Centrality Heuristics for Exact Model Counting

Bernhard Bliem and Matti Järvisalo
HIIT, Department of Computer Science, University of Helsinki, Finland

***Abstract*—Model counting is the archetypical #P-complete problem consisting of determining the number of satisfying truth assignments of a given propositional formula. In this short paper, we empirically investigate the potential of employing graph centrality measures as a basis of search heuristics in the context of exact model counting. In particular, we integrate centrality-based heuristics into the search-based exact model counter sharpSAT. Our experiments show that employing centrality information significantly improves the empirical performance of sharpSAT, and also allows for simplifying the search heuristics compared to the current default heuristics of the model counter. In particular, we show that the VSIDS heuristic, which is an integral search heuristic employed in essentially all state-of-the-art conflict-driven clause learning Boolean satisfiability solvers, appears to be of very limited use in the context of model counting.**

***Index Terms*—exact model counting, #SAT, search heuristics, graph centrality, betweenness centrality**

I. Introduction

Given a propositional formula φ in conjunctive normal form (CNF), the model counting problem consists of counting the number of satisfying truth assignments of φ [1]. This problem, also known as #SAT, is well known to be #P-complete [2]. Thus model counting is both an extremely challenging reasoning problem and captures a wide range of AI and KR problems, including probabilistic reasoning [3]–[6], planning [7], [8], circuit design [9], quantitative information flow analysis [10], differential cryptanalysis [11], and inconsistency measurement [12]. Recent advances on exact algorithms for model counting are making progress towards enabling exact model counting on CNF formulas of increasing size. However, due to the high complexity of the problem, there is a need for improving model counters in order to handle larger formulas towards practical applications.

The current state-of-the-art approaches to exact model counting can be divided into search-based [13]–[17] and compilation-based [18]–[23] approaches. Here we focus on the search-based approach, which builds on the extraordinary success of conflict-driven clause learning (CDCL) [24]–[26] Boolean satisfiability (SAT) solvers by combining key algorithmic ideas of CDCL solvers with component caching of subformulas encountered during the model counting process [27].

In this short paper, we focus on search heuristics for SAT-based exact model counters. In particular, we lift recent work [28], which investigated speeding up CDCL SAT solvers by employing a graph-based measure called *betweenness centrality* [29] on an underlying graph structure of SAT instances, to the realm of exact model counting. For some intuition, as noted e.g. in [30], nodes with high betweenness centrality intuitively may represent variables which are "in-between" communities in the underlying community structure [31] of real-world constraint networks (and CNF formulas [32], [33]). In particular, branching on highly central variables can thus lead to "nicely" dividing CNF formulas into components.

To this end, we study the impact of coupling heuristics employed in the state-of-the-art search-based model counter sharpSAT [14] with knowledge of betweenness centrality of variables in model counting instances. We do so by investigating variants of the VSADS branching heuristic of sharpSAT—which combines variable occurrence information with the standard VSIDS decision heuristic applied in CDCL solvers [25]—boosted with betweenness centrality information computed from the so-called primal graph of the input instances. As the main findings, we empirically show that making use of betweenness centrality information noticeably improves the performance of sharpSAT, and also allows for simplifying the search heuristics compared to the current default heuristics of the model counter. In particular, we also show that the VSIDS heuristic, which is an integral search heuristic employed in essentially all state-of-the-art conflict-driven clause learning Boolean satisfiability solvers, appears to be of very limited use in the context of model counting.

In terms of the most related work, the recent application of betweenness centrality in boosting CDCL SAT solvers [28] report most successful when integrating centrality into the clause forgetting mechanism. Beyond SAT, betweenness centrality has been recently studied as a basis for top-down compilation of finite-domain constraint networks into decomposable multi-valued decision graphs (MDDGs) [30]. In constrast, here we consider betweenness centrality as a means of noticeably simplifying the decision heuristics of search-based model counters (for #SAT). We show that using centrality values computed only once at the beginning of the model counting search can already be beneficial, whereas [30] recompute betweenness centrality values at each step of a top-down compilation process. Finally, we note that our empirical observation that VSIDS is not an impactful heuristic in the context of search-based model counting agrees with recent observations on the limited applicability of VSIDS as a way to decompose search into subspaces in the context of parallel SAT solving [34].

