d, d^b and m/n; *Structure*$_{VC}$ uses α_v, Q, d and d^b; *Structure*$_{Vr}$ uses α_v, Q, d and m/n; and *Structure*$_V$ uses α_v, Q and d. Each point (x, y) in the plot represents that x instances were solved in at most y seconds (each of them).

solving a total of 288 instances. Surprisingly, using the *Structure* sets of features have very similar performances. In particular, using the features of both the VIG and CVIG (i.e., *Structure*$_{VC}$ and *Structure*$_{VCr}$) solves 285 instances in both cases, while using only VIG features (i.e., *Structure*$_V$ and *Structure*$_{Vr}$) solves 281 instances in both cases. Moreover, using the clause/variable ratio m/n does not affect the performance. Even though, the number of instances solved by these methods is very similar in all cases. Also, the runtime required for solving them is also very similar in all cases (see Table 8.3).

In conclusion, we show that computing structure SAT features is, in general, much faster than computing other sets of SAT features commonly used in portfolio approaches, as the set used by SATzilla. Also, we observe that the performance of a portfolio SAT solver is almost unaffected when, instead that training it with the 115 SATzilla features, we train it using just only 5 (or less) structure features, and the small differences between *SATzilla* and *Structure* are probably due to the richer study of the hardness performed by SATzilla.

runtime	Struct$_{VCr}$	Struct$_{Vr}$	Struct$_{VC}$	Struct$_{V}$	SATzilla
minimum	0.26	0.26	0.26	0.26	0.26
median	272.62	268.10	262.02	310.27	261.84
average	874.59	863.32	876.84	881.15	831.61
stdev	1199.59	1197.70	1200.11	1207.83	1143.55
maximum	4913.23	4913.23	4913.23	4913.23	4745.21
#solved	285	281	285	281	**288**

Table 8.3: Statistics about the runtime required to solve the set of 300 instances of the SAT Competition 2013 by ISAC trained with different set of features.

8.4 Relevance of structure SAT features

In this section, we analyze the *relevance* of all features used in the previous experiments. They are the 115 features of SATzilla plus the 4 structure features previously introduced We want to evaluate their relevance independently of the classification method used (e.g, in a portfolio SAT solver). For this purpose we use a filtering method for feature selection, a classical method of Machine Learning.

In the minimal-Redundancy-Maximum-Relevance method (mRMR) [Peng et al., 2005], we try to select a subset of features mutually as dissimilar to each other as possible (minimal *redundancy*), but marginally as similar to the classification variable as possible (maximal *relevance*). This is achieved finding the subset S of features that maximizes:

$$\max_S \left(\sum_{j\in S} I(x_j, c) - \frac{1}{M-1} \sum_{\substack{i,j\in S\\ i<j}} I(x_i, x_j) \right) \tag{8.1}$$

where c is the classification variable, M is the number of features, and $I(x_i, x_j)$ measures the mutual information between features x_i and x_j. Formally, the mutual information between two random variables x_i and x_j is defined as:

$$I(x_i, x_j) = \int\int p(x_i, x_j) \log \frac{p(x_i, x_j)}{p(x_i)p(x_j)} dx_i dx_j$$

Computing mutual information is based on estimating the probability distributions $p(x_i)$, $p(x_j)$ and $p(x_i, x_j)$. These distributions can be either discretized or estimated by density functions methods [Rodriguez-Lujan et al., 2010]. The first term of Eq. 8.1 computes the relevance and the second term the redundancy. For real applications, this objective function is difficult to compute exactly. Peng et al. [2005] propose to use a greedy or gradient algorithm that, starting with $S = \emptyset$, proceeds adding the feature that most increases the objective function at each step.

Rodriguez-Lujan et al. [2010] write the previous objective function as a pseudo-Boolean quadratic function, and use a parameter $\alpha \in [0, 1]$ to regulate

rank	feature (and description)	category	relevance
1	*SP-bias-mean*: mean of confidence of survey propagation (the higher of $P(true)/P(false)$ or $P(false)/P(true)$) for each variable	Survey Prop	0.6359
2	d: fractal dimension for VIG	***Structure***	0.5541
3	*POSNEG-RATIO-VAR-max*: max of ratio of positive to negative occurrences of each variable	Balance	0.5231
4	*POSNEG-RATIO-CLAUSE-coeff-variation*: variation coefficient of ratio of positive to negative literals in each clause	Balance	0.5171
5	d^b: fractal dimension for CVIG	***Structure***	0.4769
6	*SP-unconstraint-coeff-variation*: variation coefficient of probability that a variable is unconstrained in survey propagation	Survey Prop	0.4250
7	*POSNEG-RATIO-VAR-mean*: mean of ratio of positive to negative occurrences of each variable	Balance	0.4168
	...	...	
22	$\log \alpha_v$: powerlaw exponent	***Structure***	0.2558
	...	...	
41	Q: modularity (for VIG)	***Structure***	0.1844
	...	...	

Table 8.4: Relevance of SAT features to classify industrial SAT families.

the relative weight between relevance and redundancy:

$$\min_{X\in\{0,1\}^M} \left\{ \frac{1}{2}(1-\alpha)X^T Q X - \alpha F^T X \right\} \tag{8.2}$$

where $q_{ij} = I(x_i, x_j)$ and $f_j = I(x_j, c)$. In general, values of α closer to 1 result into smaller sets of selected features optimizing the objective function. They use the Nyström method to approximate the optimum of this function.

