

# **Using Backdoors to estimate the hardness of Boolean formulas w.r.t. SAT solving algorithms**

Semenov A. (ITMO, ISDCT SB RAS)

Kochemazov S. (ISDCT SB RAS, ITMO)

# Boolean Satisfiability problem (SAT)

Boolean Satisfiability Problem (SAT) is a classical NP-complete/NP-hard combinatorial problem with a huge spectrum of practical applications



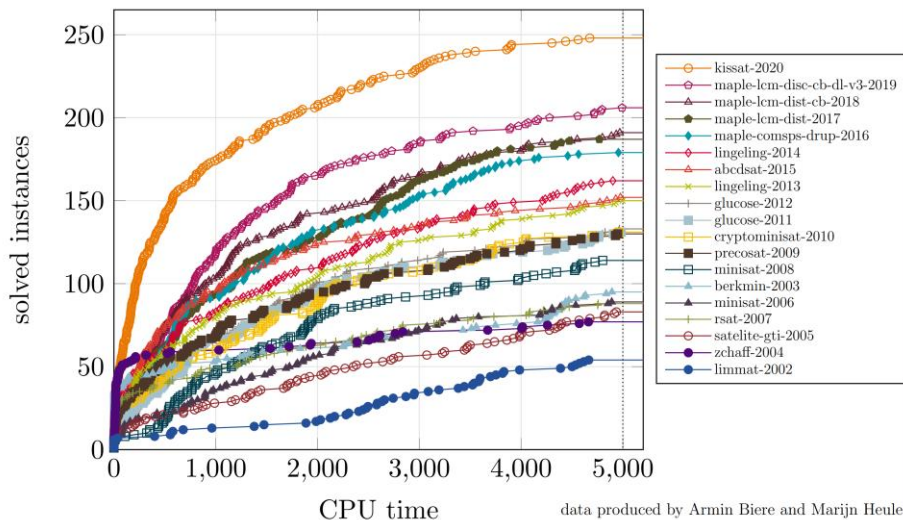
Surprising fact: state-of-the-art SAT solvers can successfully tackle Boolean formulas with millions variables and clauses.

# Boolean Satisfiability problem (SAT)

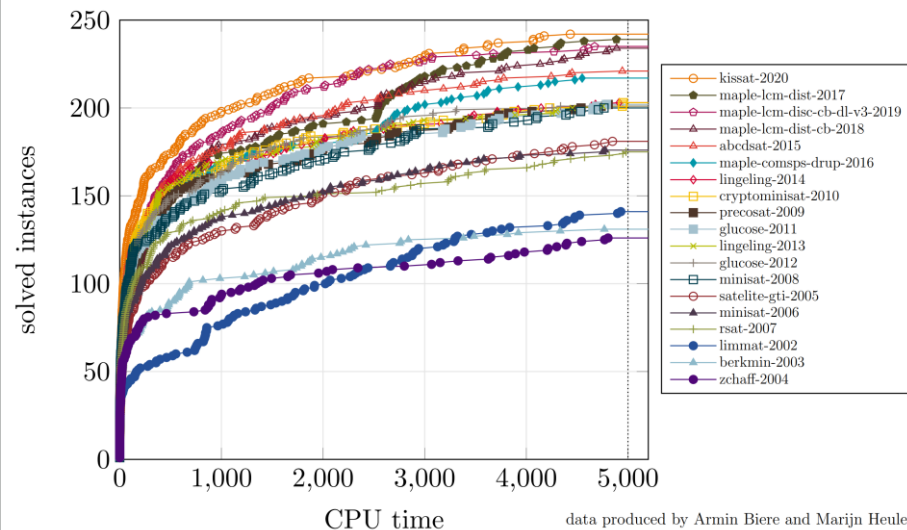
*“... However, we hear about “SAT-solvers” and other algorithms for NP-hard problems all the time. Sometimes people are just mistaken (e.g. they don’t realize their graphs have some hidden constraint), sometimes they are just using an approximation algorithm, but sometimes it legitimately seems like the empirical, real-life problems are simply not the worst-case ones.”*

