

Computation of Renameable Horn Backdoors^{*}

Stephan Kottler, Michael Kaufmann, and Carsten Sinz

Eberhard Karls Universität Tübingen, Wilhelm-Schickard-Institute, Tübingen,
Germany

Abstract. Satisfiability of real-world SAT instances can be often decided by focusing on a particular subset of variables - a so-called Backdoor Set. In this paper we suggest two algorithms to compute Renameable Horn deletion backdoors. Both methods are based on the idea to transform the computation into a graph problem. This approach could be used as a preprocessing to solve hard real-world SAT instances. We also give some experimental results of the computations of Renameable Horn backdoors for several real-world instances.

1 Introduction

It is a well known phenomenon that SAT instances evolving from industrial applications can be solved much faster than this could be expected from the theoretical point of view. This allows current state-of-the-art solvers for dealing with instances that consist of up to hundreds of thousands variables. To decide satisfiability for industrial instances it is often sufficient to focus on a particular and primarily small subset of variables - a so-called *backdoor set*. In the groundbreaking work [20] Williams, Gomes and Selman already gave examples of instances with approximately 6,700 variables and nearly 440,000 clauses that exhibit backdoor sets with only 12 variables. Ruan, Kautz and Horvitz showed empirically that an extension of the concept of backdoor sets is a good predictor for the hardness of SAT problems [16]. Moreover, Interian showed that random 3-SAT instances exhibit backdoor sets with 30% to 65% of all variables [10].

Knowing a small backdoor set for an instance in advance could speed up the solving process extraordinarily. However, according to the work of Szeider [18], it is in general not possible to decide in reasonable time whether a given SAT instance exhibits a backdoor with limited size with respect to a DPLL based subsolver (see [7, 6]). Throughout this paper we consider a variant of strong backdoors (see [20]), so-called *deletion backdoors* [13, 19]:

A backdoor is defined with respect to a base class \mathcal{C} of formulas that can be recognized and solved in polynomial time. $\mathcal{B} \subset \mathcal{V}$ is a *deletion backdoor* if the formula $F - \mathcal{B}$ belongs to \mathcal{C} , where $F - \mathcal{B}$ denotes the result of removing all occurrences (both positive and negative) of the variables in \mathcal{B} from the clauses of formula F .

^{*} This work was partly supported by DFG-SPP 1307, project “Structure-based Algorithm Engineering for SAT-Solving”

Nishimura, Ragde and Szeider proved that every deletion backdoor is a strong backdoor, if the base class \mathcal{C} is *clause-induced* ($F \in \mathcal{C} \Rightarrow F' \in \mathcal{C}$ for all $F' \subseteq F$) [13]. For the computation of backdoors with base class Horn and 2-SAT the same authors proved fixed-parameter tractability. Hence, the question whether a formula exhibits a Horn backdoor (respectively a Binary backdoor) with at most k variables can be answered in time that is only exponential in k but not in the number of variables [12]. Moreover, Interian approximated backdoors with respect to the base classes Horn and 2-SAT for random 3-SAT instances [10].

In this article we study the computation of Renameable Horn backdoors. Thus, the base class \mathcal{C} is Renameable Horn. A formula is Horn, if every clause contains at most one positive literal and it is Renameable Horn (RHorn) if it can be renamed to a Horn formula by flipping the literals of some variables. Paris et al. used a two phase approach to compute RHorn backdoors as a pre-processor in a modification of the zChaff SAT solver [15]. In a first step the algorithm tries to increase the number of Horn clauses by flipping the literals of some variables in a local search manner. Secondly, variables are chosen for the backdoor in a greedy fashion to make all non-Horn clauses become Horn. In a recent work Dilkina, Gomes and Sabharwal formulated linear programs to compute optimal RHorn backdoors [9]. An important result is that smallest RHorn backdoors can be exponentially larger than general strong backdoors.

2 Two Approaches to compute RHorn Backdoors

The computation of RHorn backdoors of both approaches presented in this article is based on an equivalent graph problem. The second approach approximates the minimum RHorn backdoor with an approximation ratio that is equal to the size of a so-called *conflict loop* in the graph. Furthermore, when using the second algorithm, the exact approximation ratio is known as soon as the RHorn backdoor is computed. In the following subsection we briefly describe how to transform the problem of finding RHorn backdoors to a problem on directed graphs.

