Eigenvector centrality in industrial SAT instances [⋆]

George Katsirelos[1] and Laurent Simon[2]

[1] UBIA, INRA, Toulouse, France
email: george.katsirelos@inra-toulouse.fr,
[2] Univ Paris-Sud, LRI / CNRS UMR8623
Orsay, F-91405, email: simon@lri.fr

Abstract. Despite the success of modern SAT solvers on industrial instances, most of the progress relies on intensive experimental testing of improvements or new ideas. In most cases, the behavior of CDCL solvers cannot be predicted and even small changes may have a dramatic positive or negative effect. In this paper, we do not try to improve the performance of SAT solvers, but rather try to improve our understanding of their behavior. More precisely, we identify an essential structural property of industrial instances, based on the Eigenvector centrality of a graphical representation of the formula. We show how this static value, computed only once over the initial formula casts new light on the behavior of CDCL solvers.
We also advocate for a better partitionning of industrial problems. Our experiments clearly suggest deep discrepancies among the families of benchmarks used in the last SAT competitions.

1 Introduction

Despite the impressive progress made in the practical solving of SAT problems in recent years, little work has been done to experimentally study the behavior of those so-called "Modern SAT Solvers". Those solvers are based on a variant of the backtrack search DPLL [4] procedure with learning [7]. While the lookahead architecture of DPLL solvers was relatively easy to understand (all the power of solvers were based on efficient pruning heuristics), the behavior of CDCL (Conflict Driven Clause Learning algorithms, *i.e.* the "Modern SAT solvers") is highly unpredictable, due to its lookback architecture and dynamic branching heuristics.

Nowadays the picture is quite complex: we know the ingredients to build an efficient SAT solver (highly reactive heuristic, resolution-based learning, frequent restarts, frequent clause database reduction), and we are constantly improving them. However, we can hardly explain why those ingredients are so efficient on "real-world" problems. It is claimed that those instances have a particular structure, well suited to the CDCL mechanisms. But what exactly is this particular structure? It is understood that real world instances are different from uniform random instances, but only recently has some progress been made toward characterizing this structure by finding that these instances exhibit modularity [1, 2]. It is also known [8] that many real-world instances share the small-world property of graphs.

[⋆] This work was partially funded by the "Agence nationale de la Recherche", reference ANR-10-BLA-0214.

In this paper, we use a new approach to identify hidden structure in "application" instances. First, we base our approach on a directed graph representation of the formula that separates positive and negative literals, which is close to the factor graph representation. Because some industrial instances are really huge, we compute the eigenvector centralities of the vertices by an efficient iterative algorithm inspired by the Google PageRank algorithm. It provides a very good approximation of the centrality of a clause or a literal in the formula. Intuitively, this measure gives us the frequency with which an infinite random walk in the graph will traverse this vertex. Second, we relate this static measure, computed only once on a preprocessed instance, to measures computed during the search of a CDCL solver. Our experiments are performed with a typical CDCL solver (GLUCOSE). Finally, we show how some structural properties of the initial formula may guide the CDCL search. Moreover, we clearly show that the granularity of SAT problems in the SAT competitions is now too coarse: some families of benchmarks exhibit distinct behavior.

2 Eigenvector Centrality on SAT Directed Graphical Models

Due to lack of space, we assume the reader is familiar with CDCL solvers.

2.1 The SAT Directed Graphical Model

In previous work, SAT instances were studied via the Variable Incidence Graph (VIG) or the Clause-Variable Incidence Graph (CVIG) [2]. Those graphs are undirected. In the VIG representation, each variable of the initial formula is encoded by exactly one vertex in the graphical model. An edge is added between the vertices of two variables if they both occur in at least one clause in the initial formula. In the CVIG, the graph is bipartite (each clause and each variable correspond to exactly one vertex). An edge links a clause vertex and a variable vertex iff the variable occurs in the clause.

