EPiC Series in Computing

Volume XXX, 2017, Pages 1–13

GCAI 2017. 3rd Global Conference on Artificial Intelligence

Improving SAT Solver Performance with Structure-Based Preferential Bumping

Sima Jamali and David Mitchell

Simon Fraser University, Vancouver, Canada
mitchell@cs.sfu.ca, sj88@sfu.ca

Abstract

We present a method we call structure-based preferential bumping, a low-cost way to exploit formula structure in VSIDS-based SAT solvers. We show that the SAT solver Glucose, when modified with preferential bumping of certain easily identified structurally important variables, out-performs unmodified Glucose on the industrial formulas from recent SAT solver competitions.

1 Introduction

Reference to "instance structure" is frequent in discussions of SAT solving, both in trying to explain the (often surprising) performance of existing solvers and in proposals for improving this performance. Modularity is an appealing notion of useful structure. If a formula can be decomposed into sub-formulas with few shared variables, then each assignment to the shared variables induces smaller sub-problems that can be solved independently. Unfortunately, while industrial formulas have interesting structure, few have suitable decompositions where the shared or "connecting" set of variables is small enough for enumeration of all assignments to be effective. This makes direct application of tree-width based dynamic programming, for example, unlikely to be feasible, as the tree-widths of industrial formulas are quite large [15]. The work reported here is part of a research program to develop lightweight methods for exploiting formula structure.

Structure of CNF formulas is often studied in terms of an underlying graph, such as the primal graph. The nodes of the primal graph are the variables of the formula, and there is an edge between two nodes if the corresponding variables co-occur in a clause (in either polarity). Recently, a number of studies have examined the "community structure" of primal graphs of industrial CNF formulas [6, 19, 5, 12]. A graph is considered to have "good community structure" if its nodes can be partitioned into "communities" so that there are relatively more edges within than between communities. This can be thought of as a kind of modularity, and indeed one measure of quality of community structure, introduced in [18], is named modularity in the literature. It has been observed that this modularity measure is correlated with CDCL SAT solver running times [19]. Edges with end points in distinct communities are called bridges, and nodes in such edges are called bridge variables. Bridge variables are the "connecting" variables in this notion of modularity. It has been observed that the VSIDS decision heuristic in CDCL SAT solvers chooses bridge variables much more frequently than other variables [12].

This raises several questions. What causes the VSIDS preference for bridge variables? Is this preference good or bad? Can we manipulate this preference to alter solver behaviour? Our results here partially answer the latter two questions.

A community graph for a formula has communities as vertices and an edge between two communities if there is a bridge between them. Weights may be used to reflect community size and the number of bridges between two communities. Community graphs of industrial formulas are diverse in structure, but we observed that some have or contain a rather simple, nearly linear, structure (see Section 3). For these formulas, the variables in the central communities of this structure can be viewed as "connecting variables" between two sub-formulas of about half the size of the original, and with similar structure. This led us to examining the most central variables in the formulas, based on the global measure called betweenness centrality [11]. We found that VSIDS also shows a strong preference for the high-centrality variables.

Bridge variables and high centrality variables appear to be very interesting. However, the sets of these variables are too large to consider enumerating all assignments to them. We determined that, rather than doing this, we can simply encourage VSIDS to choose these variables more often by adjusting the "bump factor". We call this "preferential bumping". We show here that preferential bumping of high centrality and bridge variables can be used to improve the performance of a high-performance CDCL SAT solver.

The computations to find bridge and high centrality variables can be expensive, in some cases taking longer than running a SAT solver on the instance. However, for many formulas, the computations are fast. Moreover, it seems that the information obtained tends to be more useful when the computation is fast than when it is not. Therefore, the strategy of simply running bridge detection or centrality computation algorithms for a short time as a pre-processing step is useful. If we obtain the structural information in time, we use it, otherwise we simply run the default version of the solver and pay only a small time penalty.

To evaluate the effectiveness of the resulting method, we compared the performance of the SAT solver Glucose [8, 1] with a modified version, that we call GlPB (for Glucose with Preferential Bumping). We chose Glucose because it is among the best solvers over recent competitions, is well known and relatively well understood, is easy to work with, and is very influential in that it is the base of several other successful solvers. We report data here on two versions of GlPB, identified as GlPB-Br-u and GlPB-HC-i, which perform preferential bumping of bridge and high centrality variables (respectively). On a large set of industrial benchmark formulas, GlPB-Br-u solved 5 more formulas than default Glucose, and GlPB-HC-i solved 12 more. GlPB-HC-i also solved one formula from the 2016 SAT solver competition that none of the 29 solvers in the competition solved within the competition timeout. While our time-out is the same as the competition, our hardware is not, so we cannot conclude that GlPB-HC-i would have solved it in the competition. However, the fact that no solver in the competition solved it in 5000 seconds does suggest that it is indeed a hard formula by current standards.