Renameable Horn Backdoors as Graph Problem For a given formula F we create a so-called *dependency graph* $G = (V_G, E_G)$ with $2 * |\mathcal{V}|$ vertices. Each variable v_i entails two vertices k_i^0 and k_i^1 that represent the facts that variable v_i has to be renamed (k_i^0) respectively must not be renamed (k_i^1) in order to make F a Horn formula. The directed edges of G represent the implications of renaming or not renaming variables, according to the clauses of F . A RHorn dependency graph can be created in time $O(m * \text{size_of_max_clause}^2)$ by traversing all possible pairs of literals for each of overall m clauses.

Lewis introduced a method to decide whether a given formula F belongs to class Renameable Horn [11]. The conditions that have to be satisfied to rename F to a Horn formula are formulated as a 2-SAT instance S . It was proved that F is Renameable Horn iff S is satisfiable [11]. The described dependency graph corresponds to the implication graph in [3], that could be used to solve the 2-SAT instance S . In difference to the algorithm in [3] our computations do not

deal with strongly connected components. However, the following properties of implication graphs that are needed for our algorithms can be found in [3, 4, 14] or derived straightforwardly from these results.

Definition 1. We call a vertex k_i^q ($q \in \{0, 1\}$) a *conflict vertex* if there is a path from k_i^q to $k_i^{(q \oplus 1)}$. A variable $x_i \in \mathcal{V}$ has a *conflict loop* if k_i^0 and k_i^1 are both conflict vertices.

Corollary 1. If there is no path from k_i^q to $k_i^{(q \oplus 1)}$ then none of the vertices that can be reached from k_i^q is a conflict vertex.

Lemma 1. A formula F is *Renameable Horn* iff there exists no variable that has a conflict loop in the dependency graph.

Corollary 2. If variable $x_i \in \mathcal{V}$ does not have a conflict loop than neither vertex k_i^0 nor vertex k_i^1 can be involved in a conflict loop of any other variable.

According to Lemma 1 the task to compute a RHorn backdoor can be accomplished by destroying all conflict loops in the appropriate dependency graph. In particular, we aim to delete a minimal amount of variables from the Boolean formula such that the deletion of the according vertices and their incident edges results in a dependency graph without any conflict loops. We call the set of variables involved in a conflict loop a *conflict set*. It is important to notice that a conflict loop and its conflict set do not necessarily have to have the same size.

A heuristic to destroy all conflict loops The first approach mainly considers small conflict sets and variables that occur in many of these conflict sets. The implementation is based on the function `COMPUTECONFLICTSETS(G, \mathcal{U})` that computes one conflict set for each variable in $\mathcal{U} \subseteq \mathcal{V}$ with respect of the dependency graph G . For each variable x_i in \mathcal{U} a conflict loop is computed by checking whether there is a path from vertex k_i^0 to k_i^1 and vice versa. All variables that occur in one of the two computed conflict paths constitute the conflict set for variable x_i . If, on the other hand there is no path from k_i^q to $k_i^{(q \oplus 1)}$ ($q \in \{0, 1\}$) then we know by Corollary 1 that none of the vertices $R_i \subseteq V_G$ that can be reached from vertex k_i^q can be a conflict vertex. By Corollary 2 we can disregard the according variables (\mathcal{R}) of the vertices in R_i for the remaining computation. Thus, for each variable in \mathcal{R} both representing vertices and their incident edges can be deleted from the dependency graph.

The entire computation of a RHorn backdoor starts with creating the dependency graph and computes a small conflict set for each variable of the formula. It starts with an empty backdoor set \mathcal{B} and chooses greedily one variable for \mathcal{B} that occurs most frequently in all known conflict sets S . Ties are broken in favor of variables that occur in small conflict sets. We have applied different strategies to choose a variable for the backdoor, but none of them clearly outperformed the described one. The according vertices of the chosen variable and their incident edges are removed from the dependency graph. New conflict sets are then computed for those variables whose conflict loops were destroyed. At this point the graph may shrink rapidly for some instances, due to the simplification rules in procedure `COMPUTECONFLICTSETS`. The algorithm terminates as soon as all conflict loops are destroyed.

