ALIAS: A Modular Tool for Finding Backdoors for SAT

Stepan Kochemazov and Oleg Zaikin

ISDCT SB RAS, Irkutsk, Russia
veinamond@gmail.com, zaikin.icc@gmail.com

Abstract. We present ALIAS, a modular tool aimed at finding backdoors for hard SAT instances. Here by a *backdoor* for a specific SAT solver and SAT formula we mean a set of its variables, all possible instantiations of which lead to construction of a family of subformulas with the total solving time less than that for an original formula. For a particular backdoor, the tool uses the Monte-Carlo algorithm to estimate the runtime of a solver when partitioning an original problem via said backdoor. Thus, the problem of finding a backdoor is viewed as a black-box optimization problem. The tool's modular structure allows to employ state-of-the-art SAT solvers and black-box optimization heuristics. In practice, for a number of hard SAT instances, the tool made it possible to solve them much faster than using state-of-the-art multithreaded SAT-solvers.

1 Introduction

Informally, a *backdoor* is some hidden flaw in a design of a system that allows one to do something within that system that should not be possible otherwise. In application to Constraint Satisfaction Problems the notion of backdoors first arose in [29], where *strong backdoors* were introduced and analyzed. They later were thoroughly studied in the context of SAT for example in [12,17,26]. A *strong backdoor* to some SAT instance (w.r.t. some polynomial algorithm A) in the sense of [29] is such a set of its variables that after any instantiation of the corresponding variables the resulting subproblem is solvable via algorithm A. Thus, a SAT instance which has a strong backdoor possesses a clear complexity structure. It was shown, e.g. in [17], that finding a strong backdoor for a particular SAT instance is a very hard task.

In the present paper, we move away from strong backdoors towards more general notions of backdoors. In particular, hereinafter by a *Nondeterministic Oracle Backdoor Set* (in terms of [24]) or simply a *backdoor* to some SAT instance C (w.r.t. to some SAT solver S) we mean a set of its variables B, such that the runtime of S on C is greater than the total runtime of S on all possible instances formed by instantiating variables from B in C. The intuition here is simple: when we know some backdoor, it makes it possible to solve a considered problem using the available solver faster than it is possible without such knowledge. The process of solving itself is organized in a relatively straightforward divide-and-conquer manner. Unsurprisingly, the good examples that this concept works, come from cryptanalysis applications of SAT: in particular, in [7,10,23,25] it was shown that by choosing a good backdoor it is possible to achieve significant progress in solving problems of cryptanalysis of some symmetric ciphers.

The natural question arises: how to find backdoors to SAT instances? For a given SAT instance C, solver S and backdoor B one can effectively compute the estimation of runtime of S on a family of subproblems produced by assigning values to variables from B in C. It can be done using a Monte-Carlo method [20], thus defining a black-box pseudo-Boolean function. Any black-box pseudo-Boolean optimization method can be used to traverse the search space of possible backdoors to find one with a good estimation. An important disadvantage of this approach is that in order for the search to be more or less efficient, it is necessary to interrupt the calculation of runtime estimation in non-promising points of the search space. For this purpose it is necessary to have some baseline runtime estimation for a considered problem. The latter implies, for example, the knowledge of a Strong Unit Propagation Backdoor Set (SUPBS) [29] for it.

This paper presents modulAr tooL for fInding bAckdoors for Sat (ALIAS) – a convenient customizable scalable tool that can employ arbitrary incremental state-of-the-art SAT solvers and black-box optimization heuristics to search for backdoors for hard SAT instances. The found backdoor is then used to solve the corresponding instance by the same incremental solver. Thereby, ALIAS can be viewed as a tool for constructing backdoor-based divide-and-conquer parallel SAT solvers.

ALIAS is being developed as a replacement of the divide-and-conquer parallel SAT solver PDSAT used extensively to obtain cryptographic results published in [21,23,24]. The main motivation here is that PDSAT is a complex C++ program – it's hard to incorporate new solvers or new black-box optimization heuristics into it. Also, PDSAT was oriented on cryptographic problems, while ALIAS is designed with larger scope in mind. Compared to predecessor, ALIAS is able to find good backdoors much faster with significantly fewer resources. The ALIAS tool and our benchmarks are available at `https://github.com/Nauchnik/alias`.