**Madhu Sudan**

SAT Competition Winners on the SC2020 Benchmark Suite



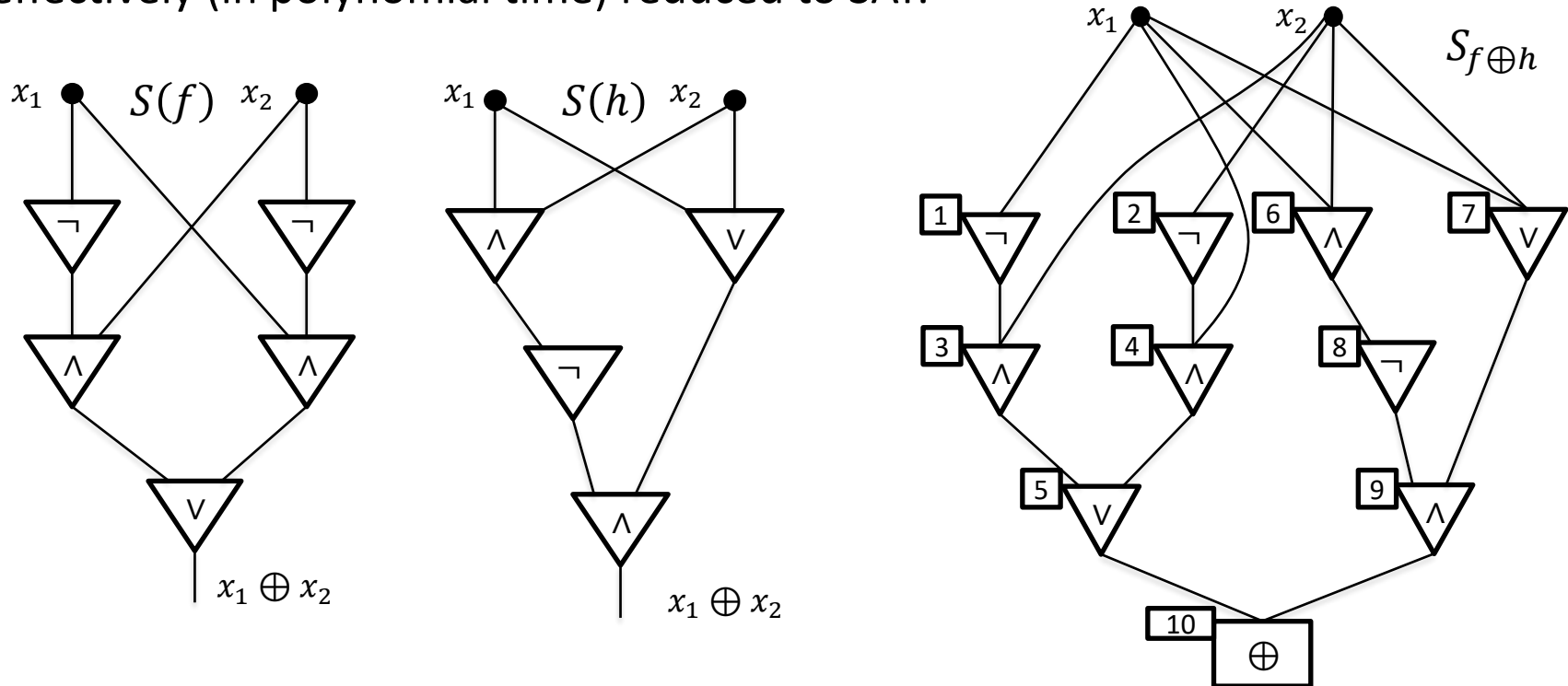
SAT Competition Winners on the SC2011 Benchmark Suite



The plots are from <http://fmv.jku.at/kissat/>

# Example 1. Hardware verification

Consider two Boolean circuits  $S_1$  and  $S_2$ . We need to check if they are equivalent, i.e. specify the same Boolean function. The problem can be effectively (in polynomial time) reduced to SAT:



First, we construct circuit  $S_{f \oplus h}$ , then apply the “Tseitin transformations”\* procedure to  $S_{f \oplus h}$  in order to construct a CNF. The resulting CNF is unsatisfiable if and only if the original circuits are equivalent.

\*Tseitin G. On the complexity of derivation in propositional calculus. 1970

(Цейтин Г.С. О сложности вывода в исчислении высказываний. 1968. Записки научных семинаров ЛОМИ. Т. 8. С. 234-259)

## Example 2. Combinatorial designs

Latin square (LS) is an array of size  $n \times n$  filled with different  $n$  symbols in such a way that each symbols occurs exactly once in each row and exactly once in each column.

It is a well-known mathematical object which appears in many problems in algebra and combinatorics (e.g. LS is a Cayley table of finite group).