In our experiments, we compute the relevance $I(x_j, c)$ of all the 119 features x_j (i.e., the 115 SATzilla features plus the 4 structure features) as proposed in [Rodriguez-Lujan et al., 2010]. Recall that the classification variable c corresponds in our case to the industrial SAT family each instance belongs to. The relevance of a set of features is the matrix F according to Eq. 8.2. In Table 8.4, we report the most relevant features, as well as the ranking of the structure features, with their correspondent relevance value (higher is better). We observe that 2 of our 4 structure features are between the 5 most relevant features. In particular, they are the fractal dimension for the VIG d (ranked in the second position), and the fractal dimension for the CVIG (ranked in the fifth position).

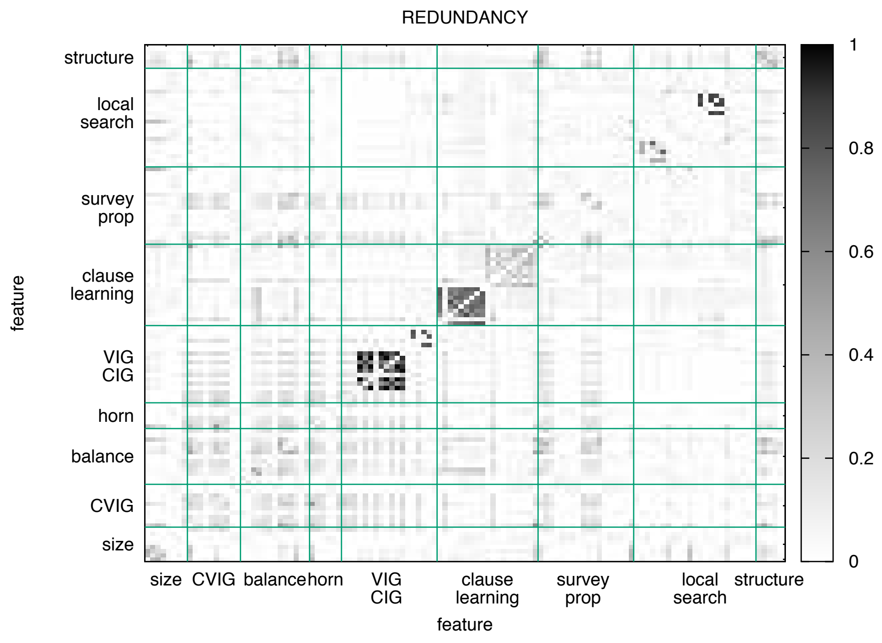


Figure 8.3: Redundancy between pairs of features. Each point (x, y) represents the redundancy between features x and y, according to the grey scale. Features are grouped by SATzilla categories, and *Structure* category contains α_v, d, d^b and Q (in this order).

The exponent α_v appears in the position 22, and the modularity Q in the position 41. Remark that we use $\log \alpha_v$ (instead of using directly α_v) because some industrial families are characterized by an exponent α_v some orders of magnitude higher than the rest. Recall that we are considering a total of 119 SAT features. Therefore, structure features are very relevant features.

It is also interesting to remark that the 5 most relevant SATzilla features belong to the categories *survey propagation* and *balance*. Survey propagation is a technique, based on works of spin glasses and statistical physics, to estimate the probability that a Boolean variable has a certain value in all satisfying assignments. The category *Balance* refers to the ratio of positive and negative polarities of literals appearing in the formula. For instance, a totally unbalanced formula only contains variables appearing with the same polarity. Notice that this kind of formulas is trivially satisfiable by the pure literal rule. Intuitively, balanced formulas are harder to solve. Therefore, the most relevant SATzilla features are related to the *hardness* of the instance.

Finally, we analyze the redundancy between pairs of features. This is the matrix Q according to Eq. 8.2. We represent the results in Fig. 8.3 as a heat map, i.e., the redundancy between features x_i and x_j is represented in the point

(i, j) (and (j, i)) according to the grey scale indicated in the figure. The values of this matrix are normalized between 0 and 1. Features are grouped into SATzilla categories: problem *size*, *CVIG* node degree statistics, *balance*, proximity to *horn* formula, *VIG* node degree and diameter statistics and *CVIG* node degree and clustering coefficient statistics, *clause learning* statistics (based on 2 seconds of running Zchaff_rand), *survey propagation*, and *local search* statistics (based on 2 seconds of running each of SAPS and GSAT). Finally, our set of *structure* features includes α_v, d, d^b and Q.

We observe that the most redundant pairs of features are found within each category. For instances, many VIG/CIG features are very redundant with each other. In the case of our proposed 4 *structure* features, we observe the same behavior. In particular, the pairs (α_v, Q) and (d, d^b) are the most redundant in this category. However, these redundancies have a normalized value around 0.3, i.e., they are not very redundant. Interestingly, the most redundant categories w.r.t. *structure* features are the categories *balance* and *survey propagation*, which are also the most relevant features.