Variations are possible. Weights can be added, for instance (links induced by shorter clauses being stronger). In the factor graph representation [3], based on CVIG, edges are additionally labelled by the polarity of the variables in the CVIG graph.

Our graphical representation is contructed such that a random walk on it mimics a random walk of a repair algorithm that would try to satisfy as many clauses as possible. The graph is bipartite and vertices are labeled by clauses and literals (not variables). There is an edge from a clause C to a literal l iff l occurs positively in the clause. There is also an edge from a literal l' to a clause C' iff the literal l' occurs negatively in the clause. A random walk on this graph mimics the operation of a naive algorithm which repairs a global assignment by randomly selecting a clause, randomly flipping a variable that does not satisfy it and then moving on to a clause that is not satisfied by the new value of that variable, repeating the process until it finds a satisfying assignment.

2.2 The Pagerank algorithm

The PageRank algorithm [5] is an efficient iterative algorithm approximating the stationary distribution of a random walk on a graph. It computes an approximation of eigenvector centrality and is particularly well suited to large graphs, such as the web. To ensure and accelerate convergence, it uses a *damping* factor, that allows the random walk to jump, with a small probability, to any vertex, uniformly at random.

The centrality of a literal C_l is exactly the approximation of the eigenvector centrality of the vertex returned by the above algorithm. We extend it to the centrality of a variable by taking the geometric average over the two literals, $Cv_x = \sqrt{{C_x}^2 + {C_{\neg x}}^2}$. In the rest of this paper, we use both literal and variable centrality.

Once we have computed the centrality of all vertices in the graph, it is possible to infer additional information. For instance, a vertex being the most central of its neighborhood is likely to be on the fringe [6] between communities.

2.3 Pagerank on SAT problems

We used a damping factor of 0.95 (Google used 0.85) and an accepted total error of $1e-9$ (sum of the changes of all vertices during one iteration). In most cases, the algorithm converged after a few hundred iterations.

We chose to use the Satelite preprocessor on all instances before computing the centrality and running the CDCL solver, for two reasons. First, the real instances passed to CDCL solvers are, in most of the cases, filtered by preprocessing, so it makes sense to work with the same input as the CDCL solver. Second, preprocessing get rid of noise like unit clauses which may complicate computation (the Markov chain of a graph that represents unit clauses is not ergodic) and produce measures that have a worse fit.

In our experiments, we tested all 658 benchmarks from SatRace 2008, SatCompetition 2009 and 2011, in the Application category. We fixed a cutoff of 5 Million conflicts, but placed no bound on CPU time.

3 Observations and Analysis of CDCL behavior

3.1 Centrality of Variables for decisions / propagations

In this first part, we focus only on variable centrality (not literal). We study the centrality of variables picked by the branching heuristic. In figure 1(left), we compare the average centrality of picked variables against the average centrality of all variables. In this figure, as in the rest of the figures in this paper, each point representing one instance. This figure clearly shows that picked variables are the most central variables in the formula. They are almost always above the average centrality of the formula. This may be one factor of the efficiency of the VSIDS heuristics: by branching on central variables, it encourages decomposition of the formula.

The comparison with the figure 1 (right) is also very interesting. Propagated variables are, in almost all the cases, more central than average. However, closer examination shows that the points on the left cluster below those on the right. This raises the question of whether decision variables are more central than propagated variables. We answer this question (positively) in section 3.4, by first refining the benchmarks into families: a few families of benchmarks display opposite behaviors and no clear global picture can be drawn without splitting the set of benchmarks into families.