II. Search-Based Exact Model Counting

Search-based exact model counters build on central search techniques applied in SAT solvers. In this work, we focus on the sharpSAT model counter [14] as a state-of-the-art search-based model counter. In particular, following CDCL, sharpSAT employs unit propagation as the main constraint propagation

mechanism during search and a decision heuristic for extending partial assignments. When it derives a conflict, it uses clause learning to infer a conflict clause that is logically entailed by the input formula and compactly enforces the fact that the current partial assignment cannot be extended to a satisfying truth assignment (model) of the input formula. Whereas CDCL SAT solvers apply conflict-driven backtracking after each learned clause based on the conflict clause, model counting imposes restrictions on the backtracking mechanism to more classical backtracking in order to guarantee that an exact model count is obtained [13].

Decision heuristics of search-based exact model counters are the focus of the current work, and we will detail the decision heuristics employed in sharpSAT separately in Section IV.

Like the model counter *Cachet* [27] before it, sharpSAT implements *component caching*, which uses the fact that we can represent a formula φ by its *primal graph*, i.e., the graph whose vertices are the variables of φ and that has an edge between two variables if they occur together in some clause; see Fig. 1 for an example. During search, this graph (and thus the formula) may split up into several connected components, and sharpSAT computes the model count for each component separately. Component caching tries to avoid re-computing the model count for already seen components.

III. Betweenness Centrality

As a main contribution, building on recent work on improving SAT solvers through centrality measures [28], we propose the use of so-called betweenness centrality [29] as a basis for improving the decision heuristics employed in search-based model counters.

For describing the relevant concepts, in the following we will consider undirected, simple graphs. When we speak of paths, we assume them to be undirected in the sense that $\langle v_1, \ldots, v_n \rangle$ is the same path as $\langle v_n, \ldots, v_1 \rangle$.

The notion of betweenness centrality [29] was originally proposed in the context of social networks. The basic idea is that some nodes have a higher potential to control communication in a network than others. Given three distinct nodes s, t, v in a network, if many of the shortest paths between s and t go over v, then v has a high potential for controlling communication between s and t. Betweenness centrality formalizes this by assigning a global score $\mathrm{centr}_G(v)$ to each node v of a graph G as

$$\mathrm{centr}_G(v) = \sum_{\{s,t\} \in \mathrm{pairs}(G)} \frac{\sigma(s, t \mid v)}{\sigma(s, t)},$$

where $\mathrm{pairs}(G)$ contains all distinct unordered pairs of connected vertices in G, $\sigma(s,t)$ is the number of shortest paths between s and t, and $\sigma(s, t \mid v)$ is the number of such paths that go through v; by "go through" we mean that the path contains v but neither starts nor ends at v.

Example 1: Consider the formula and primal graph G depicted in Fig. 1. No shortest path goes through a or e, hence $\mathrm{centr}_G(a) = \mathrm{centr}_G(e) = 0$. Next, observe that the shortest paths that go through b are exactly $\langle a, b, c \rangle$, $\langle a, b, d \rangle$, $\langle a, b, c, e \rangle$, $\langle a, b, d, e \rangle$. Moreover, $\sigma(a, c) = \sigma(a, d) = 1$ and $\sigma(a, e) = 2$. Hence we have $\mathrm{centr}_G(b) = \frac{1}{1} + \frac{1}{1} + \frac{2}{2} = 3$. Using similar reasoning, we can observe for c (and analogously for d) that the centrality score is $\mathrm{centr}_G(c) = \frac{1}{2} + \frac{1}{2} = 1$.