8.5 Conclusions

We use the set of four structure features (i.e., the exponent α_v of the power-law distribution that best fits the number of variable occurrences, the modularity Q of the VIG, and the fractal dimension d and d^b of the VIG and CVIG), plus the clause/variable ratio m/n, to classify industrial SAT instances into families. We show that this classification has an effectiveness similar to the one obtained with other sets of SAT features commonly used in portfolio approaches, as the set used by SATzilla.

Also, we observe that computing this set of structure features is, in general, more than one order of magnitude faster than computing SATzilla features. We evaluate the performance of the portfolio SAT solver ISAC trained with these two sets of features (i.e., *Structure* and *SATzilla*), and we observe that the performance of this solver is very similar in both cases.

Finally, we analyze the relevance of the features from both sets, and we show that structure features are very relevant, as other hardness features (as *Survey Propagation* or *balance* statistics) computed by SATzilla.

Part III

Discussion and conclusions

Chapter 9

Conclusions and Future Work

9.1 Conclusions

Originally, the Boolean Satisfiability problem (**SAT**) was predominantly considered a theoretical problem, used to prove the NP-completeness of other problems and show their intractability. However, the irruption of modern SAT solving techniques, which efficiently solve many SAT instances, has dramatically changed the situation, and therefore SAT has become an essential component of many real-world applications. These techniques include clause learning, conflict-based activity heuristics, non-chronological backtracking or random restarts. In general, this set of techniques is usually named as *Conflict-Driven Clause Learning* (or *CDCL*) SAT solvers. The intuition to explain the success of CDCL solvers solving application SAT benchmarks is that some *hidden* structure of these problems is exploited by these techniques. However, besides the enormous progress in SAT solving technology, there are still few works trying to understand the reasons of the power and efficiency of these techniques on solving these real-world problems.

In this thesis, we address the problem of (better) characterizing the underlying structure of SAT instances, with especial emphasis on application benchmarks. Our main goal is to give better insights on the success of CDCL techniques solving industrial instances, with the aim of improving them.

Our inspiration comes from some works in complex networks, where the structure of real-world graphs has been analyzed. It has been shown that the classical random graph model (i.e., Erdös-Rényi model) cannot be used to represent this kind of networks. On the contrary, some important structure features shared by the majority of real-world networks have been studied, and some models to produce random graphs with these features have been proposed. Some of these features are the scale-free structure, the community structure, or the self-similar structure, among others.

In our approach, we represent SAT instances as graphs (using the VIG and CVIG models; see Chapter 2 for more details), and some (graph) features are studied. This is the approach used by Ansótegui et al. [2009a] to analyze the scale-free structure of industrial SAT instance (see Section 3.3 for a brief introduction of this work).

The first two challenges of our work are summarized as:

Question 1. *What is the underlying structure of industrial SAT instances?*
Question 2. *How is this structure affected by CDCL SAT solving techniques?*

In our analysis of the underlying structure of SAT instances, we focus on the *community structure* and the *self-similar structure*. These studies (partially) cover the two goals stated in the previous questions. In a graph with clear community structure (i.e., with high *modularity*), nodes can be grouped into communities such that most of the edges connect variables of the same community. In a graph with the self-similar pattern (i.e., with *fractal dimension*), the shape of the graph is the same at different scales (i.e., grouping set of nodes into a single one).

We analyze the **community structure** of SAT instances, and we observe that most industrial SAT formulas are characterized by a very clear community structure, with a modularity greater than 0.7 in most of the cases[1]. On the contrary, the modularity of random SAT instances is very low, i.e., they do not have community structure (as expected). Moreover, we analyze the effect of adding to the original instance the clauses learnt by the solver during the search. In this case, the modularity slightly decreases, but the community structure is still clear. This means that we can find a partition of the variables into communities such that the new augmented set of clauses mostly contain variables of the same community. However, we also observe that this partition differs from the original partition that was computed for the original instances. Therefore, we conclude that clauses learning predominantly destroys the original partition of the formula.

We also study the **self-similar structure** of SAT instances, and we show that most industrial SAT formulas are characterized by self-similar pattern, with a fractal dimension ranging between 2 and 4 in most of the cases[2]. On the other hand, random SAT instances are not self-similar (as expected). Again, we also analyze the effect of adding learnt clauses to the original formula. The purpose of this analysis is to *measure* how distant the parts connected by new learnt clauses are. Notice that the previous analysis of the community structure states that the original partition of the formula is destroyed by the effect of clause learning. However, this can happen connecting either close or distant communities. Our analysis shows that learnt clauses usually connect close parts of the formula (many of them were already connected before). This means that clause learning acts *locally*.

Beyond the previous analysis on the underlying structure of SAT instances, we address three applications directly related to the structure of the formulas.

[1]The optimal value of the modularity ranges between 0 and 1.
[2]These values are also very common in real-world networks.

First, we face the problem of generating realistic pseudo-industrial random instances. This problem was stated as one of the most important 10 challenges to be achieved in the following years [Selman et al., 1996]. The need of such generators comes from the fact that the number of real-world instances is limited, and moreover, the testing and debugging of new techniques using these benchmarks is often too expensive. For this reason, having a random model that reproduces the computational properties of these real-world problems may be beneficial for the SAT community. Second, the applicability of SAT to many real-world domains requires a continuous research and progress in SAT solving techniques. For this reason, we address the problem of improving the state-of-the-art CDCL techniques using the previously analyzed structure features. Finally, we face the problem of the classification of industrial SAT instances. This problem is interesting for portfolio approaches, where the best solver to solve a *class* of problems is predicted. Therefore, the effectiveness in the classification of instances may affect the performance of these techniques.