2 The ALIAS tool

Essentially, in the core of ALIAS lies the ability to compute runtime estimation for an arbitrary backdoor. It is done by the ALIAS_ESTIMATION.PY script using the provided GENIPAINTERVAL application that is implemented based on an IPASIR-compatible solver and an auxiliary SAMPLER application. The purpose of SAMPLER is to organize the subproblems from the random sample into meaningful chunks and to filter out impossible assumptions. The reason for considering only incremental SAT solvers is because typically the subproblems to be solved are different only in values of several backdoor variables. The GENIPAINTERVAL program, given a CNF formula and a set of assumptions processes the latter in incremental way. To build it one needs the IPASIR API [4] and sources of some generic IPASIR-compatible incremental SAT solver. The searching for a good backdoor is currently implemented in the form of a LOCAL_SEARCH module – a program that realizes a relatively simple greedy local search algorithm. However, it can be substituted by other optimization algorithms with relative ease. The backdoor found by the LOCAL_SEARCH module is then used by the solving module to solve the given instance.

Note, that since the amount of computations required to traverse the search space of possible backdoors is quite significant, ALIAS is able to employ as many processor cores as possible. Let us now briefly describe the main modules in more detail.

2.1 Sampler module

Let us give a few comments on the possible structure of a random sample. From the Monte Carlo method point of view, it would be fine if for a backdoor B of size $|B| = k$ we just took N randomly generated vectors from $\{0,1\}^k$ and constructed a random sample by assigning corresponding values to variables from B. This approach was used in [10,23,25]. Thus, strictly speaking, a separate SAMPLER module is not obligatory. However, the described straightforward sampling procedure might not benefit fully from the incremental solving ability of state-of-the-art SAT solvers because the assignments of variables will be too distant from each other (for example Hamming distance-wise). Thus, by default we employ the sampling strategy proposed in [30], where the sample consists of a number of *diapasons*, each containing a continuous set of assumptions. In our experiments, this strategy shows better consistency between estimated runtime and solving time. Note that, compared to [30], the performance of sampling procedure was greatly improved in order to minimize overhead required to compute the runtime estimation for a given backdoor.

The SAMPLER module is implemented on the basis of MINISAT 2.2 [9]. Essentially, it uses the basic scheme of DPLL [8] to traverse a specified diapason of assignments of variables of a given backdoor. The result of this processing is a list of valid assumptions. Here by valid assumption we mean the assumption that requires further decisions in order to decide the corresponding problem. The application loads specification from a .JSON file and outputs the assumptions in a file with DIMACS-like format.

2.2 Runtime estimation and solving modules

The runtime estimation module is implemented as a script for PYTHON 3.6+. It takes as an input the backdoor, the considered CNF formula (.CNF file), the GENIPAINTERVAL binary and various settings that specify how large the random sample should be, what structure should it have, the current best known runtime estimation, etc.

The script implements the basic idea of the Monte-Carlo method: the average solving time of a subproblem produced by instantiating backdoor variables is treated as a random variable. Then, it is possible to estimate its expected value by averaging the solving time over a random sample of such subproblems. The script outputs the runtime estimation in seconds for one processor core (with typical settings).

The solving module is a very close relative of runtime estimation module. It organizes the solving of a specified SAT instance using the provided backdoor and GENIPAINTERVAL by splitting the instance into subproblems, organizing them into sizable chunks and controlling the computations. The result is either the first found satisfying assignment (for satisfiable CNFs) or the *UNSATISFIABLE* verdict. It can also explicitly track the progress .

Generally speaking, the ALIAS_ESTIMATE script implements a blackbox function which computes the runtime estimation expressed as a DOUBLE value. Thus, any tool

that implements pseudo-Boolean optimization algorithm can be used to search for a good backdoor. Below we describe our implementation of such tool, that was used in the experiments.

2.3 Local search module

Note that due to the reasons specified above, on the current stage we can only search for a backdoor which is a subset of some SUPBS. Of course, the whole set of Boolean variables in CNF formula can act as one, however, for smaller SUPBSs the search space is also smaller and thus the search for a good backdoor is more effective.