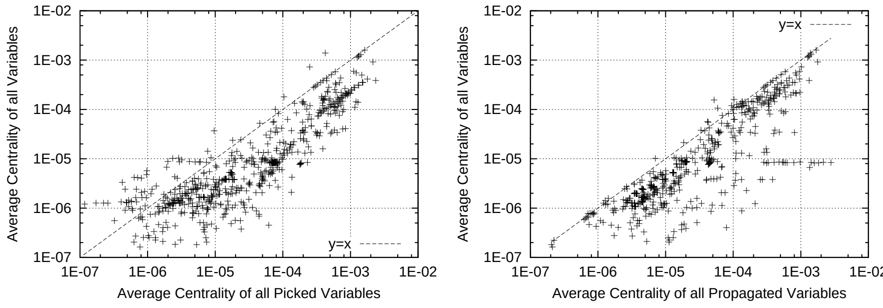


Fig. 1. (**Left**) Average Centrality of Picked Variables (x-axis) against Average Centrality of all Variables (y-axis). (**Right**) Average Centrality of Propagated Variables (x-axis) against Average Centrality of all Variables (y-axis).

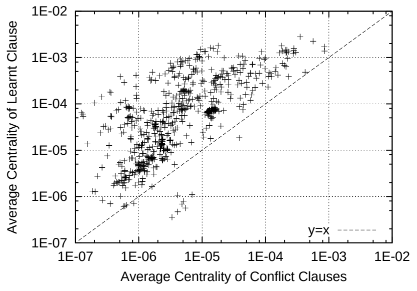


Fig. 2. Average Centrality of Variables (x-axis) occurring in conflict clauses against Average Centrality of all Variables (y-axis) occurring in learnt clauses.

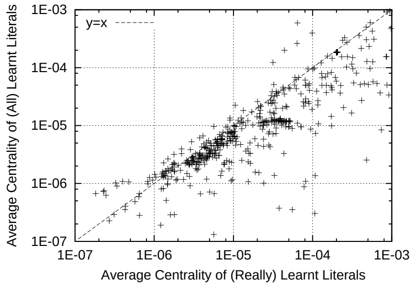
3.2 Centrality of Variables occurring in conflicts and learnt clauses

In this experiment, we compare the centrality of conflict clauses (clauses found to be empty during search) against learnt clauses. Because it was impractical to recompute the centrality of the entire formula at each conflict, we measure, for a clause, its centrality as the average of the centrality of its variables (this is again not based on literal centrality).

The results, shown in figure 2, clearly state that learnt clauses contain more central variables than conflict clauses. This is a surprising result. If learnt clauses link together central variables, it intuitively follows that conflict clauses should also be central, but this is not the case. One explanation may come from the notion of fringe (see section 2.2). Learnt clauses may be built upon a few variables from fringes. Conflicts could be detected inside a cluster, thus having no fringe variables in it. This is clearly worth further investigation.

3.3 Centrality of learnt unit clauses

Unit clauses learnt during search are particularly important. They clearly simplify the problem, and may be considered as witnesses of which parts of the search space the



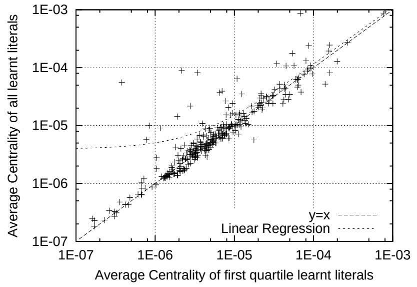


Fig. 3. (**Left**) Average Centrality of Learnt Literals during conflict analysis (x axis) against the Average Centrality of all learnt literals. (**Right**) Average Centrality of Learnt Literals during the first quartile of the computation (x axis) against the Average Centrality of all learnt literals.

solver has explored. Typically, in an UNSAT instance, the solver will learn unit clauses at a rate that is relatively high in the beginning of the search and slows as the search goes on, suggesting that each new unit clause is harder and harder to prove. In this experiment, we explore whether there is a relationship between the "hardness" of a literal (when it was learnt) and its centrality. We also note here that this behavior is also observed on SAT instances, but is not general. On some problems, the solver learns no unit clauses until the very end.