Summary of Contributions

1. We show that, by selectively altering the VSIDS "bump value" for specific sets of variables, we can influence the preference VSIDS has for choosing these variables. We call this "preferential bumping".

2. We show that preferential bumping of high centrality variables and bridge variables can improve the performance of a high-performance CDCL solver.

3. We show that sufficient structural information to improve performance with preferential bumping can be obtained quickly enough to be useful. That is, it is possible to produce a

solver that computes and uses structural information, and has improved overall running time on industrial formulas.

1.1 Related Work

Community structure is used in [7] to generate learned clauses that, when added to the original instance, improve solver performance on satisfiable instances. In [14, 17] community structure is used in SAT-based MaxSAT solvers. There, structure is used to partition the formula to obtain smaller witnesses of unsatisfiability, not to direct the SAT solver execution.

The method closest to ours (and which we were unaware of when beginning our experiments) is described in [20]. This method involves applying large bump values to variables in a particular community. However, the intent and scheme are very different than ours, as is reflected by the author's labelling of their method as "Community Branching". That work alters the parallel portfolio solver PeneLoPe [3] as follows. Each "worker" solver works on the entire formula, but is assigned some specific set of communities. The key step is that after a chosen restart all variables in one of those communities are bumped a single time by 100. The intent, as described in the paper, is that the solver will then branch on variables in that community before branching on others, thus focusing its search in that community. Our method is intended to "encourage" decisions on "important" variables over a period of time, thus gently increasing the number of decisions on those variables. In our method, we adjust the initial bump value for some selected variables by a small amount, and this affects the following sequence of bump values applied by VSIDS. In the method of [20], the activities of selected variables are bumped a single time by a large amount. We observe that in our method, setting a bump value of more than 1.2 (or less than 0.8) significantly reduced performance, whereas in [20] a bump value of 100 is used.

1.2 Organization of Paper

The remainder of the paper is organized as follows. In Section 2 we define the structural properties we use, describe how we compute them, and provide some data on the computation times. In Section 3 we describe the VSIDS decision heuristic and present data showing the preference VSIDS shows for several families of structurally significant variables. We also show that we can use preferential bumping to alter the degree of preference VSIDS shows for particular families of variables. In Section 4 we present our main results. We begin the section with an illustration of preferential bumping of sets of variables chosen based on visual inspection of "nice" community graphs. We then present results of running our modified versions of Glucose on a large set of industrial formulas. In Section 5, we discuss our results and future work.

2 Computation of Structural Properties

In this section, we define the structural properties we consider and how we compute them. We also show that, although the computation time for some formulas is prohibitive, for the majority of formulas it is fast enough to be used as a pre-processing step in a SAT solver. We assume the reader is familiar with standard terminology regarding propositional CNF formulas and graphs.

2.1 Structural Properties and Computation

The primal graph of a propositional CNF formula ϕ (also called the variable incidence graph or the variable interaction graph), is the graph $G(\phi) = \langle V, E\rangle$ with V being the set of all variables in ϕ and $(p, q) \in E$ iff there is a clause $C \in \phi$ containing both p and q (either negated or not). By the degree of atom p we mean the degree of p in $G(\phi)$.

Community Structure If G is a weighted graph and P a partition of its vertices, then define

$$Q(G, P) = \sum_{p \in P} \left[\frac{\sum_{w,y \in p} w(x, y)}{\sum_{w,y \in V} w(x, y)} - \left(\frac{\sum_{x \in p} deg(x)}{\sum_{x \in V} deg(x)} \right)^2 \right].$$

The modularity Q of G is the maximum value of $Q(G, P)$ over all partitions P. Community structure detection for a formula is typically based on a weighted version of the primal graph, where each clause of size k contributes weight $1/\binom{n}{2}$ to each associated edge. We say a formula has good community structure if the modularity Q of its weighted primal graph is large. Computing a partition that maximizes Q is NP-hard, so for large graphs, heuristic methods are used.

Centrality The intuitive notion of "centrality" of a vertex in a graph may be measured in a number of ways. A very simple centrality measure is vertex degree, sometimes called degree centrality. The betweenness centrality of a vertex v in a graph G is the number of shortest paths between pairs of distinct vertices (not including v), that visit v. The betweenness centrality of a variable v in formula ϕ is the betweenness centrality of v in the un-weighted primal graph $G(\phi)$.