These three application can be summarized in the following questions:

Question 3. *How can we generate more realistic pseudo-industrial random SAT problems?*
Question 4. *How can we use the underlying structure of instances to implement more efficient CDCL SAT solving techniques?*
Question 5. *What is the impact of the underlying structure on the classification of SAT instances?*

First, we propose a new model of generation of random SAT instances, called **Community Attachment**, based on the notion of modularity. For a high value of modularity, we realistically model pseudo-industrial random SAT instances. On the contrary, a low value of modularity produces instances very similar to the classical random model. We show that the performance of SAT solvers is consistent with the expected properties of the generated formulas, i.e., SAT solvers *specialized* in industrial benchmarks perform better on high modular instances than solvers *specialized* in random formulas, and vice versa. We also prove that the phase transition point of this model is independent of the modularity, i.e., it is at the same point than the phase transition point of the classical random model.

Second, we identify a set of highly useful learnt clauses, and we show that augmenting the formulas with these clauses results into an overall improvement of several CDCL solvers, especially in satisfiable instances. These clauses are related to the community structure of the formula, and they can be computed in a fast preprocessing step. In particular, we propose a method, called **modprep**, that split the original formula into disjoint subformulas, and solve every pair of connected subformulas,[3] keeping all the learnt clauses produced in this process.

Finally, we use the underlying structure (i.e., the scale-free structure, the community structure, and the self-similar structure) to address the problem of **classifying** SAT instances. In particular, we analyze the effectiveness of these features when used in some classifiers, and we compare it to the effectiveness

[3]Subformulas that share, at least, a variable.

of other sets of features commonly used in portfolio SAT approaches, as the set used by SATzilla (which contains a total of 115 features). We observe that the effectiveness of both set of features is very similar (independently of the classifier), and the performance of the portfolio SAT solver ISAC is almost unaffected when trained with these two sets. Finally, we measure the relevance of the structure features, and we show that they are among the most relevant SAT features.

This thesis is hold by the following publications:

1. Ansótegui, C., Giráldez-Cru, J., and Levy, J. (2012). The community structure of SAT formulas. In *Proceedings of the 15st International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 410–423.
2. Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2014). The fractal dimension of SAT formulas. In *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14)*, pages 107–121.
3. Giráldez-Cru, J. and Levy, J. (2015). A modularity-based random SAT instances generator. In *Proceedings of the 24st International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1952–1958.
4. Giráldez-Cru, J. and Levy, J. (2016). Generating SAT instances with community structure. *Artificial Intelligence.* Accepted with revisions.
5. Ansótegui, C., Giráldez-Cru, J., Levy, J., and Simon, L. (2015b). Using community structure to detect relevant learnt clauses. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, pages 238–254.
6. Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2015a). On the classification of industrial SAT families. In *Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence (CCIA'15)*, pages 163–172.
7. Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2016). Structure features for SAT instances classification. *Journal of Applied Logics.* Submitted.

9.2 Future work

9.2.1 Graph representation of SAT instances

The graph models used in this thesis to represent SAT instance are the Variable Incidence Graph (VIG) and the Clause-Variable Incidence Graph (CVIG). See Chapter 2 for more details. These models represent Boolean variables, but they do not represent their corresponding literals, i.e., the sign of these variables in the clauses they occur. Therefore, multiple formulas may be modeled by the same graph. For instance, if we modify an instance negating some of its literals, the resulting VIG and CVIG of this new formula are exactly the same than the VIG and CVIG of the original one.

The main consequence is that we cannot distinguish between hard and easy formulas (e.g., a trivially satisfiable instance by pure literal elimination). Obviously, this may have some impact on the applications in which the graph model is used, *somehow*, in a solving technique. An example is presented in Chapter 7, where the VIG model is used to partition a SAT instance into disjoint subformulas, and then a community-based solving technique is performed. Therefore, it seems natural to analyze other graph representations of SAT formulas. In what follows, we mention some potential models that may improve the ones used in this dissertation.

The **Literal Incidence Graph**. The simplest model to represent the sign of variables is a graph with vertexes the set of literals, and edges between pairs of literals that appear in the same clause. As in the VIG, edges can be weighted to consider the length of the clauses. Moreover, there exists an edge between each pair of literals x and $\neg x$, with weight equal to 1. Notice that any SAT instance is equisatisfiable if adding the clauses $(x_i \vee \neg x_i)$, for $1 \leq i \leq n$, where n is the number of variables. This is graph is as easy to compute as the VIG. However, it offers a more powerful information about the instance. To the best of our knowledge, there is no work analyzing or using this model yet.

The **(extended) Factor Graph**. This is a model very similar to the CVIG, but with two important differences: (i) edges do not have weight, and (ii) there exist two types of edges to represent the two possible signs of each variable occurring in each clause. This model has been used, for instance, in Survey Propagation [Braunstein et al., 2005]. This method, based on some works on spin glasses, computes the probability that a variable has a certain value in all satisfying assignments using a message passing method through the edges of this graph. The natural extension of this model can consider weighted edges in order to keep some information of the length of the clauses (without traversing the graph). Again, this model is as simple to compute as the CVIG. However, its particularities (e.g., two types of edges) make that some methods may require an *ad-hoc* adaptation to be used on this model. For instance, in order to compute the community structure of this graph, an algorithm, adapted for bi-partite graphs, should also consider the two types of edges. In some works, this model is also named as *Clause Variable Graph* [Kullmann, 2009].

