Using Monte Carlo Method for Searching Partitionings of Hard Variants of Boolean Satisfiability Problem

Alexander Semenov and Oleg Zaikin

Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia
biclop.rambler@yandex.ru, zaikin.icc@gmail.com

Abstract. In this paper we propose the approach for constructing partitionings of hard variants of the Boolean satisfiability problem (SAT). Such partitionings can be used for solving corresponding SAT instances in parallel. We suggest the approach based on the Monte Carlo method for estimating time of processing of an arbitrary partitioning. We solve the problem of search for a partitioning with good effectiveness via the optimization of the special predictive function over the finite search space. For this purpose we use the tabu search strategy. In our computational experiments we found partitionings for SAT instances encoding problems of inversion of some cryptographic functions. Several of these SAT instances with realistic predicted solving time were successfully solved on a computing cluster and in the volunteer computing project SAT@home. The solving time agrees well with estimations obtained by the proposed method.

Keywords: Monte Carlo method, SAT, partitioning, tabu search, cryptanalysis

1 Introduction

The Boolean satisfiability problem (SAT) consists in the following: for an arbitrary Boolean formula (formula of the Propositional Calculus) to decide if it is satisfiable, i.e. if there exists such an assignment of Boolean variables from the formula that makes this formula true. The satisfiability problem for a Boolean formula can be effectively (in polynomial time) reduced to the satisfiability problem for the formula in the conjunctive normal form (CNF). Hereinafter by SAT instance we mean the satisfiability problem for some CNF.

Despite the fact that SAT is NP-complete (NP-hard as a search problem) it is very important because of the wide specter of practical applications. A lot of combinatorial problems from different areas can be effectively reduced to SAT [**?**]. In the last 10 years there was achieved an impressive progress in the effectiveness of SAT solving algorithms. While these algorithms are exponential in the worst case scenario, they display high effectiveness on various classes of industrial problems.

Because of the high computational complexity of SAT, the development of methods for solving hard SAT instances in parallel is considered to be relevant. Nowadays the most popular approaches to parallel SAT solving are *portfolio* approach and *partitioning* approach [**?**]. In the portfolio approach several copies of the SAT solver process the same search space in different directions. The partitioning approach implies that

the original SAT instance is decomposed into a family of subproblems and this family is then processed in a parallel or in a distributed computing environment. This family is in fact a partitioning of the original SAT instance. The ability to independently process different subproblems makes it possible to employ the systems with thousands of computing nodes for solving the original problem. Such approach allows to solve even some cryptanalysis problems in the SAT form. However, for the same SAT instance one can construct different partitionings. In this context the question arises: if we have two partitionings, how can we know if one is better than the other? Or, if we look at this from the practical point of view, how to find if not best partitioning, then at least the one with more or less realistic time required to process all the subproblems in it? In the present paper we study these two problems.

2 Monte Carlo Approach to Statistical Estimation of Effectiveness of SAT Partitioning

Let us consider the SAT for an arbitrary CNF C. The partitioning of C is a set of formulas

$$C \wedge G_j, j \in \{1, \ldots, s\} \tag{1}$$

such that for any $i, j : i \neq j$ formula $C \wedge G_i \wedge G_j$ is unsatisfiable and

$$C \equiv C \wedge G_1 \vee \ldots \vee C \wedge G_s.$$

(where "$\equiv$" stands for logical equivalence). It is obvious that when one has a partitioning of the original SAT instance, the satisfiability problems for CNFs (**??**) can be solved independently in parallel.

There exist various partitioning techniques [**?**]. The results of the research on estimating the time required to process SAT partitionings can be found in a number of papers on logical cryptanalysis [**?**], [**?**], [**?**]. In the present paper we propose to construct time estimations for the processing of SAT partitionings using the Monte Carlo method in its classical form [**?**].