Computation We performed heuristic community detection and computed approximate betweenness centrality values using the corresponding functions in the NetworkX python package. We use the Louvain method [10] as implemented in the NetworkX package to detect community structure, following [12, 7, 5]. Exactly computing betweenness centrality requires an all-pairs-shortest-paths computation, which is too expensive for large graphs. The NetworkX 'betweenness_centrality' function takes a parameter that sets a sample size for shortest paths to use to generate approximate betweenness centralities. We set the sample size to the number of variables divided by 50.

Benchmark Sets and Execution Environment For our exploration of the cost of computing structural measures and their role in CDCL solvers, we used a set of formulas consisting of all formulas from the industrial categories of the 2013 and 2016 SAT solver competitions for which we could perform centrality computations within one hour, using sample size set to the number of variables divided by 50. The resulting set has 223 formulas from a wide variety families within the competition benchmark sets. The community detection took less then one hour for each of these formulas. We used this set of formulas in the experiments reported in Section 2.2 and in Section 3.

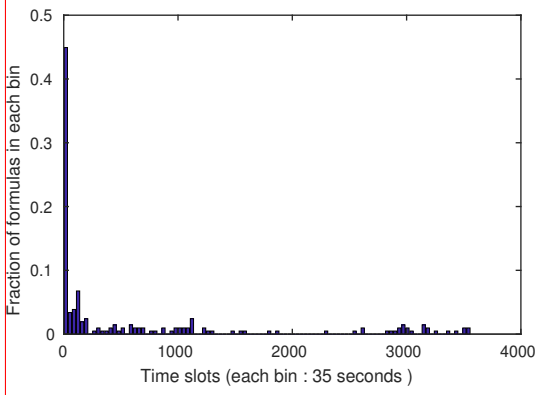
For the evaluation of the performance of our preferential bumping solvers presented in Section 4.2, we used the 742 distinct formulas from the industrial categories of the three most recent SAT solver competitions (2013, 2014 and 2016) [2].

All computations were run on Intel(R) Core(TM) 3.4 GHz i7-3770 quad-core processors (single-threaded) with 16Gb of RAM, running Linux.

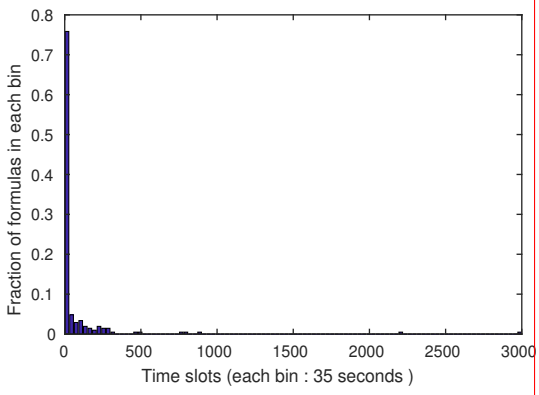
2.2 Structure Computations

For a significant fraction of formulas from industrial benchmarks, the time to detect community structure or compute variable centralities is substantial, and often greater than the time to decide satisfiability. For many industrial formulas we could not obtain good approximations of centralities in less than an hour. Clearly, one could not (at least with current methods) perform these computations in a SAT solver for every input. However, for a large fraction of the formulas reasonable approximations can be computed quite quickly.

Figure 1a shows a histogram of the centrality computation times for our set of 223 industrial formulas for which we could find good approximate centralities in one hour. As can be seen, for many formulas the computation is quite fast, and only a small fraction have relatively long computation times. More than half the formulas required less than 70 seconds. Figure 1b shows a histogram of community detection times, which shows a similar pattern. The community detection generally was faster than centrality computation, and more than three quarters of the formulas required less than 35 seconds.



(a) Histogram of Centrality Computation Times.



(b) Histogram of Community Detection Times

Further, it turned out that the most useful information for our preferential bumping schemes (at least as currently implemented) was generally obtained quickly. For example, for all instances that were solved by Glucose enhanced with centrality-based preferential bumping (GlPB-HC-i as described in Section 3.2) but not solved by default Glucose with the same time-out (see Section 4.2), centrality computation time was less than 150 seconds.

It seems slightly counter-intuitive that useful information for hard formulas should be easy to compute. We examined the correlation between running time of default Glucose, and the times to compute centrality and community information. Table 1 shows the results. Because SAT solver run times are very far from normally distributed, it is more appropriate to use a non-parametric measure of correlation than, for example, the well-known Pearson correlation. We use the Kendall τ rank-based correlation coefficient. A τ of 1 indicates perfect correlation, -1 perfect inverse correlation, and 0 no correlation at all. As can be seen, the times to compute both structural measures are highly correlated. Glucose solving time is only weakly correlated with structure computation times. However, the p values for these measures are extremely small ($p < 0.01$ is generally regarded as highly significant), so we can be fairly confident this correlation is real. The interesting point is that these correlations are negative: Computing structural information is easier for formulas that are harder for Glucose. Of course, for similar formulas of different size, structure computation time must increase with formula size. The negative correlation of solver time and structure computation time may reflect a bias in the

benchmark selection process: The small formulas are hard relative to their size, and the large formulas are easy relative to their size.