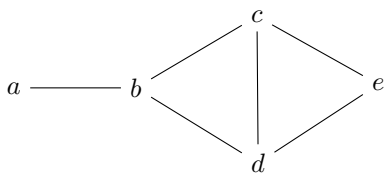

Fig. 1. Primal graph of the formula $(a \vee b) \wedge (\neg b \vee c \vee \neg d) \wedge (c \vee d \vee e)$

IV. Centrality-Based Heuristics for Model Counting

Whenever sharpSAT assigns variables, it splits up (the primal graph of) the formula under consideration, and it computes the model count for each resulting component separately. For deciding which variable to assign, sharpSAT uses the following heuristic called VSADS. For each variable v in the current component, it computes the value $\mathrm{score}(v)$ as

$$\mathrm{score}(v) = \mathrm{freq}(v) + 10 \cdot (\mathrm{act}(v) + \mathrm{act}(\neg v)), \qquad (1)$$

where

- the *frequency score* $\mathrm{freq}(v)$ denotes the number of (positive or negative) occurrences of v in the residual formula (i.e., the current component); and
- the *activity score* $\mathrm{act}(\ell)$, for any literal ℓ, is similar to the VSIDS score[1] of ℓ.

In other words, the only difference to VSIDS is that $\mathrm{act}(\ell)$ is not initialized to 0 but to the number of occurrences of ℓ in the original formula (after some basic preprocessing). After computing the score for each variable of the current component, sharpSAT will choose a variable with maximum score and assign it either true of false, depending on which literal has the higher activity score.

We propose to integrate betweenness centrality into the variable scores. Specifically, we compute for each variable v the score

$$\mathrm{score}(v) = \mathrm{freq}(v) + 10 \cdot (\mathrm{act}(v) + \mathrm{act}(\neg v)) + w \cdot \mathrm{centr}_G(v), \qquad (2)$$

where G is the primal graph of the original formula, $\mathrm{centr}_G(v)$ is the betweenness centrality score of v, and w is some weight that normalizes the centrality score so that it is at most the number of clauses in the original formula. The motivation for this normalization is to have centrality scores that potentially have the same range as the frequency scores. In preliminary experiments, we also tested different weights for this and the other scores, but found that tweaking the weights does not affect performance greatly.

[1] Whenever a clause C is learned during CDCL-style search, VSIDS increments the scores of the variables in C by 1, and it periodically decreases all scores by a fixed amount, e.g., by multiplying them with 0.95.

As in [28], we use the NetworkX library [35] to compute the betweenness centrality scores. We do this once before solving begins (after sharpSAT's basic preprocessing). As exact centrality scores may not be necessary for a useful heuristic, especially for larger formulas it can be advisable to compute approximate centrality scores. When asked to produce exact values, NetworkX computes for each node s all shortest paths originating from s. By using only a part of the vertices (chosen uniformly at random) as origins for shortest paths, it also offers the possibility to approximate centrality.

In contrast to [28], who always approximate centrality scores with $\frac{n}{50}$ nodes, where n is the number of variables, we use a more fine-grained criterion: If the graph has 400 nodes or less, we compute the exact value, otherwise we approximate it using between 400 and 800 nodes (proportional to the number of variables). To be precise, we use $800 - \frac{400^2}{n}$, where n is the number of variables. However, this exact scheme and the exact numbers are not crucial for our approach. Nevertheless, we chose 400 as the number of nodes until which exact values are computed because the computation then takes very little time in practice. The upper bound of 800 for the number of nodes for approximation was chosen because the computation times are still reasonable and further increases only led to slightly better values that did not seem to be worth the additional computation time. Hence increasing these values will most likely have almost no effect on the performance of the model counter. We did not encounter any issues with memory in the computation of the centrality scores.

V. Experiments

We empirically evaluate the impact of integrating betweenness centrality into the decision heuristics of sharpSAT on a range of typical exact model counting benchmarks. We also evaluate the importance of the other heuristic components of sharpSAT with and without using betweenness centrality, and report on our findings in the following.