Consider the satisfiability problem for an arbitrary CNF C over a set of Boolean variables $X = \{x_1, \ldots, x_n\}$. We call an arbitrary set $\tilde{X} = \{x_{i_1}, \ldots, x_{i_d}\}$, $\tilde{X} \subseteq X$ a decomposition set. Consider a partitioning of C that consists of a set of $s = 2^d$ formulas of the kind (**??**), where G_j, $j \in \{1, \ldots, 2^d\}$ are all possible minterms over $\tilde{X}$. Note that an arbitrary formula G_j takes a value of true on a single truth assignment $\left(\alpha_1^j, \ldots, \alpha_d^j\right) \in \{0,1\}^d$. Therefore, an arbitrary formula $C \wedge G_j$ is satisfiable if and only if $C\left[\tilde{X}/\left(\alpha_1^j, \ldots, \alpha_d^j\right)\right]$ is satisfiable. Here $C\left[\tilde{X}/\left(\alpha_1^j, \ldots, \alpha_d^j\right)\right]$ is produced by setting values of variables x_{i_k} to corresponding α_k^j, $k \in \{1, \ldots, d\} : x_{i_1} = \alpha_1^j, \ldots, x_{i_d} = \alpha_d^j$. A set of CNFs

$$\Delta_C(\tilde{X}) = \left\{C\left[\tilde{X}/\left(\alpha_1^j, \ldots, \alpha_d^j\right)\right]\right\}_{\left(\alpha_1^j, \ldots, \alpha_d^j\right) \in \{0,1\}^d}$$

is called a decomposition family produced by $\tilde{X}$. It is clear that the decomposition family is the partitioning of the SAT instance C.

Consider some algorithm A solving SAT. In the remainder of the paper we presume that A is complete, i.e. its runtime is finite for an arbitrary input. We also presume that A is a non-randomized deterministic algorithm. We denote the amount of time required for A to solve all the SAT instances from $\Delta_C\left(\tilde{X}\right)$ as $t_{C,A}\left(\tilde{X}\right)$. Below we concentrate mainly on the problem of estimating $t_{C,A}\left(\tilde{X}\right)$.

Define the uniform distribution on the set $\{0,1\}^d$. With each randomly chosen truth assignment $(\alpha_1,\ldots,\alpha_d)$ from $\{0,1\}^d$ we associate a value $\xi_{C,A}(\alpha_1,\ldots,\alpha_d)$ that is equal to the time required for the algorithm A to solve SAT for $C\left[\tilde{X}/(\alpha_1,\ldots,\alpha_d)\right]$. Let $\xi^1,\ldots,\xi^Q$ be all the different values that $\xi_{C,A}(\alpha_1,\ldots,\alpha_d)$ takes on all the possible $(\alpha_1,\ldots,\alpha_d)\in\{0,1\}^d$. Let us denote $\xi_{C,A}\left(\tilde{X}\right)=\left\{\xi^1,\ldots,\xi^Q\right\}$, and let $\sharp\xi^j$ be the number of $(\alpha_1,\ldots,\alpha_d)$, such that $\xi_{C,A}(\alpha_1,\ldots,\alpha_d)=\xi^j$. Then $\xi_{C,A}\left(\tilde{X}\right)$ is a random variable with distribution $P\left(\xi_{C,A}\left(\tilde{X}\right)\right)=\{p_1,\ldots,p_Q\}$, where $p_k=\frac{\sharp\xi^k}{2^d}$, $k\in\{1,\ldots,Q\}$. Thus, it is easy to see that

$$t_{C,A}\left(\tilde{X}\right)=\sum_{k=1}^{Q}\left(\xi^k\cdot\sharp\xi^k\right)=2^d\cdot\mathrm{E}\left[\xi_{C,A}\left(\tilde{X}\right)\right]. \tag{2}$$

To estimate the expected value $\mathrm{E}\left[\xi_{C,A}\left(\tilde{X}\right)\right]$ we will use the Monte Carlo method [**?**], according to which, a probabilistic experiment, that consists of N independent observations of values of an arbitrary random variable ξ, is used to approximately calculate $\mathrm{E}[\xi]$. Let $\zeta^1,\ldots,\zeta^N$ be the results of the corresponding observations. From the theoretical basis of the Monte Carlo method it follows that if ξ has finite expected value and finite variance, then the value $\frac{1}{N}\cdot\sum\limits_{j=1}^{N}\zeta^j$ is a good approximation of $\mathrm{E}[\xi]$ when the number of observations is large enough. In our case from the assumption regarding the completeness of the algorithm A it follows that random variable $\xi_{C,A}(\tilde{X})$ has finite expected value and finite variance. We would like to mention that an algorithm A should not use randomization, since if it does then the observed values in the general case will not have the same distribution. The fact that N can be significantly less than 2^d makes it possible to use the preprocessing stage to estimate the effectiveness of the considered partitioning.