The **Resolution-based Graph**. In this graph, clauses are nodes, and there is an edge between two clauses if the resolution rule can be applied between these two clauses (i.e., there exists a literal that appears in both clauses with opposite sign) and the resolvent clause is not a tautology (i.e., it does not contain a variable and its negation). The weight of the edges is proportional to the length of the resolvent clause (repeated literals are not considered). This model was introduced in [Yates et al., 1970], and adapted in [Neves et al., 2015], where it was used to improve the performance of a MaxSAT solver using community detection algorithms. However, in the context of SAT instances the situation is more complex. Notice that it cannot be known *a priori* either the length of the resolvent or whether it is a tautology. Therefore, the exact computation of this model depends on the number of clauses and their length. This makes this model inefficient for very large instances (as real-world SAT formulas are) in applications where the solving time is an important issue. A preliminary computation of this model on some industrial instances used in our experimentation resulted into a runtime even higher than the runtime needed to solve the formula. On the contrary, some approximation can be made. One possibility would be to assign an edge between any two clauses that can apply resolution, with a weight proportional to the sum of the length of these two clauses minus 2 (the two resolving literals). This way, neither repeated literals nor tautologies are filtered. However, this approximation can be computed more efficiently, since it only depends on the number of occurrences of each literal. Whether this approximation is accurate and useful for further applications has not been studied yet, to the best of our knowledge.

9.2.2 Relation between structure features

In Chapters 4 and 5, we have shown that (most) industrial SAT instances are characterized by a clear community structure and a self-similar pattern [Ansótegui et al., 2014; Ansótegui et al., 2012]. Moreover, the previous work of Ansótegui et al. [2009a] shows that these instances also have scale-free structure (see Section 3.3 for more details). We consider that these features are three very important components of the underlying structure of real-world SAT formulas. However, in some application presented in this thesis, we focus our contribution in the use of the community structure.

A natural question is: what is the relation between these three features? In particular, a graph with scale-free topology and self-similar pattern implies a clear community structure? To the best of our knowledge, there is no work analyzing the relations between these structure features. Obviously, such a work would be an interesting contribution for the community of complex networks, and it would have impact in other research areas, as the SAT community.

9.2.3 Overlapping communities

In our analysis of the community structure (see Chapter 4 for more details), we only consider partitions of the graphs into *disjoint* communities. This is the traditional approach in the field of complex networks. However, a number of problems requires *overlapped communities*. In this case, a node of the graph can belong to several communities at the same time. This can be the case, for instance, of a social network whose communities represent family, friend or colleagues. See [Xie et al., 2013] for a survey on overlapping community detection methods.

In the case of SAT instances, similar questions also arise. Our experience tells us that *many* variables of an industrial SAT formula, even when they belong to a certain community (i.e., they mostly occur in clauses with other variables of the same community), they also occur with *many* variables of other communities. In other words, there is a number of variables in the *fringe* of communities. Some preliminary observations show that this fringe may contain more than 50% of the variables of some industrial SAT instances.

An interesting direction to investigate would be the ways of computing overlapping communities in the context of SAT instances. For instance: which variables can belong to an overlapped community? Or what do these communities represent? Also, some method handling these overlapping communities may improve the performance of the state-of-the-art solutions that use the community structure to solve SAT or MaxSAT problems.

9.2.4 Other structure features

In the context of complex networks, other structure features have been studied. One example is the notion of *similarity* [Papadopoulos et al., 2012], which complements the notion of popularity (i.e., preferential attachment) to explain the growing of real-world networks. Another example is the notion of *conductance* [Kannan et al., 2004], that measures how well-knit a graph is, or the notion of *coverage*, that measures the fraction of internal edges w.r.t. a partition. See [da F. Costa et al., 2005] for an extended survey on the characterization of complex networks, and [Almeida et al., 2011] for a review on graph clustering metrics.

Interestingly, these advances in complex networks may result into a better understanding about the *hidden* structure of industrial SAT instances, and possibly better explain the success of CDCL techniques solving these problems. This can serve to further improvements of the state-of-the-art CDCL SAT solving techniques.

9.2.5 Pseudo-industrial SAT instances generator

The future work on the generation of realistic pseudo-industrial SAT instances can be tackled from two perspectives.

First, some improvements of the *Community Attachment* can be considered. This model (see Chapter 6 for more details) forces some features of the resulting SAT instance to be as much regular as possible. In particular, all clauses have exactly the same number of literals (i.e., k literals), all communities approximately have the same number of variables (i.e., $\lfloor n/c \rfloor$ or $\lfloor n/c \rfloor + 1$), and all variables approximately have the same number of occurrences. This allows us to study the real impact of certain SAT solving techniques on the community structure without any undesired secondary effect.