For the evaluation, we used the same set of benchmark instances as in [20], including a wide selection of instances used for evaluating model counters in earlier papers [13], [14], [16], [27]; see Table I. The experiments were run under CentOS 7 Linux using Intel Xeon E5-2680 v4 2.4 GHz CPUs with 256 GB RAM. Each run was limited to 30 minutes of CPU time and 10 GB of memory. The maximum size for the component cache in sharpSAT was set to 8 GB.

We observed that the centrality scores are usually fast to compute compared to the solving times. Among the instances that sharpSAT (default configuration) solved within the resource limits, the average time for computing centrality scores was 8 seconds, and over all instances 10 seconds. On 22 instances, centrality computation times out, but no other contestant could solve those instances either.[2]

In the experiments, we considered the following variants of the sharpSAT decision heuristics.

- **default:** The default decision heuristics of sharpSAT (Eq. 1 in Section IV).
- **w/centrality:** The default decision heuristics of sharpSAT with (possibly approximated) betweenness centrality included as an additive term (Equation 2).
- **w/centrality w/o activity:** *w/centrality* with activity scores removed.
- **w/centrality w/o freq:** *w/centrality* with frequency scores removed.
- **centrality only:** *w/centrality* with activity and frequency scores removed.
- **freq only:** *default* with activity scores removed.
- **activity only:** *default* with frequency scores removed.
- **VSIDS only:** like *activity only*, but all activity scores initialized to 0.

An overview of the results is presented in Fig. 2 , which shows the number of solved instances (x-axis) in terms of the per-instance time limit (y-axis). As a first observation, note that *default* and *freq only* exhibit essentially the same performance. This interestingly suggests that the activity-score heuristic, which is a variant of VSIDS, is not a major contributor to the performance of sharpSAT; furthermore, *VSIDS only* exhibits noticeably worse performance. This suggests that solely using VSIDS as the heuristic is far from a viable option, which is in stark contrast to modern SAT solvers where VSIDS is widely recognized and employed as the decision heuristic.

Centrality only, on the other hand, exhibits significantly better performance than *default*, *freq only*, *activity only* and *VSIDS only*. Utilizing all of centrality, frequency and activity scores together seems to only slightly improve performance compared to *centrality only*. We conclude that betweenness centrality on its own yields both a significantly simpler and more effective heuristic.

Fig. 3 gives a per-instance runtime comparison of the default sharpSAT heuristics (y-axis) and the default heuristics boosted with betweenness centrality. This further corroborates the benefits of integrating centrality information into the default heuristics of sharpSAT.

More detailed results per each benchmark domain are shown in Table I. We observe that integrating centrality information into sharpSAT significantly improves performance almost across the board. For a comparison with a state-of-the-art compilation-based model counter, the table also includes results for the recent model counter *D4* [20]. While overall, and especially in the *Bayesian network* domain, the more recent compilation-based model counter D4 is still faster, adding centrality information into sharpSAT narrows the gap and makes the search-based approach competitive in several domains. As compilation-based model counters, including D4, are also based on computing a DPLL SAT solver trace, we find it likely that centrality-based heuristics have potential for further speeding up compilation-based approaches as well. However, integration of centrality-based heuristics would require access

[2]For such instances, one could approximate centrality over a smaller sample.

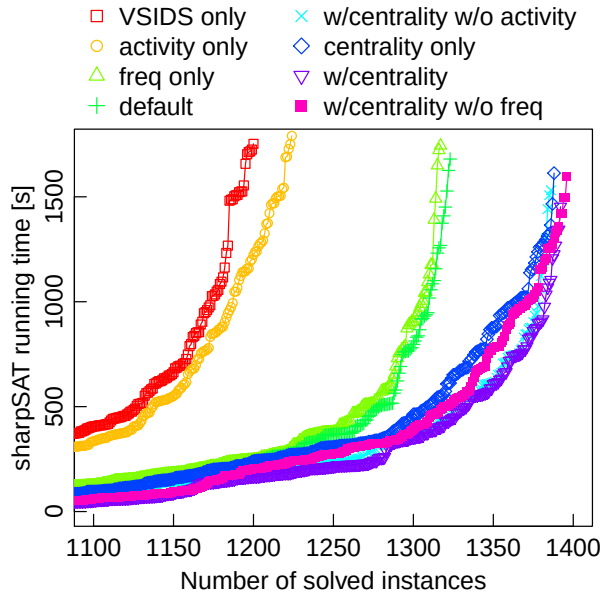

Fig. 2. The number of instances solved by sharpSAT using the different heuristic variants.