So the process of estimating the value (**??**) for a given $\tilde{X}$ is as follows. We construct a random sample $\alpha^1,\ldots,\alpha^N$, where $\alpha^j=\left(\alpha_1^j,\ldots,\alpha_d^j\right)$, $j\in\{1,\ldots,N\}$ is a truth assignment of variables from $\tilde{X}$. Then consider values $\zeta^j=\xi_{C,A}\left(\alpha^j\right),j=1,\ldots,N$ and calculate the value

$$F_{C,A}\left(\tilde{X}\right)=2^d\cdot\left(\frac{1}{N}\cdot\sum_{j=1}^{N}\zeta^j\right). \tag{3}$$

By the above, if N is large enough then the value of $F_{C,A}\left(\tilde{X}\right)$ can be considered as a good approximation of (**??**). Therefore, instead of searching for a decomposition

set with minimal value (**??**) one can search for a decomposition set with minimal value of $F_{C,A}(\cdot)$. Below we refer to function $F_{C,A}(\cdot)$ as *predictive function*.

3 Algorithm for Minimization of Predictive Function

As we already noted above, different partitionings of the same SAT instance can have different values of $t_{C,A}\left(\tilde{X}\right)$. In practice it is important to be able to find partitionings that can be processed in realistic time. Below we will describe the scheme of automatic search for good partitionings that is based on the procedure minimizing the predictive function value in the special search space.

So we consider the satisfiability problem for some CNF C. Let $X = \{x_1, \ldots, x_n\}$ be the set of all Boolean variables in this CNF and $\tilde{X} \subseteq X$ be an arbitrary decomposition set. The set $\tilde{X}$ can be represented by the binary vector $\chi = (\chi_1, \ldots, \chi_n)$. Here

$$\chi_i = \begin{cases} 1, if\ x_i \in \tilde{X} \\ 0, if\ x_i \notin \tilde{X} \end{cases}, i \in \{1, \ldots, n\}$$

With an arbitrary vector $\chi \in \{0,1\}^n$ we associate the value of function $F(\chi)$ computed in the following manner. For vector χ we construct the corresponding set $\tilde{X}$ (it is formed by variables from X that correspond to 1 positions in χ). Then we generate a random sample $\alpha^1, \ldots, \alpha^N$, $\alpha^j \in \{0,1\}^{|\tilde{X}|}$ and solve SAT for CNFs $C\left[\tilde{X}/\alpha^j\right]$. For each of these SAT instances we measure ζ^j — the runtime of algorithm A on the input $C\left[\tilde{X}/\alpha^j\right]$. After this we calculate the value of $F_{C,A}\left(\tilde{X}\right)$ according to (**??**). As a result we have the value of $F(\chi)$ in the considered point of the search space. Then we solve the problem $F(\chi) \to min$ over the set $\{0,1\}^n$.

The minimization of function $F(\cdot)$ over $\{0,1\}^n$ is considered as an iterative process of transitioning between the points of the search space. By $N_\rho(\chi)$ we denote the neighborhood of point χ of radius ρ in the search space $\{0,1\}^n$. The point from which the search starts we denote as χ_{start}. We will refer to the decomposition set specified by this point as $\tilde{X}_{start}$. The current Best Known Value of $F(\cdot)$ is denoted by F_{best}. The point in which the F_{best} was achieved we denote as χ_{best}. By χ_{center} we denote the point the neighborhood of which is processed at the current moment. We call the point, in which we computed the value $F(\cdot)$, a *checked point*. The neighborhood $N_\rho(\chi)$ in which all the points are checked is called *checked neighborhood*. Otherwise the neighborhood is called *unchecked*.

For the minimization of $F(\cdot)$ we employed the tabu search strategy [**?**]. According to this approach the points from the search space, in which we already calculated the values of function $F(\cdot)$ are stored in special tabu lists, to which we refer below as to L_1 and L_2. The L_1 list contains only points with checked neighborhoods. The L_2 list contains checked points with unchecked neighborhoods. Below we present the pseudocode of the tabu search algorithm for $F(\cdot)$ minimization.