There are in fact two kinds of unit clauses that can be learnt. When conflict analysis produces a unit clause, the solver immediately adds this literal as a new fact in the formula and, in many cases, a few other unit clauses are immediately propagated. We distinguish this first literal from the propagated literals, even though they were propagated at the root of the search tree, and call it a "really" learnt literal.

From now on, we consider literal centrality instead of variable centrality. First, figure 3 (left) shows the different kind of unit clauses learnt. This figure clearly shows that the "really" learnt literal is the most central one. Intuitively, other literals simply follow from that assignment. This is compatible with our hypothesis of CDCL solvers working on fringes. This can also be explained by the fact that the branching heuristic favors more central variables (see figure 1). Thus, central literals are the most likely to be learnt first.

We now answer the initial question of this subsection. The answer is given in figure 3 (right). We compare the average centrality of the first quartile of all learnt literals against the centrality of the learnt literals during the whole computation (until the formula is solved or the cutoff is reached). Even if it is not clearly above the $y = x$ line, there is a general tendency showing that centrality of learnt literals tends to increase as the search progresses (the regression line is clearly above the line). This result is interesting for several reasons. First, it explains in part how CDCL solvers focus on particular parts of the search space. Second, it casts new light on parallelization of SAT solvers. In order to efficiently parallelize a CDCL solver, we need to understand which parts of the search space the solver explores, in order to distribute the work evenly. More precisely, by knowing which parts of the search space are relevant to the current proof, it is possible to distribute the effort by giving each process a relevant and precise part

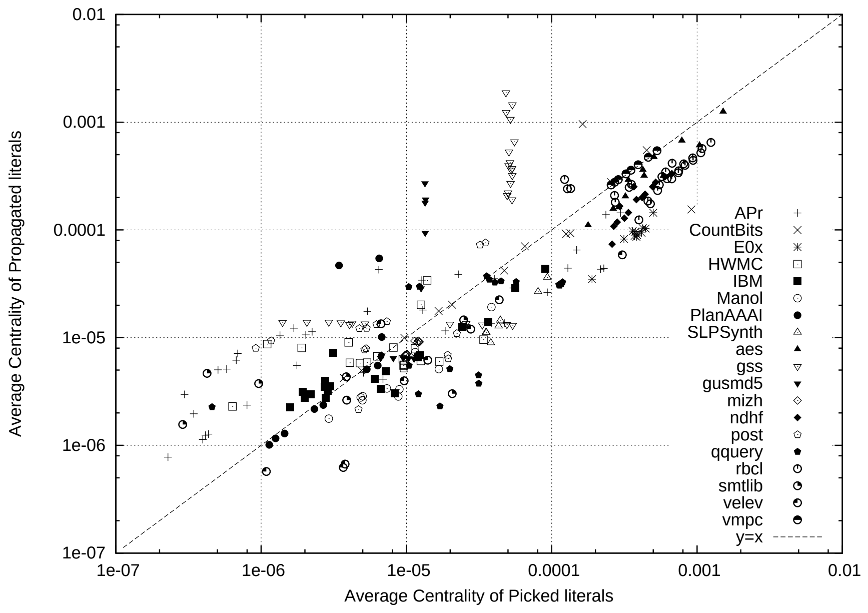


Fig. 4. Average Centrality of Picked Literals against Average Centrality of Propagated Literals, refined by series of benchmarks.

of the proof to build, instead of only ensuring orthogonal searches. Moreover, it shows that, intuitively, each unit clause may have a price, in terms of proof length. That means that trying to prove it by orthogonal search may not be the best choice (each search will have to pay more or less the same price).

3.4 Using families of benchmarks for refining results

We now refine our study by selecting subfamilies among the 653 benchmarks we tested. It is indeed hard to identify general tendencies, because of the discrepancy between different families. We start with figure 4. It shows that, in general, CDCL solvers branch on central literals and propagate less central ones. Most of the instances are below the $y = x$ line except a few families. For instance, the crypto problems (gss, gusmd5) do not follow this trend. This result suggests that these families of instances might benefit from specialized CDCL solvers.