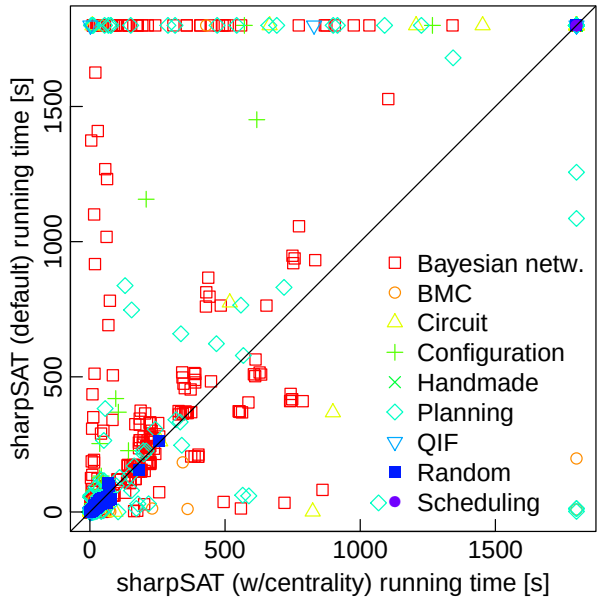

Fig. 3. Per-instance runtime comparison between sharpSAT using its default heuristic with and without betweenness centrality.

to the source code of D4.[3]

Finally, we note that the most successful technique reported in the paper that applies centrality to SAT [28] was integrating centrality into the clause forgetting mechanism. We also investigated taking centrality into account for forgetting clauses in sharpSAT. By default, it deletes the clauses that are most seldom used for unit propagation. As in [28], instead of using the activity in unit propagation as the score of a clause, we used the average centrality of the variables occurring in it. However, preliminary experiments showed that this only gives a marginal benefit, if any; this is potentially due to the fact that clause forgetting does not appear to play a central role—compared to the very central component caching—in exact model counters, which is in stark contrast with the case of CDCL SAT solving.

[3]Unfortunately we have not been able to obtain the source code of D4 despite our efforts to contact the authors.

VI. Conclusions

We proposed employing betweenness centrality as the basis of decision heuristics for search-based exact model counters, motivated by the recent promising results on employing centrality measures in SAT solvers [28]. We showed that the performance of sharpSAT, a state-of-the-art implementation of a search-based exact model counter, noticeably improves when integrating the centrality measure into its default heuristics. We also showed that, in contrast to SAT solving, the VSIDS decision heuristic plays—somewhat surprisingly—a negligible role in improving the performance of sharpSAT.

In terms of future work, the current results motivate studying the integration of betweenness centrality also for evaluating the components cached during search; centrality could make a difference in terms of component forgetting, i.e., in heuristically choosing which cached component to forget when the component cache runs out of memory during the counting process. Graph representations other than primal graphs could be considered as an alternative basis for computing the centrality scores. We hope that our observations motivate further research on the non-trivial question of what properties betweenness centrality captures in CNF formulas as the underlying reason for the promising results on centrality-based heuristics in both SAT solving and model counting. Finally, we note that the recent empirical study [30] took related first steps on the relative impact of betweenness centrality and hypergraph partitioning as basis for heuristics in the context top-down compilation of finite-domain constraint networks into decomposable multi-valued decision graphs. We believe it could be insightful to study in-depth—both empirically as well as more theoretically towards a more rigorous understanding of—the relationship between graph centrality measures and graph partitioning techniques such as the one employed in the D4 compiler in the focused context of #SAT.

Acknowledgments

This work has been financially supported by Academy of Finland via grants 276412 and 312662.

References

[1] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 633–654.