Real application benchmarks are characterized by a certain variability in the clause size, community size, and number of variable occurrences. Therefore, some natural extensions of the Community Attachment model may consider these cases. A possibility would be to assign a distinct probability to each variable, as it is described in [Ansótegui et al., 2009b]. This would result into random instances with scale-free structure and high modularity, as observed in real-world instances.

Another possibility is other models of generation that can be proposed. For instance, many complex networks have heterogenous degree distributions and strong clustering, and random geometric graphs in hyperbolic spaces can be used to adequately model them. Aldecoa et al. [2015] propose an hyperbolic graph generator to produces these features. This idea could be easily adapted to the context of SAT instances.

9.2.6 Relevant learnt clauses

Our work on the detection of relevant learnt clauses (see Chapter 7) can be extended in a number of ways.

An important development of our work could be the design of a parallel solver. Each core could work only on a subset of the initial clauses, without communications. This could also allow us to extend our approach to tuples of communities instead of pairs of communities.

Another extension can be the improvement of our approach by trying to guess which pairs of communities are important to work on. It is important to link the community structure of formulas with their initial problem and generation. For instance, in bounded model checking, a community could be a sub-circuit. Linking the original problem with the detected communities could be also an interesting direction to investigate.

9.2.7 CDCL improvements

Finally, we recall the possibly most important future work: the improvement of the state-of-the-art CDCL techniques. This thesis gives some important insights about the underlying structure of applications problems, with the aim of improving these techniques. In fact, one of the applications we propose (i.e., the detection of relevante learnt clauses; see Chapter 7 for more details) is directly related to a CDCL component: the clause removal policy.

Besides this contribution, there is a number of hypotheses, which relate the underlying structure of SAT instances to some particular CDCL component. Some examples could be: can we use the fractal dimension to implement a more efficient restart policy? Can we implement a heuristic that takes into account the the scale-free structure of the formula? Can we design a clause learning mechanism based on the community structure and the partition of the formula in order to learn more useful clauses? And so forth...

It is worth noting that we have tested some of these hypotheses. In particular, our initial focus was to find a heuristic that uses some notion of structure. To this purpose, we tested some static heuristic, as the most frequent variable, the less frequent variable, the variables of communities ordered by community size in both ascending and descending order. Unfortunately, none of these hypotheses resulted into an overall performance of the solver.

Bibliography

Achlioptas, D., Gomes, C., Kautz, H., and Selman, B. (2000). Generating satisfiable problem instances. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 256–261.

Aha, D. and Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

Albert, R., Jeong, H., and Barabási, A.-L. (1999). The diameter of the WWW. *Nature*, 401:130–131.

Aldecoa, R., Orsini, C., and Krioukov, D. V. (2015). Hyperbolic graph generator. *Computer Physics Communications*, 196:492–496.

Almeida, H., Neto, D. O. G., Jr., W. M., and Zaki, M. J. (2011). Is there a best quality metric for graph clusters? In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'11)*, pages 44–59.

Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2014). The fractal dimension of SAT formulas. In *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14)*, pages 107–121.

Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2015a). On the classification of industrial SAT families. In *Proceedings of the 18th International Conference of the Catalan Association for Artificial Intelligence (CCIA'15)*, pages 163–172.

Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., and Levy, J. (2016). Structure features for SAT instances classification. *Journal of Applied Logics.* Submitted.

Ansótegui, C., Bonet, M. L., and Levy, J. (2009a). On the structure of industrial SAT instances. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP'09)*, pages 127–141.

Ansótegui, C., Bonet, M. L., and Levy, J. (2009b). Towards industrial-like random SAT instances. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 387–392.

Ansótegui, C., Giráldez-Cru, J., and Levy, J. (2012). The community structure of SAT formulas. In *Proceedings of the 15st International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 410–423.

Ansótegui, C., Giráldez-Cru, J., Levy, J., and Simon, L. (2015b). Using community structure to detect relevant learnt clauses. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, pages 238–254.

Audemard, G. and Simon, L. (2009). Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 399–404.

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286:509–512.

Barber, M. J. (2007). Modularity and community detection in bipartite networks. *Physical Review E*, 76(6):066102.

Biere, A. (2012). Lingeling and friends entering the SAT Challenge 2012. In *Proceedings of the SAT Challenge 2012: Solver and benchmark descriptions*, pages 33–34.

Biere, A. (2014). Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver Lingeling. In *Proceedings of the 5th Pragmatics of SAT Workshop (POS'14)*.

Biere, A. and Sinz, C. (2006). Decomposing SAT problems into connected components. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):201–208.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.

Boufkhad, Y., Dubois, O., Interian, Y., and Selman, B. (2005). Regular random k-SAT: Properties of balanced formulas. *Journal of Automated Reasoning*, 35(1-3):181–200.

Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2008). On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188.

Braunstein, A., Mézard, M., and Zecchina, R. (2005). Survey propagation: An algorithm for satisfiability. *Random Structure Algorithms*, 27(2):201–226.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Burg, S., Kaufmann, M., and Kottler, S. (2012). Creating industrial-like SAT instances by clustering and reconstruction. In *Proceedings of the 15st International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 471–472.

Chen, J. (2014). A Bit-encoding phase selection strategy for satisfiability solvers. In *Proceedings of the 11th Annual Conference on Theory and Applications of Models of Computation (TMAC'14)*, pages 158–167.

Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, 70(6):066111.