$A$	$B$	$C$
$B$	$C$	$A$
$C$	$A$	$B$

$\alpha$	$\gamma$	$\beta$
$\beta$	$\alpha$	$\gamma$
$\gamma$	$\beta$	$\alpha$

Latin squares of order 3

$A\alpha$	$B\gamma$	$C\beta$
$B\beta$	$C\alpha$	$A\gamma$
$C\gamma$	$A\beta$	$B\alpha$

Greco-Latin square  
of order 3

All pairs of symbols are  
different (at least in one  
coordinate)

Two Latin Squares of the same order are called orthogonal if they form a Greco-Latin square (GLS) (or Euler's Square).

# Greco-Latin squares

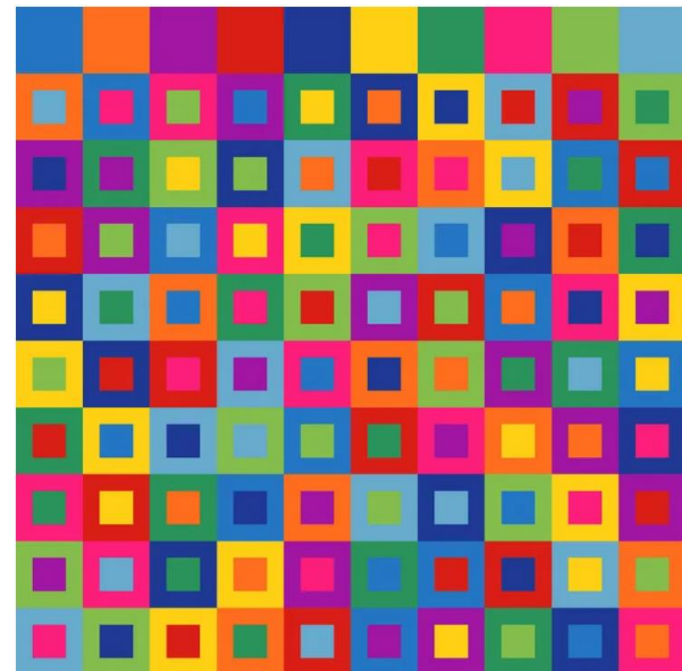
Latin squares exist for any order  $\geq 2$  (it follows from the properties of  $Z_n$  group). The question is, for what orders do Graeco-Latin squares exist?

Leonard Euler (appr. in 1780) proposed a conjecture that GLSs do not exist for all orders of the kind  $n = 4k + 2, k \in \mathbb{N}$ .

This conjecture was refuted by R. Bose and S. Shrikhande in 1959 and in the end of the same year R. Bose, S. Shrikhande and E. Parker gave the first example of GLS of order 10. Thus, Euler conjecture stayed relevant for more than 150 years!

Modern SAT solvers generate GLS of order 10 in several minutes! (each such a GLS refutes Euler conjecture).

However, SAT solvers are still not able to answer the question whether a triple of mutually pair-wise orthogonal Latin squares of order 10 exists? (this is one of the most well-known open problems of finite combinatorics.)



A Greco-Latin square of order 10 using colors. The inner and outer squares are the two sets. Each color appears once as an inner square and once as an outer square in each row and column, and each pair of colors appears exactly once. Credit: Rob Bulmahn Flickr (CC BY 2.0)

# Estimating the hardness of hard SAT instances

Modern SAT solvers can be used to solve many hard combinatorial problems.

However, there is no way to know in advance, whether the solver will be able to solve a SAT instance under specific time limit or not, because SAT solving algorithms show drastically different behavior on different input words.

Thus, the following problem arises:

We have some good (in practice) SAT solver  $A$  and some Boolean formula  $C$ , the satisfiability of which we want to determine. The goal is to estimate how hard is formula  $C$  for algorithm  $A$ ?

Note, that in this formulation we work with ***a single specific formula***, unlike the situation common for complexity, e.g. when we deal with some lower or upper bounds for families of formulas of increasing size.

In this context, it will be more convenient to use the notion of “hardness of  $C$  w.r.t. algorithm  $A$ ” (instead of “complexity”).

# Backdoors to SAT

How to estimate the hardness of  $C$  w.r.t.  $A$  effectively? (without waiting for  $A$  to terminate).

Some of the results discussed further were inspired by the notion of *Backdoor sets* or simply *Backdoors*.

Originally, Backdoors were introduced in the well-known paper\* and the corresponding definition implies that the algorithm  $A$  (in the above context) is polynomial.