Assume that a SUPBS for a considered SAT instance contains N variables. Then the search space has 2^N points, each point corresponding to some backdoor. On each backdoor we can launch ALIAS_ESTIMATE.PY as an objective function. For our experiments we implemented LOCAL_SEARCH in the form of a simple optimization algorithm based on the Greedy Best First Search (GBFS) [22]. GBFS starts from a starting point in the role of which we use the whole SUPBS to obtain a baseline runtime estimation (since for SUPBS it can always be computed effectively). Then GBFS checks all points from the neighborhood of the starting point (a set of points at Hamming distance of 1). If it finds a better point, then it starts checking its neighborhood. If all points from a neighborhood are worse than the current best known value, then it means that a local minimum is reached. Since the computation of the objective function in an arbitrary point is quite costly, all passed points are stored in order to not recompute the value of objective function in them.

Compared to implementation in PDSAT [23], the GBFS implementation in ALIAS acquired two major improvements. First, at the beginning of the search the algorithm tries to quickly traverse the search space by removing large amount of random variables (10 in our experiments) from the current record point at each step as long as it leads to updating the record. In our experiments this allowed to quickly move from backdoors with hundreds of variables to ones with dozens. The second improvement is that when a local minimum is reached, the algorithm tries to jump from it by constructing a new starting point by permuting the current record point. The algorithm stops either if the time limit is exceeded, if the limit of jumps is reached or if the current runtime estimation is lower than the (scaled) elapsed time.

On the current stage the ALIAS components are configured in a way to support optimization tools, which were used in Configurable SAT Solver Competition (CSSC) 2013 and 2014 [16], such as ParamILS [15], SMAC [14], and GGA [1]. Similar to our implementation, all these tools make use of the .pcs file that contains Boolean variables corresponding to the starting point (SUPBS).

3 Experimental results

In all experiments described below we employed one node of the HPC-cluster "Academician V.M. Matrosov" [1] (2 $\times$ 18-core Intel Xeon E5-2695 CPUs and 128 Gb of RAM). Each considered solver was launched on 1 node with 36 threads.

[1] Irkutsk Supercomputer Center of SB RAS, http://hpc.icc.ru

We benchmarked ALIAS against the Top 3 solvers from the SAT Competition 2017 Parallel track: SYRUP [3], PLINGELING [5] and PAINLESS-MAPLECOMSPS [11]. All these solvers are portfolio. PAINLESS-MAPLECOMSPS uses MAPLECOMSPS [18] as a core sequential solver.

As IPASIR-based solvers for ALIAS we used the Top 3 solvers from the SAT Competition 2017 Incremental track: ABCDSAT [6], GLUCOSE [2] and RISS [19]. As a result, we constructed 3 versions of ALIAS: ALIAS-ABCDSAT, ALIAS-GLUCOSE and ALIAS-RISS. Each of these programs can be viewed as a parallel SAT solver.

To search for good backdoors we tried to use two programs: SMAC [14] and our implementation of GBFS described in Subsection 2.3. In preliminary experiments GBFS found backdoors with better runtime estimation for all considered instances. Hence, in the main experiments described further GBFS was used. On each instance first GBFS is launched. When it stops due to any stop criterion (see 2.3), the best current backdoor is then used to solve the instance. In practice, the estimation stage took from 8 minutes up to 15 hours on the considered instances.

Two benchmark sets of hard SAT instances were considered. The first set consists of instances, in which a relatively small SUPBS is known. In particular, we considered SAT encodings of cryptanalysis of the alternating step generator (ASG) [13] and two its modifications, MASG and MASG0 [28]. SAT instances for these problems were taken from [30]: 10 for each of ASG-72, ASG-96, MASG-72 and MASG0-72 (40 in total). Naturally, for ASG-72, MASG-72 and MASG0-72 there is a SUPBS of 72 variables and for ASG-96 one of 96 variables (corresponding to secret key). Thus, ALIAS-based solutions were provided with this information. Note, that each instance from this set has exactly one satisfying assignment.

The second benchmark set contains hard small crafted SAT instances. To construct it we first took all crafted instances with less than 500 variables from SAT Competitions 2007, 2009, 2011, 2014, 2016, 2017 and also *challenge-105.cnf* described in [27]. Then we launched SYRUP, PLINGELING and PAINLESS-MAPLECOMSPS on each of them with the time limit of 5000 seconds. It turned out that 33 instances were not solved in time by any solver: 7 from SAT Competition 2007, 10 from SAT Competition 2009, 9 from SAT Competition 2011, 6 from SAT Competition 2014 and also *challenge-105.cnf*. Thus, these 33 instances form the second benchmark set.