[2] L. G. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. 8, pp. 189–201, 1979.

[3] D. Roth, "On the hardness of approximate reasoning," *Artificial Intelligence*, vol. 82, no. 1-2, pp. 273–302, 1996.

[4] T. Sang, P. Beame, and H. A. Kautz, "Performing bayesian inference by weighted model counting," in *Proc. AAAI*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press / The MIT Press, 2005, pp. 475–482.

TABLE I

COMPARISON OF SHARPSAT HEURISTICS (DEFAULT VS. CENTRALITY INTEGRATED): NUMBER OF SOLVED INSTANCES (**#SOL**) AND THE CUMULATIVE RUNNING TIMES OVER SOLVED INSTANCES (**CUM. T**) IN SECONDS PER BENCHMARK INSTANCE DOMAIN. DATA FOR THE RECENTLY PROPOSED COMPILATION-BASED MODEL COUNTER D4 [20] IS ALSO INCLUDED.

		default		**w/centrality**		**D4**	
Domain	**#instances**	**#sol**	**(cum. t)**	**#sol**	**(cum. t)**	**#sol**	**(cum. t)**
Bayesian Network	1116	671	69467	718	69276	968	77035
BMC	18	12	479	12	1617	15	1091
Circuit	65	51	1463	55	6102	56	5457
Configuration	35	33	4437	35	3274	32	4428
Handmade	68	34	154	34	242	35	2082
Planning	557	429	14103	444	19178	458	43900
QIF	7	4	5	6	843	6	831
Random	104	102	2382	102	2615	102	1336
Scheduling	6	0	0	0	0	0	0
Total	1976	1336	92490	1406	103147	1672	136160

[5] M. Chavira and A. Darwiche, "On probabilistic inference by weighted model counting," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 772–799, 2008.

[6] U. Apsel and R. I. Brafman, "Lifted MEU by weighted model counting," in *Proc. AAAI*, J. Hoffmann and B. Selman, Eds. AAAI Press, 2012.

[7] H. Palacios, B. Bonet, A. Darwiche, and H. Geffner, "Pruning conformant plans by counting models on compiled d-DNNF representations," in *Proc. ICAPS*, S. Biundo, K. L. Myers, and K. Rajan, Eds. AAAI, 2005, pp. 141–150.

[8] C. Domshlak and J. Hoffmann, "Probabilistic planning via heuristic forward search and weighted model counting," *Journal of Artificial Intelligence Research*, vol. 30, pp. 565–620, 2007.

[9] J. Burchard, D. Erb, and B. Becker, "Characterization of possibly detected faults by accurately computing their detection probability," in *Proc. DATE*. IEEE, 2018, pp. 385–390.

[10] F. Biondi, M. A. Enescu, A. Heuser, A. Legay, K. S. Meel, and J. Quilbeuf, "Scalable approximation of quantitative information flow in programs," in *Proc. VMCAI*, ser. Lecture Notes in Computer Science, I. Dillig and J. Palsberg, Eds., vol. 10747. Springer, 2018, pp. 71–93.

[11] S. Kölbl, G. Leander, and T. Tiessen, "Observations on the SIMON block cipher family," in *Proc. CRYPTO*, ser. Lecture Notes in Computer Science, R. Gennaro and M. Robshaw, Eds., vol. 9215. Springer, 2015, pp. 161–185.

[12] M. Thimm and J. P. Wallner, "Some complexity results on inconsistency measurement," in *Proc. KR*, C. Baral, J. P. Delgrande, and F. Wolter, Eds. AAAI Press, 2016, pp. 114–124.

[13] T. Sang, P. Beame, and H. A. Kautz, "Heuristics for fast exact model counting," in *Proc. SAT*, ser. Lecture Notes in Computer Science, F. Bacchus and T. Walsh, Eds., vol. 3569. Springer, 2005, pp. 226–240.

[14] M. Thurley, "sharpSAT - counting models with advanced component caching and implicit BCP," in *Proc. SAT*, ser. Lecture Notes in Computer Science, A. Biere and C. P. Gomes, Eds., vol. 4121. Springer, 2006, pp. 424–429.