Clauset, A., Shalizi, C. R., and Newman, M. E. J. (2009). Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703.

Cleary, J. G. and Trigg, L. E. (1995). K*: An instance-based learner using an entropic distance measure. In *Proceedings of the 12th International Conference on Machine Learning (ML'95)*, pages 108–114.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning (ML'95)*, pages 115–123.

Coja-Oghlan, A. (2014). The asymptotic k-SAT threshold. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC'14)*, pages 804–813.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC'71)*, pages 151–158.

da F. Costa, L., Rodrigues, F. A., Travieso, G., and Boas, P. R. V. (2005). Characterization of complex networks: A survey of measurements. *Advances in Physiscs*, 56:167–242.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215.

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Dinh, T. N. and Thai, M. T. (2011). Finding community structure with performance guarantees in scale-free networks. In *Proceedings of the IEEE 3rd International Conference on Privacy, Security, Risk and Trust and IEEE 3rd International Conference on Social Computing (SocialCom/PASSAT'11)*, pages 888–891.

Eén, N. and Biere, A. (2005). Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 61–75.

Eén, N. and Sörensson, N. (2003). An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518.

Erdös, P. and Rényi, A. (1959). On random graphs. *Publicationes Mathematicae*, 6:290–297.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3-5):75–174.

Frank, E., Wang, Y., Inglis, S., Holmes, G., and Witten, I. (1998). Using model trees for classification. *Machine Learning*, 32(1):63–76.

Ganian, R. and Szeider, S. (2015). Community structure inspired algorithms for SAT and #sat. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, pages 223–237.

Gent, I. P., Hoos, H. H., Prosser, P., and Walsh, T. (1999). Morphing: Combining structure and randomness. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI'99)*, pages 654–660.

Giráldez-Cru, J. and Levy, J. (2015). Benchmarks description. In *Proceedings of the SAT Race 2015.*

Giráldez-Cru, J. and Levy, J. (2015). A modularity-based random SAT instances generator. In *Proceedings of the 24st International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1952–1958.

Giráldez-Cru, J. and Levy, J. (2016). Generating SAT instances with community structure. *Artificial Intelligence.* Accepted with revisions.

Gomes, C. P., Fernández, C., Selman, B., and Bessière, C. (2004). Statistical regimes across constrainedness regions. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)*, pages 32–46.

Gomes, C. P. and Selman, B. (1997). Problem structure in the presence of perturbations. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 221–226.

Gomes, C. P. and Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62.

Gomes, C. P., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 431–437.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18.

Heule, M. J. H., van Zwieten, J. E., Dufour, M., and van Maaren, H. (2004). March_eq: implementing additional reasoning into an efficient Look-ahead SAT solver. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 345–359.

Hogg, T. (1996). Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81(1-2):127–154.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th Conference on Learning and Intelligent Optimization (LION-5'11)*, pages 507–523.

Järvisalo, M., Berre, D. L., Roussel, O., and Simon, L. (2012a). The international SAT solver competitions. *AI Magazine*, 33(1).

Järvisalo, M., Heule, M., and Biere, A. (2012b). Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR'12)*, pages 355–370.

Järvisalo, M., Kaski, P., Koivisto, M., and Korhonen, J. H. (2012c). Finding efficient circuits for ensemble computation. In *Proceedings of the 15st International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 369–382.

Järvisalo, M. and Niemelä, I. (2008). The effect of structural branching on the efficiency of clause learning SAT solving: An experimental study. *Journal of Algorithms*, 63:90–113.

John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI'95)*, pages 338–345.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2011). Algorithm selection and scheduling. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*, pages 454–469.

Kadioglu, S., Malitsky, Y., Sellmann, M., and Tierney, K. (2010). ISAC - Instance-specific algorithm configuration. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, pages 751–756.

Kahlert, L., Krüger, F., Manthey, N., and Stephan, A. (2015). Riss solver framework v5.05. In *Proceedings of the SAT Race 2015*.

Kannan, R., Vempala, S., and Vetta, A. (2004). On clusterings: good, bad and spectral. *Journal of the ACM*, 51(3):497–515.

Katebi, H., Sakallah, K. A., and Marques-Silva, J. P. (2011). Empirical study of the anatomy of modern SAT solvers. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT'11)*, pages 343–356.

Katsirelos, G. and Simon, L. (2012). Eigenvector centrality in industrial SAT instances. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12)*, pages 348–356.

Kautz, H. A. and Selman, B. (2003). Ten challenges redux: Recent progress in propositional reasoning and search. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)*, pages 1–18.

Kautz, H. A. and Selman, B. (2007). The state of SAT. *Discrete Applied Mathematics*, 155(12):1514–1524.

Kullmann, O. (2009). Fundaments of branching heuristics. In *Handbook of satisfiability*, Frontiers in Artificial Intelligence and Applications, pages 205–244. IOS Press.

le Cessie, S. and van Houwelingen, J. (1992). Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201.

Levin, L. A. (1973). Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266.

Li, L., Alderson, D., Doyle, J. C., and Willinger, W. (2005). Towards a theory of scale-free graphs: definition, properties, and implications. *Internet Mathematics*, 2(4):431–523.

Liang, J. H., Ganesh, V., Poupart, P., and Czarnecki, K. (2015a). Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In *Proceedings of the 30th National Conference on Artificial Intelligence (AAAI'16)*, pages 225–241.