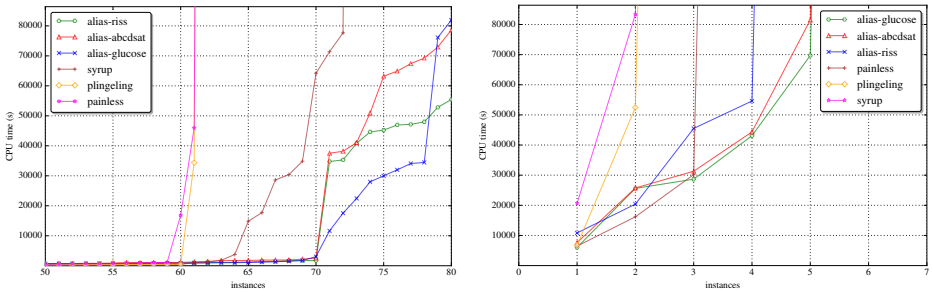
We then launched all 6 considered solvers (multithreaded solvers SYRUP, PLINGELING, PAINLESS-MAPLECOMSPS, and ALIAS-based ALIAS-ABCDSAT, ALIAS-GLUCOSE and ALIAS-RISS) on two described sets (73 instances in total) with the time limit of 1 day.

It should be noted, that for instances from the second benchmark set the ALIAS-based solvers were not given any SUPBS – in this role the set of variables of corresponding formulas was used. Also, for these solvers the time of backdoors searching is included in the total runtime. Hence, on these instances the comparison with solvers from the SAT Competition 2017 was made in equal conditions.

The obtained results are presented in Fig. 1a and Fig. 1b. Table 1 also lists the instances from the second benchmark set, which were solved within the time limit by at least one solver.

Table 1: Results on small hard crafted benchmarks, time in seconds.

instance	source	alias-abcdsat	alias-glucose	alias-riss	painless	syrup	plingeling
mod4block_2vars_ 9gates_u2_autoenc	SAT09	-	-	-	16189	20688	-
sgen1-sat-250-100	SAT09	-	-	-	-	83356	52474
sgen6-1200-5-1	SAT14	7647	6029	10832	6461	-	-
sgen6-1320-5-1	SAT14	31304	28674	54616	-	-	-
sgen6-1440-5-1	SAT14	81527	69715	-	-	-	-
sgen3-n240- s78945233-sat	SAT14	44368	43095	45509	-	-	6728
challenge-105	[27]	25756	25613	20400	30183	-	-
solved in time		5	5	4	3	2	2



(a) ASG-72, ASG-96, MASG-72, MASG0-72 (b) Hard small crafted instances

Fig. 1: Comparison of 3 ALIAS-based solvers with the Top 3 solvers from the SAT Competition 2017 Parallel track

It is clear that on cryptographic tests (from the first benchmark set) ALIAS-based solvers greatly outperform the competitors. We also tested ALIAS-based solvers on these instances without known SUPBS, and as a result bad backdoors were found. Hence, in this case the knowledge of a small SUPBS is a big advantage. However, for many instances arising from practice such small SUPBS can naturally be constructed. ALIAS-based solvers also won on hard small crafted benchmarks, but here the situation is more complex: there are instances which are solved by ALIAS but not by the competitors and vice versa. It turned out, that *sgen3-n240-s78945233-sat* and *sgen1-sat-250-100* are satisfiable, while *challenge-105.cnf*, *mod4block_2vars_9gates_u2_autoenc*, *sgen6-1200-5-1*, *sgen6-1320-5-1* and *sgen6-1440-5-1* are unsatisfiable

4 Conclusions and future work

We believe that the presented ALIAS tool may be useful in the study of hard SAT instances and sometimes shed the light on some aspects of their inner structure undetectable by state-of-the-art SAT solvers.

In the nearest future we plan to extend the functionality of ALIAS by being able to employ arbitrary solver binaries, make the tool compatible with other available black-box optimization tools and reuse the information obtained during the search for a backdoor when solving the problem.

Acknowledgements. The research was funded by Russian Science Foundation (project No. 16-11-10046) and by Council for Grants of the President of the Russian Federation (stipend no. SP-1829.2016.5).