[15] J. Davies and F. Bacchus, "Using more reasoning to improve #SAT solving," in *Proc. AAAI*. AAAI Press, 2007, pp. 185–190.

[16] R. J. Bayardo, Jr. and J. D. Pehoushek, "Counting models using connected components," in *Proc. AAAI*, H. A. Kautz and B. W. Porter, Eds. AAAI Press / The MIT Press, 2000, pp. 157–162.

[17] E. Birnbaum and E. L. Lozinskii, "The good old Davis-Putnam procedure helps counting models," *Journal of Artificial Intelligence Research*, vol. 10, pp. 457–477, 1999.

[18] A. Darwiche, "New advances in compiling CNF into decomposable negation normal form," in *Proc. ECAI*, R. L. de Mántaras and L. Saitta, Eds. IOS Press, 2004, pp. 328–332.

[19] F. Koriche, J. Lagniez, P. Marquis, and S. Thomas, "Knowledge compilation for model counting: Affine decision trees," in *Proc. IJCAI*, F. Rossi, Ed. IJCAI/AAAI, 2013, pp. 947–953.

[20] J. Lagniez and P. Marquis, "An improved decision-DNNF compiler," in *Proc. IJCAI*, C. Sierra, Ed. ijcai.org, 2017, pp. 667–673.

[21] C. J. Muise, S. A. McIlraith, J. C. Beck, and E. I. Hsu, "Dsharp: Fast d-DNNF compilation with sharpSAT," in *Proc. AI*, ser. Lecture Notes in Computer Science, L. Kosseim and D. Inkpen, Eds., vol. 7310. Springer, 2012, pp. 356–361.

[22] A. Darwiche and P. Marquis, "A knowledge compilation map," *Journal of Artificial Intelligence Research*, vol. 17, pp. 229–264, 2002.

[23] J. Lagniez, E. Lonca, and P. Marquis, "Improving model counting by leveraging definability," in *Proc. IJCAI*, S. Kambhampati, Ed. IJCAI/AAAI Press, 2016, pp. 751–757.

[24] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[25] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. DAC*. ACM, 2001, pp. 530–535.

[26] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT 2003 Selected Revised Papers*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Springer, 2003, pp. 502–518.

[27] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi, "Combining component caching and clause learning for effective model counting," in *SAT 2004 Online Proceedings*, 2004.

[28] S. Jamali and D. Mitchell, "Centrality-based improvements to CDCL heuristics," in *Proc. SAT*, ser. Lecture Notes in Computer Science, O. Beyersdorff and C. M. Wintersteiger, Eds., vol. 10929. Springer, 2018, pp. 122–131.

[29] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

[30] J. Lagniez, P. Marquis, and A. Paparrizou, "Defining and evaluating heuristics for the compilation of constraint networks," in *Proc. CP*, ser. Lecture Notes in Computer Science, J. C. Beck, Ed., vol. 10416. Springer, 2017, pp. 172–188.

[31] M. Girvan and M. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, pp. 7821–7826, 2002.

[32] C. Ansótegui, J. Giráldez-Cru, and J. Levy, "The community structure of SAT formulas," in *Proc. SAT*, ser. Lecture Notes in Computer Science, A. Cimatti and R. Sebastiani, Eds., vol. 7317. Springer, 2012, pp. 410–423.

[33] Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, and L. Simon, "Impact of community structure on SAT solver performance," in *Proc. SAT*, ser. Lecture Notes in Computer Science, C. Sinz and U. Egly, Eds., vol. 8561. Springer, 2014, pp. 252–268.

[34] L. L. Frioux, S. Baarir, J. Sopena, and F. Kordon, "Modular and efficient divide-and-conquer SAT solver on top of the painless framework," in *Proc. TACAS*, ser. Lecture Notes in Computer Science, T. Vojnar and L. Zhang, Eds., vol. 11427. Springer, 2019, pp. 135–151.

[35] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.