Assume that  $\{0,1\}^{|B|}$  denotes the set of all possible assignments of variables from  $B$ , and  $C[\beta/B]$  refers to a formula obtained from  $C$  by substituting an assignment  $\beta$  of variables from  $B$  to it.

## Definition 1.

For Boolean formula  $C$  over set of variables  $X$  the set  $B \subseteq X$  is Strong Backdoor Set (SBS) w.r.t. polynomial algorithm  $A$  if for any  $\beta \in \{0,1\}^{|B|}$  SAT for formula\*\*  $C[\beta/B]$  is correctly solved by  $A$ .

\*R. Williams, C. Gomes, B. Selman Backdoors to typical case complexity. IJCAI 2003.



# Backdoor hardness

Keeping in mind the SBS definition, we have the following observation\*:

Knowing some strong backdoor set  $B$  w.r.t. some polynomial algorithm  $A$  allows us to construct an upper bound on hardness of  $C$  of the kind

$$Poly(|C|) \cdot 2^{|B|}.$$

It is clear that a trivial SBS always exists (e.g.  $B = X$ ), and we can define “backdoor hardness” of  $C$  w.r.t.  $A$  as follows:

Definition 2 (directly follows from\*).

The backdoor hardness (***b-hardness***) of  $C$  is the following value:

$$\min_{\substack{B \subseteq X: \\ B \text{ is SBS}}} \mu_{C,A}(B)$$

where  $\mu_{A,C}(B)$  is a total runtime of  $A$  on all formulas  $C[\beta/B]$  ( $\beta \in \{0,1\}^{|B|}$ ).

In this context the problem to find the SBS of minimum cardinality is of particular interest. Unfortunately, finding one is very hard (e.g. “Williams et al.” algorithm for this problem has a trivial upper bound of kind  $Poly(|C|) \cdot 3^{|X|}$ ).

\*C. Ansotegui, M.-L. Bonet, J. Levy, F. Manyà Measuring the hardness of SAT instances. AAAI 2008.

# Decomposition hardness

The basic idea of Strong backdoors – to decompose a formula into subformulas and evaluate their hardness w.r.t.  $A$  (not necessarily polynomial) gives rise to the following approach, described in e.g. \*.

Let  $A$  be some deterministic complete SAT solver and  $C$  be a CNF formula over variables  $X$ . Denote by  $\mu_{C,A}(\beta \in \{0,1\}^{|B|})$  the runtime of  $A$  on  $C[\beta/B]$ .

Definition 3.

Decomposition hardness (**d-hardness**) of  $C$  w.r.t.  $A$  and  $B \in 2^X$  is the value

$$\mu_{C,A}(B) = \sum_{\beta \in \{0,1\}^{|B|}} \mu_{C,A}(\beta \in \{0,1\}^{|B|})$$

The decomposition hardness of  $C$  w.r.t.  $A$  is the following value:

$$\min_{B \subseteq A} \mu_{C,A}(B)$$

At first glance, d-hardness is almost the same as b-hardness. But it is not.

In case of d-hardness we have to deal with a *complete algorithm*, and SAT for each formula  $C[\beta/B]$  is solved in *finite time*. In case of classical Strong Backdoors the value  $\mu_{A,C}(B)$  makes sense only if  $B$  is an SBS.

\*A. Semenov, D. Chivilikhin, A. Pavlenko, I. Otpuschennikov, V. Ulyantsev, A. Ignatiev Evaluating the hardness of SAT instances using evolutionary optimization algorithms. CP 2021.

# Monte Carlo approach to estimating d-hardness

In case of d-hardness it is possible instead of calculating  $\mu_{A,C}(B)$  exactly to estimate it with some justified accuracy.

Associate with specific  $C, B$  and  $A$  a random variable  $\xi_B$  with finite expected value and variance, such that the following holds:

$$\mu_{A,C}(B) = 2^{|B|} \cdot E[\xi_B] \quad (1)$$

We can estimate  $E[\xi_B]$  using the Monte Carlo method: consider  $\frac{1}{N} \sum_{j=1}^N \xi^j$  instead of  $E[\xi_B]$  keeping in mind the following fact (follows from Chebyshev's inequality):

$$\Pr \left[ (1 - \varepsilon) \mu_{A,C}(B) \leq \frac{2^{|B|}}{N} \sum_{j=1}^N \xi^j \leq (1 + \varepsilon) \mu_{A,C}(B) \right] \geq 1 - \frac{\text{Var}(\xi_B)}{\varepsilon^2 N E[\xi_B]} \quad (2)$$

for any fixed  $\varepsilon, \delta \in (0,1)$ .