The first UIP literal [9], during conflict analysis, is also an essential ingredient of CDCL solvers. Figure 5 shows that, except for a few families (smtlib, velev, vmpc), the FUIP literal tends to be one of the most central literal in learnt clauses. Despite the discrepancy between decision literals and propagated literals, even in crypto problems, first UIP literals tend to be very central (see the very small cloud of the gss problems for instance).

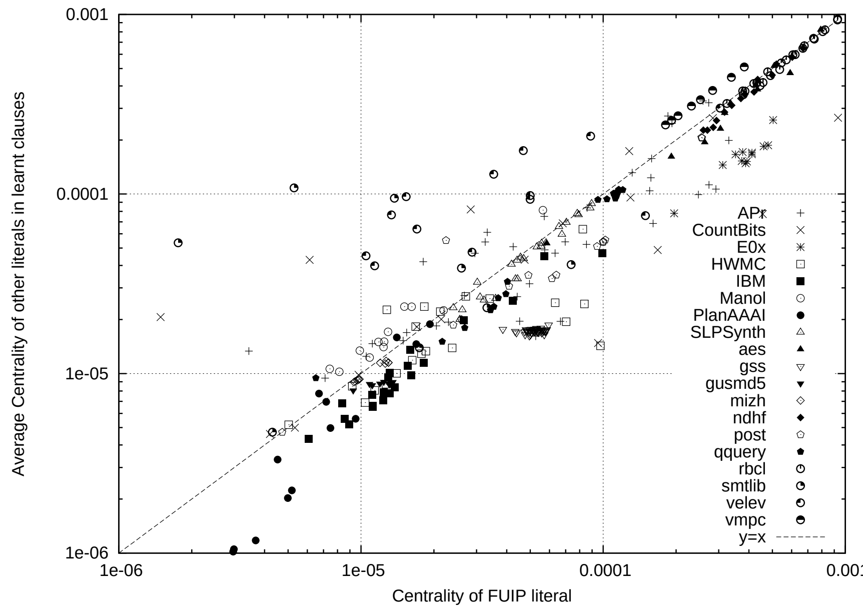


Fig. 5. Average Centrality of FUIP Literals against Average Centrality of other literals in learnt clauses. Most of the instances are below the line $y = x$ except a few families.

3.5 Influence of the biases of variables on the search

Once the centrality of literals has been computed, one may define the biases of a variable as the tendency of the random walk to visit more often the variable positively or negatively. In this last experiment, we explore the relationship between this tendency and the actual CDCL search. The biases of a variable x is defined as $bi(x) = (Cx_{pos} - Cx_{neg})/(Cx_{pos} + Cx_{neg})$, and takes values between -1 and 1. We also defined the "observed Biases" as $ob(x)(Nx_{pos} - Nx_{neg})/(Nx_{pos} + Nx_{neg})$. In the above notations, Cx_{pos} (resp. Cx_{neg}) is the centrality of literal x (resp. $\neg x$). Nx_{pos} (resp. Nx_{neg}) is the number of times the literal x (resp. $\neg x$) is propagated during the CDCL search (until the solution is found, or the bound of 5M conflicts is reached).

In order to study the possible relationships between $bi(x)$ and $ob(x)$, we computed the disagreement between them, $d(x) = |bi(x) - ob(x)|$ (a value between 0 and 2). A value of 1, for instance, means that the CDCL solver did not follow the prediction given by $bi(x)$. A value of 0 means that it strictly follows it, and a value of 2 means that it systematically take the opposite. We also refined the predictive power of $bi(x)$ by considering that only variables having $|bi(x)| > 0.5$ are "biased", meaning that the bias value is significant. We do the same for $ob(x)$ by considering it as meaningful if the measure for the variable x is based on more than 10 assignments during search. The x-axis of figure 6 takes all variables into account (even variables with biases of 0, or variables that were never assigned during search), while the y-axis takes only biased