Liang, J. H., Ganesh, V., Zulkoski, E., Zaman, A., and Czarnecki, K. (2015b). Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In *Proceedings of the 11th International Haifa Verification Conference on Hardware and Software: Verification and Testing (HVC'15)*, pages 225–241.

Lu, F., Wang, L.-C., Cheng, K.-T., and Huang, R. C.-Y. (2003). A circuit SAT solver with signal correlation guided learning. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, pages 10892–10897.

Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2011). Non-model-based algorithm portfolios for SAT. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT'11)*, pages 369–370.

Mandelbrot, B. B. (1983). *The fractal geometry of nature.* Macmillan.

Manthey, N. (2015). Results of Riss on modularity-based SAT benchmarks. Personal Communication.

Marques-Silva, J. P., Lynce, I., and Malik, S. (2009). Conflict-driven Clause Learning SAT solvers. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press.

Marques-Silva, J. P. and Sakallah, K. A. (1999). Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521.

Martins, R., Manquinho, V. M., and Lynce, I. (2013). Community-based partitioning for maxSAT solving. In *Proceedings of the 16st International Conference on Theory and Applications of Satisfiability Testing (SAT'13)*, pages 182–191.

Mertens, S., Mézard, M., and Zecchina, R. (2006). Threshold values of random K-SAT from the cavity method. *Random Structure and Algorithms*, 28(3):340–373.

Mitchell, D., Selman, B., and Levesque, H. (1992). Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Annual Design Automation Conference (DAC'01)*, pages 530–535.

Neves, M., Martins, R., Janota, M., Lynce, I., and Manquinho, V. M. (2015). Exploiting resolution-based representations for MaxSAT solving. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, pages 272–286.

Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133.

Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113.

Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G., and Simon, L. (2014). Impact of community structure on SAT solver performance. In *Proceedings of the 17st International Conference on Theory and Applications of Satisfiability Testing (SAT'14)*, pages 252–268.

Newsham, Z., Lindsay, W., Ganesh, V., Liang, J. H., Fischmeister, S., and Czarnecki, K. (2015). SATGraf: Visualizing the evolution of SAT formula structure in solvers. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, pages 62–70.

Nikolic, M., Maric, F., and Janicic, P. (2009). Instance-based selection of policies for SAT solvers. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, pages 326–340.

Nikolic, M., Maric, F., and Janicic, P. (2013). Simple algorithm portfolio for SAT. *Artificial Intelligence Review*, 40(4):457–465.

Papadopoulos, F., Kitsak, M., Serrano, M. Á., Boguñá, M., and Krioukov, D. (2012). Popularity versus similarity in growing networks. *Nature*, 489:537–540.

Peng, H., Long, F., and Ding, C. (2005). Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238.

Pipatsrisawat, K. and Darwiche, A. (2007). A lightweight component caching scheme for satisfiability solvers. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, pages 294–299.

Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*, pages 185–208. MIT Press.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc.

Raghavan, U. N., Albert, R., and Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118.

Rodriguez-Lujan, I., Huerta, R., Elkan, C., and Cruz, C. S. (2010). Quadratic programming feature selection. *Journal of Machine Learning Research*, 11:1491–1516.

Roy, J. A., Markov, I. L., and Bertacco, V. (2004). Restoring circuit structure from SAT instances. In *Proceedings of the 13th International Workshop on Logic and Synthesis (IWLS'04)*.

Selman, B., Kautz, H. A., and McAllester, D. A. (1997). Ten challenges in propositional reasoning and search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 50–54.

Selman, B., Mitchell, D. G., and Levesque, H. J. (1996). Generating hard satisfiability problems. *Artificial Intelligence*, 81(1–2):17–29.

Silverthorn, B. and Miikkulainen, R. (2010). Latent class models for algorithm portfolio methods. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI'10)*.

Simon, L. (2014). Post mortem analysis of SAT solver proofs. In *Proceedings of the 5th Pragmatics of SAT Workshop (POS'14)*, pages 26–40.

Slater, A. (2002). Modelling more realistic SAT problems. In *Proceedings of the 15th Australian Joint Conference on Artificial Intelligence (AJCAI'02)*, pages 591–602.

Song, C., Gallos, L. K., Havlin, S., and Makse, H. A. (2007). How to calculate the fractal dimension of a complex network: the box covering algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2007(03):P03006.

Sonobe, T., Kondoh, S., and Inaba, M. (2014). Community branching for parallel portfolio SAT solvers. In *Proceedings of the 17st International Conference on Theory and Applications of Satisfiability Testing (SAT'14)*, pages 188–196.

Walsh, T. (1999). Search in a small world. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 1172–1177.

Walsh, T. (2001). Search on high degree graphs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 266–274.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393:440–442.

Williams, R., Gomes, C. P., and Selman, B. (2003). Backdoors to typical case complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1173–1178.

Xie, J., Kelley, S., and Szymanski, B. K. (2013). Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys*, 45(4):43:1–43:35.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2007). The design and analysis of an algorithm portfolio for SAT. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, pages 712–727.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32(1):565–606.

Yates, R. A., Raphael, B., and Hart, T. P. (1970). Resolution graphs. *Artificial Intelligence*, 1(4):257–289.