Measures	Kendal τ	p-value
Glucose *vs.* Centrality computation	-0.13	0.003
Glucose *vs.* Community detection	-0.20	$\ll 0.0001$
Centrality computation *vs.* Community detection	0.71	$\ll 0.0001$

Table 1: Correlations amoung structure computation and solver times.

3 VSIDS Preferences and Preferential Bumping

We assume the reader is familiar with the fundamentals of CDCL SAT solvers [13]. The VSIDS decision heuristic, with a number of small variations, has been the dominant decision heuristic almost since it was introduced in [16]. A useful recent discussion of VSIDS can be found in [9]. The VSIDS method is based on recording an "activity score" for each variable of the formula. At each decision, the unassigned variable with highest score is chosen. As implemented in Glucose, the activity score of a variable is updated each time the variable is seen while deriving a new learned clause. The update is performed by adding the current "bump value" to the score. The bump value is initially 1, and at each conflict it is multiplied by $1/d$, where d is the "decay factor" (VSIDS approximates an exponential moving average), which is initially 0.8 and is gradually increased to 0.95.

3.1 VSIDS Preferences

It was observed in [12] that VSIDS chooses bridge variables with far greater frequency than might be expected. Here, we confirm that this holds for our test benchmark set, and add several more observations. We were interested in the degree to which VSIDS, in Glucose, would show a preference for high degree variables, for high centrality variables, and also for high degree and high centrality bridge variables. (In [12] it is claimed that VSIDS prefers high centrality bridge variables. The centrality measure used there is degree centrality, a local property, whereas the betweenness centrality measure we use here is a global property.)

We ran Glucose on the formulas of our test benchmark set, and recorded the number of decisions from each variable family. Table 2 shows the mean of each of these measures over the 186 formulas from our test benchmark set that Glucose solved within our 5000 second time-out.

The first column of Table 2 identifies the family of variables in question. We defined the set of high degree variables to be the one-third fraction with the highest degree, and high centrality similarly. The second column gives the fraction of all atoms in the formula that belong to each family, while the third column gives the fraction of all decisions that are variables from the family. We observe that, for each family, the fraction of decisions belonging to the family is much higher than the fraction of atoms belonging to that family, indicating that VISDS chooses variables from these structurally "special" families more often than others. In fact (see discussion of the Precision column below) on average 63% of variables are not chosen as decisions even once during a run of the solver.

The forth column is the ratio of the fraction of decisions from each family to the fraction of atoms from the family. (It is the mean of the ratios, not the ratio of the means, so can't be

Family	Atoms (%)	Decisions (%)	Strength	Hit Rate	Precision
All	100	100	1.0	100	37
Bridge	49	83	2.86	77	64
High Degree	33	80	2.38	62	50
High Centrality	33	73	2.22	45	52
High Deg. Bridge	22	67	4.05	49	71
High Cent. Bridge	21	65	4.17	36	82
High Deg. High Cent.	18	65	3.8	34	69

Table 2: Measures of the degree to which VSIDS prefers "special" families of variables.

computed from the preceding columns.) This gives a measure of how "important" the family is relative to its size. The sets of bridge, high degree and high centrality variables all account for about twice as many decisions as their sizes would suggest. The high degree and high centrality bridge sets, while much smaller, account for far more decisions relative to their size.

The fifth and sixth column are standard evaluations of the predictive power of a factor, where we are viewing membership in the special family as a predictor of being chosen at least once as a decision variable. Hit rate, also called sensitivity or "true positive rate", is the fraction of special atoms that are chosen by VSIDS at least once during a run. Precision, also called "positive predictive value", is the fraction of variables that are chosen at least once that are also from the family. Notice that the All row of the Precision column gives us the fact that only 37% of variables are chosen as decisions at some time during a run.

Our data show that, while VSIDS preferentially chooses several families of "special" variables, some of these preferences are much stronger than others, and in particular, some are much stronger relative to the size of the "special" set. Although the high degree and high centrality bridges seem very interesting, in preliminary experiments, preferential bumping of these sets did not appear especially promising. It is not clear whether this is because there are just too few of them or something else.