$\xi^j$  is the runtime of  $A$  on formula  $C[\beta/B]$  for  $\beta$  randomly and uniformly chosen from  $\{0,1\}^{|B|}$ .

Formula (2) gives us  $(\varepsilon, \delta)$ -approximation\* of  $\mu_{A,C}(B)$  and we can reach admissible values of tolerance  $\varepsilon$  and confidence level  $1 - \delta$  increasing the number of observations  $N$ .

\*R. Karp, M. Luby, N. Madras Monte Carlo approximation algorithms for enumeration problems. J. Algorithms, 10(3): 429 – 448, 1989

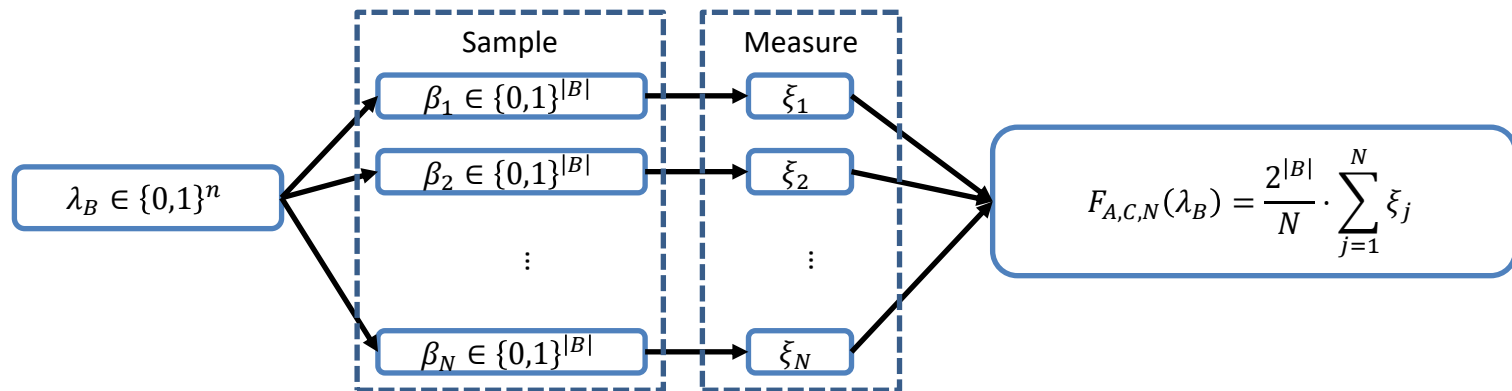
# d-hardness estimation as an optimization problem

Considering all said above we can view the problem of d-hardness estimation for an arbitrary formula  $C$  as finding a set  $B$  such that the estimation

$\frac{2^{|B|}}{N} \sum_{j=1}^N \xi^j$  is as small as possible. We can consider this problem as minimization problem of pseudo-Boolean function

$$F: \{0,1\}^{|X|} \rightarrow \mathbb{R} \quad (3)$$

$\lambda_B \in \{0,1\}^{|X|}$  specifies  $B$  (ones point to variables from  $X$  which appear in  $B$ ).



Function (3) is not analytical and to minimize it we have to use metaheuristic algorithms. In particular in \* and \*\* we use evolutionary algorithms.

\*A. Semenov, D. Chivilikhin, A. Pavlenko, I. Otpuschennikov, V. Ulyantsev, A. Ignatiev Evaluating the hardness of SAT instances using evolutionary optimization algorithms. CP 2021.

\*\*A. Semenov, A. Pavlenko, D. Chivilikhin, S. Kochemazov On probabilistic generalization of Backdoors in Boolean Satisfiability. AAAI 2022.

## $\rho$ – backdoors

In \*we constructed a probabilistic generalization of classical Strong Backdoor Sets (i.e. w.r.t. polynomial sub-solver  $A$ ).

Namely, we defined a  $\rho$  – backdoor ( $\rho \in [0,1]$ ) as such a set  $B$  that the fraction of formulas of kind  $C[\beta/B]$  for which SAT is solved by  $A$  is no less than  $\rho$ .

We also reduced the problem of finding  $\rho$  –backdoors with  $\rho$  close to 1 and relatively small cardinality to a combination of Monte-Carlo sampling and pseudo-Boolean optimization.