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: CP 2009. LNCS, vol. 5732, pp. 142–157 (2009)
2. Audemard, G., Lagniez, J., Simon, L.: Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In: SAT 2013. LNCS, vol. 7962, pp. 309–317 (2013)
3. Audemard, G., Simon, L.: Lazy clause exchange policy for parallel SAT solvers. In: SAT 2014. LNCS, vol. 8561, pp. 197–205 (2014)
4. Balyo, T., Biere, A., Iser, M., Sinz, C.: SAT race 2015. Artif. Intell. 241, 45–65 (2016)
5. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT entering the SAT competition 2017. In: Proc. of SAT Competition 2017. vol. B-2017-1, pp. 14–15 (2017)
6. Chen, J.: Improving abcdSAT by At-Least-One recently used clause management strategy. CoRR abs/1605.01622 (2016), http://arxiv.org/abs/1605.01622
7. Courtois, N.: Low-complexity key recovery attacks on GOST block cipher. Cryptologia 37(1), 1–10 (Jan 2013)
8. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
9. Eén, N., Sörensson, N.: An extensible SAT-solver. In: SAT 2003. Selected Revised Papers. LNCS, vol. 2919, pp. 502–518 (2003)
10. Eibach, T., Pilz, E., Völkel, G.: Attacking Bivium using SAT solvers. In: SAT 2008. LNCS, vol. 4996, pp. 63–76 (2008)
11. Frioux, L.L., Baarir, S., Sopena, J., Kordon, F.: PaInleSS: A framework for parallel SAT solving. In: SAT 2017. LNCS, vol. 10491, pp. 233–250 (2017)
12. Ganian, R., Ramanujan, M.S., Szeider, S.: Backdoor treewidth for SAT. In: SAT 2017. pp. 20–37 (2017)
13. Günther, C.G.: Alternating Step Generators Controlled by De Bruijn Sequences, LNCS, vol. 304, pp. 5–14 (1988)
14. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. of LION-5. p. 507523 (2011)
15. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR 36, 267–306 (2009)
16. Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H., Leyton-Brown, K.: The configurable SAT solver challenge (CSSC). Artificial Intelligence 243, 1 – 25 (2017)
17. Kilby, P., Slaney, J.K., Thiébaux, S., Walsh, T.: Backbones and backdoors in satisfiability. In: AAAI 2005. pp. 1368–1373 (2005)
18. Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 123–140. Springer (2016)

19. Manthey, N.: Towards next generation sequential and parallel SAT solvers. Constraints 20(4), 504–505 (2015)
20. Metropolis, N., Ulam, S.: The Monte Carlo Method. J. Amer. statistical assoc. 44(247), 335–341 (1949)
21. Otpuschennikov, I., Semenov, A., Gribanova, I., Zaikin, O., Kochemazov, S.: Encoding cryptographic functions to SAT using TRANSALG system. In: ECAI 2016. FAIA, vol. 285, pp. 1594–1595 (2016)
22. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 3rd edn. (2009)
23. Semenov, A., Zaikin, O.: Algorithm for finding partitionings of hard variants of Boolean satisfiability problem with application to inversion of some cryptographic functions. SpringerPlus 5(1), 1–16 (2016)
24. Semenov, A., Zaikin, O., Otpuschennikov, I., Kochemazov, S., Ignatiev, A.: On cryptographic attacks using backdoors for SAT (in print). In: AAAI 2018 (2018)
25. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: SAT 2009. LNCS, vol. 5584, pp. 244–257 (2009)
26. Szeider, S.: Backdoor sets for DLL subsolvers. Journal of Automated Reasoning 35(1), 73–88 (2005)
27. Van Gelder, A., Spence, I.: Zero-one designs produce small hard SAT instances. In: SAT 2010. LNCS, vol. 6175, pp. 388–397 (2010)
28. Wicik, R., Rachwalik, T.: Modified alternating step generators. IACR Cryptology ePrint Archive 2013, 728 (2013)
29. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. In: IJCAI'03. pp. 1173–1178 (2003)
30. Zaikin, O., Kochemazov, S.: An improved SAT-based guess-and-determine attack on the alternating step generator. In: ISC 2017. LNCS, vol. 10599, pp. 21–38 (2017)