VSIDS Preferences are Stable It has been observed in [7], and confirmed by our own experiments, that clause learning destroys the initial community structure of formulas. That is, the set of clauses the solver is working with after running for a while does not have the same community structure as the initial formula. (In particular, a large fraction of learned clauses would add bridges to the initial set of communities.) Since VSIDS, overall, prefers variables with certain initial structural roles, one might suspect that this preference changes during a run as the structure of the current clause set changes. Since our sets of "special variables" are computed only on the initial formula, it might be expected that they become less relevant over time. We cannot determine if that is the case, but we did find that the preferences VSIDS shows for the "special" families of interest here are generally quite stable of an entire run. Figure 2 is illustrative of several experiments we ran that show this. The figure shows the average fraction of three families of special variables over a run. To show full-run data and also the initial "settling down", we use an unconventional x-axis. The left half of the axis shows the values every 100K decisions for the first 1 million decisions. The right half shows the values at 9 points equally spread over the remainder of each run.

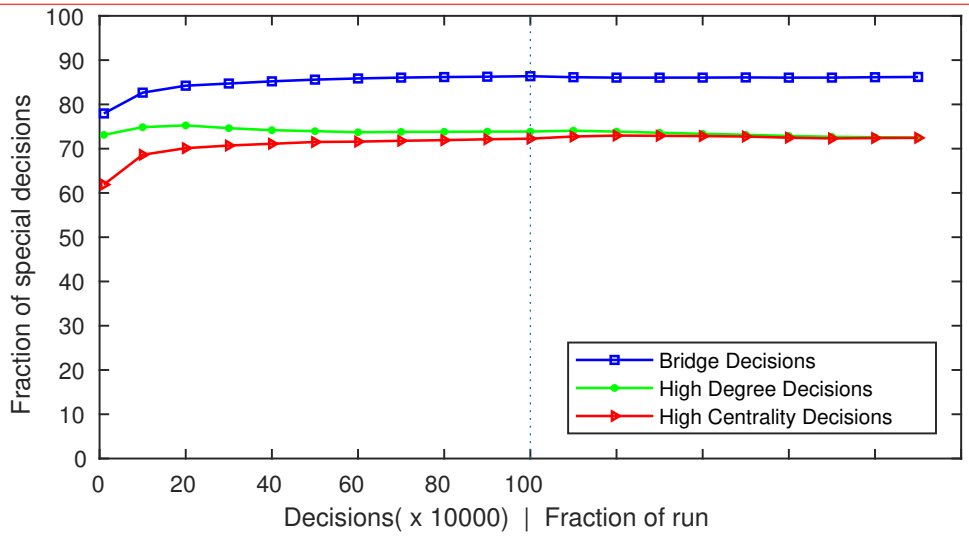


Figure 2: Mean fraction of special decisions over full run. Left half is for the first 1M decisions; right half is the remainder of the run to completion.

3.2 Preferential Bumping

The strong preference shown by VSIDS for certain families of variables led us to wonder whether these preferences are always good or not, and to consider whether we could alter the fractions of these families of decisions, thereby perhaps adjusting for "non-optimal preference". We found that by selectively increasing or decreasing the VSIDS activity bump value for certain variables, we can increase or decrease (respectively) the tendency of VSIDS to pick them. We illustrate this here by showing the fraction of Glucose decisions that are from a chosen family of variables when we increase the bump value for the variables in that family variables. We experimented with a number of preferential bumping schemes, two of which we report here.

Our preferential bumping scheme alters the VSIDS score update factor for a selected set S of variables. For variables not in S, the score update is addition of the default bump value, as described above. We maintain a second "special" bump value, which is initialized to some value b different from 1, and updated in the same manner as the default bump value. This "special bump value" is used to update the scores of variables in S. We verified that, for a number of special families of variables, increasing the special bump value increases the fraction of decisions among the special family, while decreasing the special bump value reduces the fraction of decisions among the special variables.

The two preferential bumping schemes we report data on below are as follows. Having selected a set S of variables we want to influence, and a "special bump value", we do one of:

1. Uniform scheme: Variables from S are bumped using the "special" bump value during the entire run;

2. Initial scheme: Variables from S are bumped by the "special" bump value until 100K decisions have been made, after which they are bumped using the default bump value.

We call Glucose modified with preferential bumping GlPB, and use names of the form GlPB-S-B, where S identifies the set of special variables and B the preferential bumping scheme. S