Approximating minimal Renameable Horn Backdoors The second approach to compute Renameable Horn backdoors basically adapts the idea of [8] to approximate a weighted FEEDBACK VERTEX SET in a directed graph. The Algorithm is divided into two phases. In the first phase conflict loops in the graph are destroyed by always taking all related variables of a chosen conflict loop for the backdoor. In the second phase the algorithm tries to shrink the backdoor by reinserting the related vertices and edges of some backdoor variables into the graph. Since an optimal backdoor has to contain at least one variable of any conflict set, it is evident that always taking all variables of any remaining conflict set into the backdoor (phase 1) already approximates the optimal RHorn backdoor by a factor that is smaller or equal to the size of the biggest chosen conflict set. Due to the fact that in the second phase of the algorithm the found backdoor can be only improved the approximation ratio applies for the entire Algorithm. To keep the approximation ratio small it is reasonable to choose as small as possible conflict loops in the first phase.

Using the reachability data structure introduced in [8] the time to destroy all conflict loops in a given graph can be bounded by $O(|\mathcal{V}|^3)$ for both algorithms. However, for industrial SAT instances the computation benefits from the fact that on the one hand, at the beginning, small conflict paths are computed which requires clearly less than the worst-case bound. On the other hand, at the end, the number of vertices and edges has substantially decreased (see section 3).

3 Some Experimental Results

In order to get an idea of the sizes of backdoors of real-world SAT instances we computed RHorn backdoors for several instances in [1, 2, 17]. For the computation of Horn and Binary backdoors it turned out that a simple greedy strategy yields the best results for most instances. For Binary backdoors we always choose that variable for the backdoor that reduces the size of the most clauses with more than two literals, terminating as soon as all clauses are binary. The computation of Horn backdoors can be done analogously. A few results are listed in Fig. 1. Especially the last two rows, the results for the two instances *eq.atree.braun** have to be emphasized. Though relatively small, both instances could not be solved by any solver in the sat competition 2007 within the allowed time (10,000 seconds). Our heuristic found a RHorn backdoor with 761 variables for the instance *eq.atree.braun.13** in less than four minutes. Although a solving process cannot examine all 2^{761} Renameable Horn instances, this still reduces the amount of 'relevant' variables by more than 62%.

It is also worth to mention the good results for instances from Car Configuration [17]. E.g., the optimal RHorn backdoors for the two instances *C208_FA** contain 4.51% resp. 7.46% of all variables [9]. For these instances the heuristic found backdoors with 4.73% resp. 8.21% of all variables (lines 2,3).

An alternative approach to compute RHorn backdoors was used in [15] as a pre-processor in a modified zChaff SAT solver. For the most instances that are given in [15] our algorithm could discover slightly smaller RHorn backdoors. However, for a few instances like e.g. *dp10s10** with 8,372 variables our heuristic did considerably better: The local search strategy found a backdoor with 2,635 variables

[15], whereas the described heuristic discovered a backdoor with 1,543 variables. The reason for this might be that unlike the local search approach, a computation based on the dependency graph is mainly independent of the number of renamings that have to be made to make the remaining instance $F - \mathcal{B}$ Horn.

Instance	# Vars	# Cls	Binary	Horn	RHorn
C169.FW	1402	1982	56	59	2
C208_FA_SZ_120	1608	5278	161	168	76
C208_FA_UT_3254	1876	7334	419	434	154
apex7_gr_rcs_w5.shuffled.cnf	1500	11695	900	832	635
dp10s10.shuffled.cnf	7759	23004	2005	3256	1543
vda_gr_rcs_w9.shuffled.cnf	6498	130997	5054	4695	4262
cnf-r4-b4-k1	9528	59248	8363	4569	576
comb3	4774	16331	1641	2095	1119
dp06u05	2055	6053	560	889	457
ezfact256_1	49153	324873	24092	32936	35998
f2clk_30	20458	59559	7109	7813	3338
par32-4	3176	10313	463	1658	1290
cnf-r4-b1-k1.1-03-416	2424	14812	2113	1141	174
f2clk_40-03-424	27568	80439	9597	10562	4514
eq.atree.braun.12.unsat	1694	5726	686	1003	647
eq.atree.braun.13.unsat	2010	6802	822	1194	761

Fig. 1. Computation of different Backdoors. The three rightmost columns indicate the sizes of the found backdoors with base classes 2-SAT, Horn and RHorn. The smallest backdoor of an instance is highlighted with bold font.

A further interesting aspect when analyzing the computation of RHorn backdoors for industrial SAT instances is the simplification of the dependency graph. In Fig. 2 it can be observed that the simplification of the dependency graphs of instances of the same family behaves similar. For easy instances like those of the family $C220_FV^*$ (left plot) there are several break downs where numbers of vertices can be disregarded and hence deleted according to Corollary 2. On the other hand the computation of backdoors for the very hard real-world instances of the family $eq.atree.braun^*$ (right plot) nearly behaves like the computations for generated instances. Applying Corollary 2 is practically impossible in the first two-thirds of the computation.