In this algorithm the function `markPointInTabuLists`(χ, L_1, L_2) adds the point χ to L_2 and then marks χ as checked in all neighborhoods of points from L_2 that contain χ. If as a result the neighborhood of some point χ' becomes checked, the point

Algorithm 1: Tabu search altorithm for minimization of the predictive function

Input: CNF C, initial point χ_{start}
Output: Pair $\langle \chi_{best}, F_{best} \rangle$, where F_{best} is a prediction for C, χ_{best} is a corresponding decomposition set

$\langle \chi_{center}, F_{best} \rangle \leftarrow \langle \chi_{start}, F(\chi_{start}) \rangle$
$\langle L_1, L_2 \rangle \leftarrow \langle \emptyset, \chi_{start} \rangle$ `// initialize tabu lists`
repeat
  bestValueUpdated $\leftarrow$ false
  repeat `// check neighborhood`
    $\chi \leftarrow$ any unchecked point from $N_\rho(\chi_{center})$
    compute $F(\chi)$
    `markPointInTabuLists(`χ, L_1, L_2`)` `// update tabu lists`
    if $F(\chi) < F_{best}$ **then**
      $\langle \chi_{best}, F_{best} \rangle \leftarrow \langle \chi, F(\chi) \rangle$
      bestValueUpdated $\leftarrow$ true
  until $N_\rho(\chi_{center})$ *is checked*
  if bestValueUpdated **then** $\chi_{center} \leftarrow \chi_{best}$
  else $\chi_{center} \leftarrow$ `getNewCenter(`L_2`)`
until `timeExceeded()` *or* $L_2 = \emptyset$
return $\langle \chi_{best}, F_{best} \rangle$

χ' is removed from L_2 and is added to L_1. If we have processed all the points in the neighborhood of χ_{center} but could not improve the F_{best} then as the new point χ_{center} we choose some point from L_2. It is done via the function `getNewCenter(`L_2`)`. To choose the new point in this case one can use various heuristics. At the moment the tabu search algorithm chooses the point for which the total conflict activity [**?**] of Boolean variables, contained in the corresponding decomposition set, is the largest.

4 Computational Experiments

The algorithms presented in the previous section were implemented as the MPI-program PDSAT.[1] In PDSAT there is one leader process, all the other are computing processes (each process corresponds to 1 CPU core). For every new point $\chi = \chi\left(\tilde{X}\right)$ from the search space the leader process creates a random sample of size N (we use neighborhoods of radius $\rho = 1$). Each assignment from this sample in combination with the original CNF C define the SAT instance from the decomposition family $\Delta_C\left(\tilde{X}\right)$. These SAT instances are solved by computing processes. The value of the predictive function is always computed assuming that the decomposition family will be processed by 1 CPU core. The fact that the processing of $\Delta_C\left(\tilde{X}\right)$ consists in solving independent

[1] https://github.com/Nauchnik/pdsat

subproblems makes it possible to extrapolate the estimation obtained to an arbitrary parallel (or distributed) computing system. The computing processes use slightly modified MINISAT solver[2] for solving SAT instances.

Below we present the results of computational experiments in which PDSAT was used on the computing cluster "Academician V.M. Matrosov" to estimate the time required to solve problems of logical cryptanalysis of the A5/1 [**?**] and Bivium [**?**] keystream generators. The SAT instances that encode these problems were produced using the TRANSALG system [**?**]. All the estimations presented below are in seconds.

4.1 Time Estimations for Logical Cryptanalysis of A5/1

For the first time we considered the logical cryptanalysis of the A5/1 keystream generator in [**?**]. In that paper we described the corresponding algorithm in detail, therefore we will not do it in the present paper. We considered the cryptanalysis problem for the A5/1 keystream generator in the following form: given the 114 bits of keystream we needed to find the secret key of length 64 bits, which produces this keystream (in accordance with the A5/1 algorithm). During predictive function minimization PDSAT used random samples of size $N = 10^4$ SAT instances and worked for 1 day using 5 computing nodes (160 CPU cores in total) within the computing cluster. Using the tabu search algorithm we found the set $S_2 = \{x_2, ..., x_{10}, x_{20}, ..., x_{30}, x_{39}, x_{40}, x_{42}, ..., x_{52}\}$. We compared the time estimations for this set with that of the decomposition set S_1, the structure of which was described in [**?**]. The S_1 set was constructed manually based on the analysis of the algorithmic features of the A5/1 keystream generator. The value of predictive function for S_1 is equal to 4.45140e+08, and for S_2 is equal to 4.64428e+08.