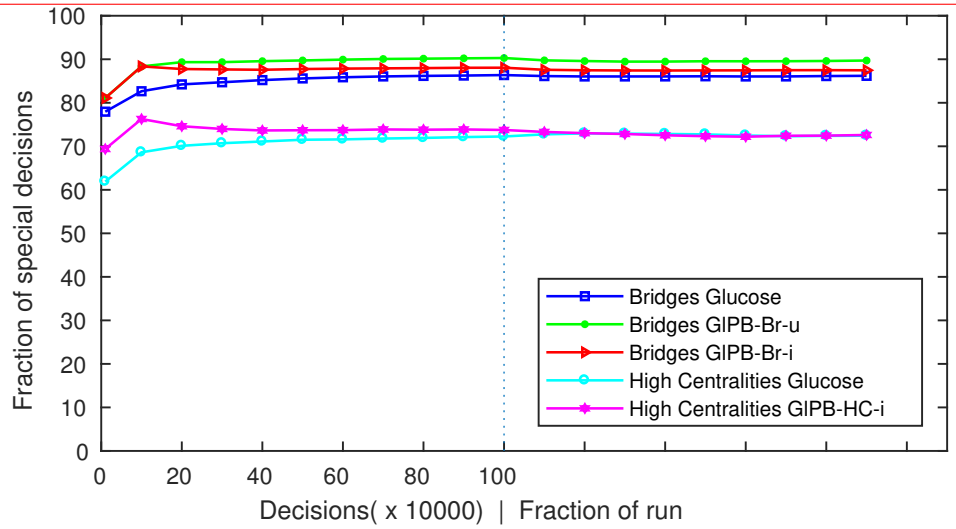


Figure 3: Effect of preferential bumping on the fraction of special decisions.

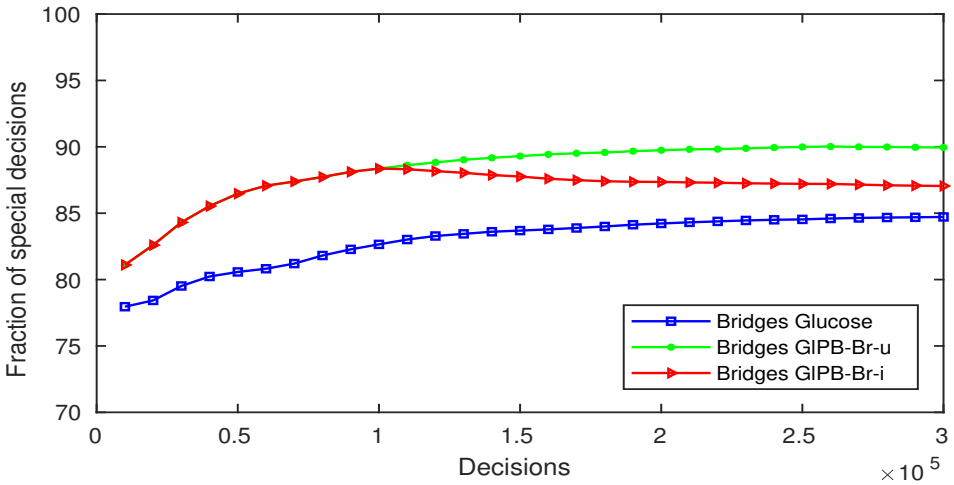


Figure 4: Effect of initial *vs* uniform preferential bumping: magnified inital segmant.

is either HC, for "high centrality variables", or Br for "bridge variables". B is either u for "uniform preferential bumping" or i for "initial preferential bumping". In all the reported experiments, the (initial) special bump value is 1.1.

Figure 3 illustrates the effect of preferential bumping. The two lower lines are fractions of high centrality decisions for default Glucose and for the variant GlPB-HC-i, with initial preferential bumping of high-centrality variables. The three upper lines show the fraction of bridge decisions for default Glucose and for the solvers GlPB-Br-u, with preferential bumping of bridges all the time, and GlPB-Br-i, with initial preferential bumping of bridges. Figure 4 shows a blow-up view of the initial segments of those three curves. Preferential bumping for the entire run increases the fraction of "preferred" decisions for the entire run. With initial preferential bumping, the fraction increases at first, then drops back toward, but never meeting,

the default value.

4 Structure-Based Preferential Bumping

In this section we demonstrate the effectiveness of structure-based preferential bumping in two versions of GlPB (Glucose modified with preferential bumping), on a large set of industrial formulas. We begin with a small motivating example.

4.1 Preferential Bumping of Central Communities

Community graphs of many formulas do not seem to have an easy-to-understand structure, but a number, such as those shown in Figure 5, do. In the figure, the size of nodes reflects community size, and the width of edges reflects the number of bridges between the relevant communities. The formula UR-15-10p0 takes the form of a linear chain-like structure. The formula hwmcc10-timeframe-expansion-k45 takes the form of a path of large communities connected by wide edges, plus a number of small communities connected by narrow edges. We view the dominant linear structures here as being a kind of "coarse structure": It is the apparent structure if we ignore most small details. There are many other formulas with "nice coarse structure" – another example is shown in Figure 1 of [7].