4 Conclusions and Further Work

In this paper we have presented two approaches to compute RHorn backdoors for CNF formulas in polynomial time. Both approaches are based on the idea to destroy conflict loops in a RHorn dependency graph. We think that this idea could be used as a preprocessing step for solving small but hard real-world SAT instances in order to drastically reduce the amount of variables to consider for the solving process. For the more general case where a minimal amount of variables has to be deleted in order to make a 2-SAT instance satisfiable, the idea of destroying conflict loops in the implication graph of [3] can be adapted. Furthermore, it is still an open problem if the computation of a minimum RHorn backdoor is fixed parameter tractable. In 2007 the FEEDBACK VERTEX SET problem in directed graphs was proved to be in FPT [5]. It might be possible to adapt their approach for RHorn dependency graphs to remove vertices in order to destroy all conflict loops in the graph.

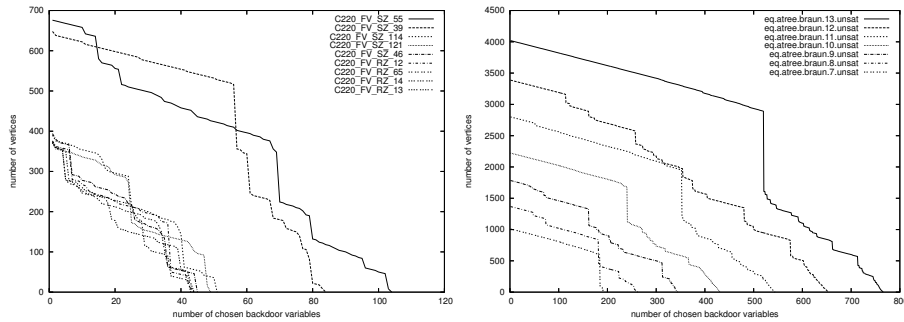


Fig. 2. Simplification of dependency graphs for different families of SAT instances from [2, 17]. The y-axis indicate the number of vertices in the graph and the x-axis indicate the number of variables that are chosen for the present backdoor.

References

1. Dimacs. <ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/>.
2. The international SAT competition. www.satcompetition.org, 2002-2007.
3. B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Proc. Lett.*, 8:121–123, 1979.
4. J. Buresh-Oppenheimer and D. G. Mitchell. Minimum witnesses for unsatisfiable 2CNFs. In *SAT*, 2006.
5. J. Chen, Y. Liu, and S. Lu. Directed feedback vertex set problem is fpt. In *Structure Theory and FPT Algorithmics for Graphs, Digraphs and Hypergraphs*, 2007.
6. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
7. M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.
8. C. Demetrescu and I. Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Inf. Process. Lett.*, 86(3):129–136, 2003.
9. B. Dilkina, C. P. Gomes, and A. Sabharwal. Tradeoffs in the complexity of backdoor detection. In *Principles and Practice of Constraint Programming - CP 2007*, 2007.
10. Y. Interian. Backdoor sets for random 3-sat. In *SAT*, 2003.
11. H. R. Lewis. Renaming a set of clauses as a horn set. *J. ACM*, 25:134–135, 1978.
12. N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to Horn and Binary clauses. In *SAT*, 2004.
13. N. Nishimura, P. Ragde, and S. Szeider. Solving #SAT using vertex covers. *Acta Informatica*, 44(7-8):509–523, 2007.
14. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
15. L. Paris, R. Ostrowski, P. Siegel, and L. Sais. Computing horn strong backdoor sets thanks to local search. In *ICTAI '06*. IEEE Computer Society, 2006.
16. Y. Ruan, H. A. Kautz, and E. Horvitz. The backdoor key: A path to understanding problem hardness. In *AAAI*, pages 124–130, 2004.
17. C. Sinz. SAT benchmarks. www-sr.informatik.uni-tuebingen.de/~sinz/DC, 2003.
18. S. Szeider. Backdoor sets for dll subsolvers. *J. Autom. Reasoning*, 35:73–88, 2005.
19. S. Szeider. Matched formulas and backdoor sets. In *SAT*, 2007.
20. R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *IJCAI*, 2003.