Since the obtained estimations turned out to be realistic, we decided to solve non-weakened cryptanalysis instances for A5/1. For this purpose we used the BOINC-based volunteer computing project SAT@home.[3] In total we performed two computational experiments on solving cryptanalysis of A5/1 in SAT@home. In the first experiment we solved 10 cryptanalysis instances using the S_1 set and in the second we solved same 10 instances using the S_2 set. To construct the corresponding tests we used the known rainbow-tables for the A5/1 algorithm. These tables provide about 88% probability of success when analyzing 8 bursts of keystream (i.e. 914 bits). We randomly generated 1000 instances and applied the rainbow-tables technique to analyze 8 bursts of keystream, generated by A5/1. Among these 1000 instances the rainbow-tables could not find the secret key for 125 problems. From these 125 instances we randomly chose 10 and in the computational experiments applied the SAT approach to the analysis of first bursts of the corresponding keystream fragments (114 bits). In all cases we successfully found the secret keys.

4.2 Time Estimations for Logical Cryptanalysis of Bivium

The Bivium keystream generator [**?**] uses two shift registers. The first register contains 93 cells and the second contains 84 cells. To initialize the cipher, a secret key of length

[2] http://minisat.se

[3] http://sat.isa.ru/pdsat/

80 bit is put to the first register, and a fixed (known) initialization vector of length 80 bit is put to the second register. All remaining cells are filled with zeros. An initialization phase consists of 708 rounds during which keystream output is not released.

In accordance with [**?**] we considered cryptanalysis problem for Bivium in the following formulation. Based on the known fragment of keystream we search for the values of all registers cells at the end of the initialization phase. Therefore, in our experiments we used the CNF encoding where the initialization phase was omitted. Usually it is believed that to uniquely identify the secret key it is sufficient to consider keystream fragment of length comparable to the total length of shift registers. Here we followed [**?**], [**?**] and set the keystream fragment length for Bivium cryptanalysis to 200 bits. In the role of $\tilde{X}_{start}$ for the cryptanalysis of Bivium we chose the set formed by the variables encoding the cells of registers of the generator considered at the end of the initialization phase. Further we refer to these variables as *starting* variables. Therefore $\left|\tilde{X}_{start}\right| = 177$. During predictive function minimization PDSAT used random samples of size $N = 10^5$ SAT instances and worked for 1 day using 5 computing nodes (160 CPU cores in total) within the computing cluster. Time estimations obtained for the Bivium cryptanalysis is $F_{best} = 3.769 \times 10^{10}$.

In [**?**], [**?**] a number of time estimations for logical cryptanalysis of Bivium were proposed. In particular, in [**?**] several fixed types of decomposition sets were analyzed. Time estimation for the best decomposition set from [**?**] is equal to 1.637×10^{13}, it was calculated using random samples of size 10^2. Authors of [**?**] constructed estimations for the sets of variables chosen during the solving process and extrapolated the estimations obtained to time points of the solving process that lay in the distant future. Apparently, as it is described in [**?**], the random samples of size 10^2 and 10^3 were used. In the Table **??** all three estimations mentioned above are demonstrated. The performance of one core of the processor we used in our experiments is comparable with that of one core of the processor used in [**?**].

Table 1: Time estimations for the Bivium cryptanalysis problem

Source	Sample size	Time estimation
From [**?**]	10^2	1.637×10^{13}
From [**?**]	10^3	9.718×10^{10}
Found by PDSAT	10^5	3.769×10^{10}

To compare obtained time estimations with real solving time we solved several weakened logical cryptanalysis problems for Bivium. Below we use the notation *BiviumK* to denote a weakened problem for Bivium with known values of *K* starting variables. We used the volunteer computing project SAT@home to solve 5 instances of *Bivium9*. For all considered instances the time required to solve the corresponding instances agrees well with our estimations. An extended version of this paper can be found online.[4]

[4] https://github.com/Nauchnik/Monte-Carlo-SAT/blob/master/Semenov-Monte-Carlo-SAT.pdf

Acknowledgements The authors wish to thank Stepan Kochemazov for numerous valuable comments. This work was partly supported by Russian Foundation for Basic Research (grants 14-07-00403-a and 15-07-07891-a) and by the President of Russian Federation grant for young scientists SP-1184.2015.5.