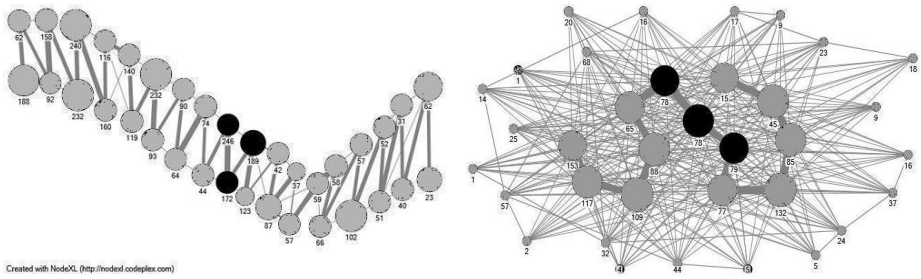


Figure 5: The community graphs of instances hwmcc10-timeframe-expansion-k45 (left) and UR-15-10p0 (right) have a linear "coarse structure". Black nodes are those selected for preferential bumping. (Numeric labels are number of decisions, in thousands, made by Glucose with preferential bumping, within each community.)

For a number of formulas with a clear linear coarse structure, we selected a small number of (visually) central communities, and then increased the Glucose bump value for the variables in these communities. For the formulas hwmcc10-timeframe-expansion-k45 (hwmc..k45 for short) and UR-15-10p0, the selected communities are the shown in black in Figure 5. Table 3 shows, for each of the formulas of Figures 5, the effect of uniform preferential bumping on the fraction of decisions within the chosen central communities, and the corresponding reduction in Glucose run times, which is significant.

These, and similar examples, provided some of the first evidence of success of structure-based preferential bumping, and motivated exploration of a number of notions of centrality in both community and primal graphs.

4.2 Performance of GlPB-HC-i and GlPB-Br-u

In the following, we demonstrate that fully automatic preferential bumping of specific families of structurally determined variables can in fact improve solver performance. In particular, we

Instance	Default Glucose		Glucose + Central Bumping	
	Central Decisions	CPU Time	Central Decisions	CPU Time
hwmc...k45	8.5 %	2147	12.6 %	616
UR-15-10p0.cnf	6.3 %	1293	15 %	749

Table 3: Effect on solving time and fraction of decisions in central communities of increasing the bump value for variables in those communities.

ran default Glucose, GlPB-HC-i and GlPB-Br-u on all of the 742 distinct formulas from the industrial categories of the three most recent (at time of writing) SAT solver competitions (2013, 2014 and 2016) combined. This set includes the 223 formulas used in previous sections of this paper.

Recall that GlPB-Br-u preferentially bumps variables that are bridges in the primal graph for the entire run, and GlPB-HC-i preferentially bumps variables that have high centrality in the primal graph for the first 100K decisions. We allowed up to 50 seconds for community bridge detection in GlPB-Br-u and 200 seconds for centrality computation in GlPB-HC-i, and 5000 seconds total running time (structure computation plus solving) for each formula.

Of the 742 formulas in the benchmark set, 210 were not solved by any of the three solvers. Glucose solved 512 formulas, GlPB-Br-u solved 517, and GlPB-HC-i solved 524.

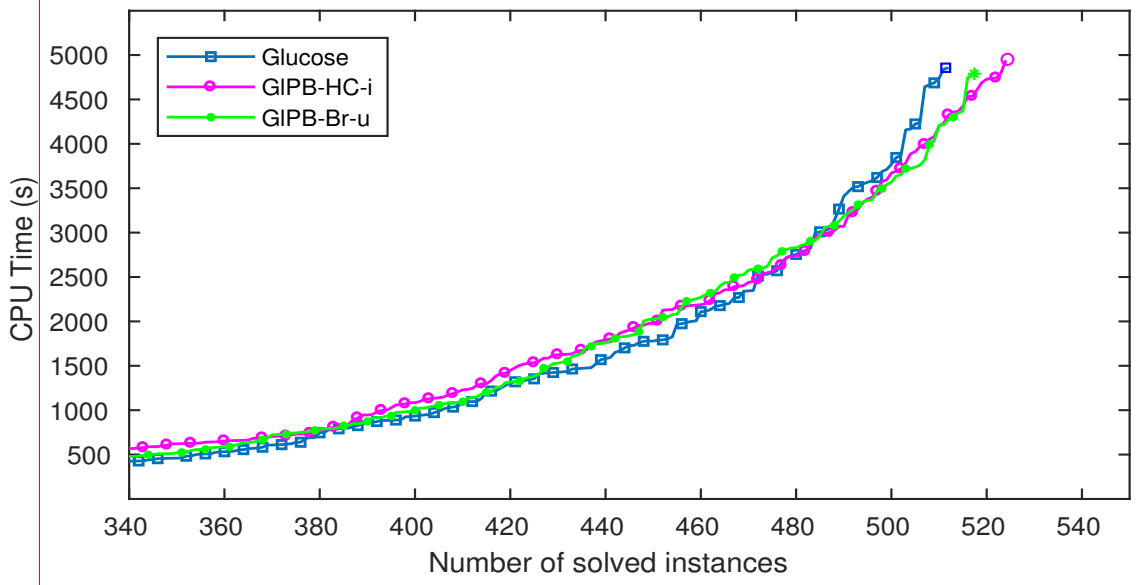


Figure 6: Relative performance of Glucose, GlPB-Br-u and GlPB-HC-i.