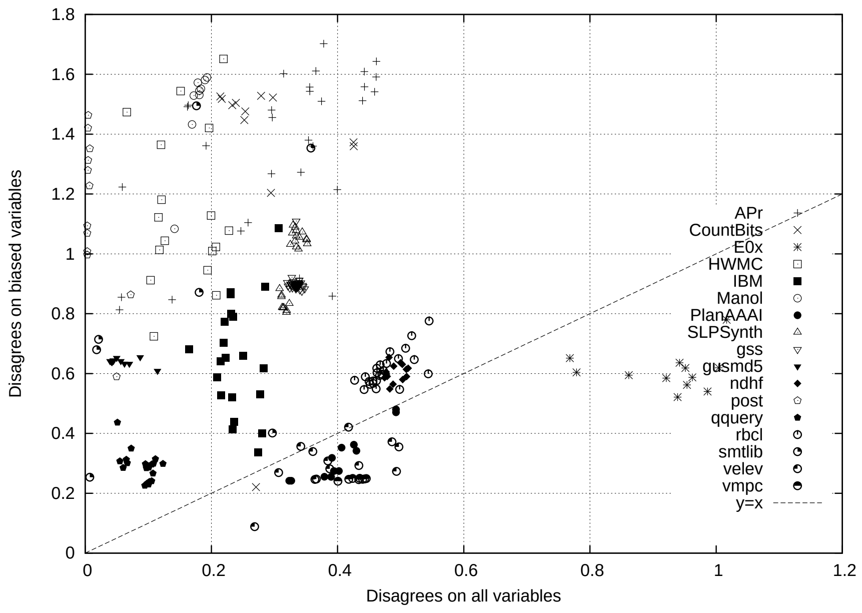


Fig. 6. Predicting phase of variables according to their biases.

and meaningful variables. Note that the existence of biased variables is not guaranteed. On some instances, there was only a few biased variables. However, as shown figure 6, some instances show strong correlation between the two values, especially when restricting the cases to biased variables. Clearly, no general rule can be drawn here. However, benchmarks from the same families generally cluster around a common disagreement value.

4 Conclusion

We have shown in this paper that the the centrality of literals and variables in industrial SAT instances is correlated with various aspects of the behavior of CDCL solvers during search. Let us recall that the centrality is computed only at the beginning. The values we compute do not change during search. Despite this approximation, the results we obtained clearly show that centrality plays an important role. This is also one of the first experimental studies of CDCL solvers, that link such a static measure to their actual behavior. We also showed that, even in the application category, families of benchmarks show a large discrepancy in their behavior. This clearly suggests that a finer granularity of categories is needed, in order to specialize solvers and continue to improve them.

However, we were not yet able to turn these observations into predictions and guess, for instance, which parts of the search space the solve is likely to explore or which literal are likely to be first UIP literals or unit clauses. Our quest to explain and understand the CDCL solvers needs more experimental study of those complex systems we built.

References

1. Carlos Anstegui, Jess Girldez, and Jordi Levy. The community structure of sat formulas. In *SAT*, 2012.
2. Carlos Anstegui and Jordi Levy. On the modularity of industrial sat instances. In *CCIA*, 2011.
3. Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures & Algorithms*, 27(2):201–226, 2005.
4. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communication of ACM*, 5(7):394–397, 1962.
5. Amy Langville and Carl Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton Unviersity Press, 2006.
6. Devavrat Shah and Tauhid Zaman. Community detection in networks: The leader-follower algorithm. arXiv:1011.0774v1, 2010.
7. Joao Marques Silva and Karem Sakallah. Grasp - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.
8. Toby Walsh. Search in a small world. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1172–1177, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
9. Lintao Zhang, Conor Madigan, Matthew Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of IEEE/ACM International Conference on Computer Design (ICCAD)*, pages 279–285, 2001.