Figure 6 is the standard (in the SAT solver literature) "cactus plot", which shows for each solver how many formulas were solved in time at most t, as a function of t. We see that on easier formulas the time spent on structure computation does not pay off, and Glucose performs best. However, the preferential bumping solvers perform better on harder formulas.

5 Discussion

We have introduced a new method, that we call structure-based preferential bumping, for exploiting formula structure in VSIDS-based CDCL solvers. Experiments on a large industrial benchmark set showed it can improve the performance of Glucose, a high-performance CDCL solver. Our experiments (including those presented here and others), suggest that the performance of our versions of Glucose enhanced with structure-based preferential bumping scale better than that of Glucose.

There are many possible parameter settings for GlPB, and we have made very little effort to optimize them: Our choices of 1.1 for the preferential bump value; a 1/3 fraction of variables to be called "high centrality"; and 100K decisions as an "initial segment" are the result of minimal experimentation. Most likely large-scale experiments (in particular using appropriate automated tools, such as SMAC [4]) can improve the performance further.

We continue to explore various approaches to structure-based preferential bumping and other light-weight ways to exploit structure in CDCL solvers. In particular, we are currently experimenting with features of community graphs and of tree decompositions.

Acknowledgements

The work reported here was supported in part by an NSERC Discovery grant to the second author. We thank the anonymous reviewers for suggestions that improved the paper.

References

[1] The Glucose SAT solver. http://www.labri.fr/perso/lsimon/glucose/.

[2] The international SAT competitions web page. http://www.satcompetition.org.

[3] PeneLoPe, a parallel sat solver. http://www.cril.univ-artois.fr/~hoessen/penelope.html.

[4] SMAC: Sequential model-based algorithm configuration. http://www.cs.ubc.ca/labs/beta/Projects/SMAC/.

[5] Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, and Jordi Levy. Community structure in industrial SAT instances. *arXiv preprint*, arXiv:1606.03329, 2016.

[6] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of SAT formulas. In *Proceedings of SAT 2012*, volume 7317 of *LNCS*, pages 410–423. Springer, 2012.

[7] Carlos Ansótegui, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. Using community structure to detect relevant learnt clauses. In *Proceedings of SAT 2015*, volume 9340 of *LNCS*, pages 238–254. Springer, 2015.

[8] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[9] Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In Marijn Heule and Sean Weaver, editors, *Proceedings of SAT 2015*, volume 9340 of *LNCS*, pages 405–422. Springer, 2015.

[10] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *The Journal of statistical mechanics: theory and experiment*, 2008(10):10008, 2008.

[11] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[12] Jia Hui Liang, Vijay Ganesh, Ed Zulkoski, Atulan Zaman, and Krzysztof Czarnecki. Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In *Hardware and Software: Verification and Testing: Haifa Verification Conference*, volume 9434 of *LNCS*, pages 225–241. Springer, 2015.

[13] João P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[14] Ruben Martins, Vasco Manquinho, and Inês Lynce. Community-based partitioning for MaxSAT solving. In Matti Järvisalo and Allen Van Gelder, editors, *Proceedings of SAT 2013*, volume 7962 of *LNCS*, pages 182–191. Springer, 2013.

[15] Robert Mateescu. Treewidth in industrial sat benchmarks. Technical Report MSR-TR-2011-22, Microsoft, February 2011.

[16] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 530–535. ACM, 2001.

[17] Miguel Neves, Ruben Martins, Mikoláš Janota, Inês Lynce, and Vasco Manquinho. Exploiting resolution-based representations for MaxSAT solving. In Marijn Heule and Sean Weaver, editors, *Proceedings of SAT 2015*, volume 9340 of *LNCS*, pages 272–286. Springer, 2015.

[18] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[19] Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, Gilles Audemard, and Laurent Simon. Impact of community structure on SAT solver performance. In *Proceedings of SAT 2014*, volume 8561 of *LNCS*, pages 252–268. Springer, 2014.

[20] Tomohiro Sonobe, Shuya Kondoh, and Mary Inaba. Community branching for parallel portfolio SAT solvers. In Carsten Sinz and Uwe Egly, editors, *Proceedings of SAT 2014*, volume 8561 of *LNCS*, pages 188–196. Springer, 2014.