Unlike d-hardness  $\rho$  –backdoors do not allow to construct hardness estimations with some reasonable arguments of accuracy, because there can be  $\beta \in \{0,1\}^{|B|}$  such that  $C[\beta/B]$  can not be solved by polynomial sub-solver  $A$ . One can solve such  $C[\beta/B]$  using a complete SAT solver.

As it was demonstrated in \*, sometimes this approach gives quite promising results.

\*A. Semenov, A. Pavlenko, D. Chivilikhin, S. Kochemazov On probabilistic generalization of Backdoors in Boolean Satisfiability. AAAI 2022.

# Finding backdoors in practice

**Input:** CNF  $C$  over  $X$ ,  $|X| = n$  variables, subsolver  $A$ .

- Represent set  $B \subseteq X$  by  $\lambda_B \in \{0,1\}^n$ ,  $\lambda_B = (\lambda_1, \dots, \lambda_n)$ , 
$$\begin{cases} \lambda_i = 1 & \text{if } x_i \in B \\ \lambda_i = 0 & \text{if } x_i \notin B \end{cases}$$

**Goal:** given reasonable amount of computing resources find a good backdoor

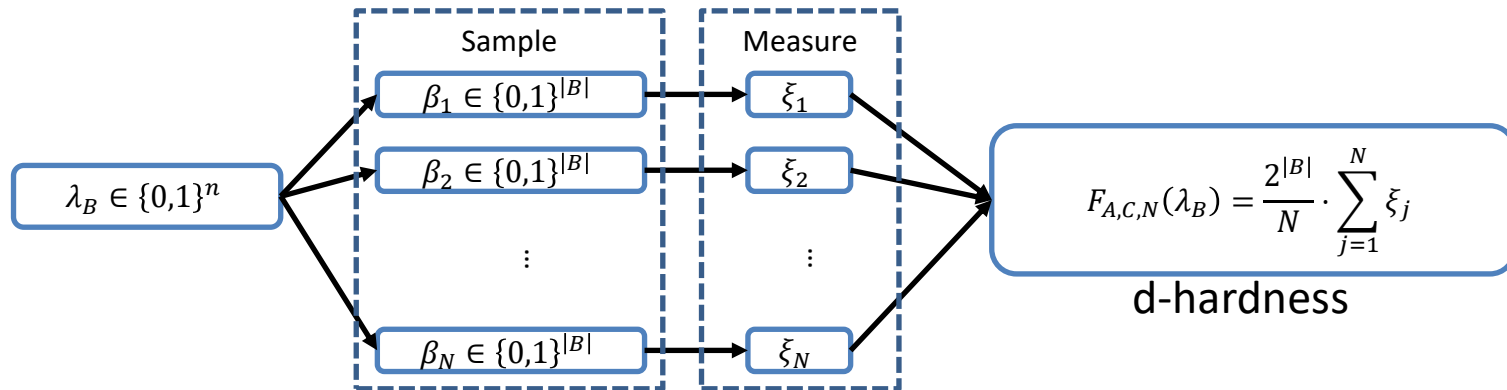
**Fitness function:**  $\lambda_B \rightarrow (0, 2^n)$  (pseudo-Boolean black box function)

1. Construct a sample of  $N$  (different)  $\beta \in \{0,1\}^{|B|}$
2. Run  $A$  on each  $C[\beta/B]$
3. Aggregate the results depending on backdoor type

**Method:**

1. Choose an initial point, set it as a current point
2. Compute fitness function in current point
3. Choose new current point (usually, via a metaheuristic algorithm), go to 2




# Fitness function



	$A$	Sample size	$\xi_i$	Initial point	Fitness function interpretation
Strong Backdoors	UP	$2^{ B }$	$= 1$ if $C[\beta/B]$ is solved by $A$ , 0 otherwise	$0^n$	Portion of <b>all</b> subproblems solved by $A$
d-hardness backdoors	SAT solver	$\leq \max(2^{ B }, 10^6)$	$=$ runtime of $A$ on $C[\beta/B]$	$1^n$	Estimated runtime of $A$ on all subproblems
$\rho$ – backdoors	UP	$\leq \max(2^{ B }, 10^6)$	$= 1$ if $C[\beta/B]$ is solved by $A$ , 0 otherwise	any	<b>Estimated</b> portion of all subproblems solved by $A$

# How to find backdoors faster

1. Reduce the search space
2. Compute fitness function faster

	$A$	Sample size	$\xi_i$	Initial point	Fitness function interpretation
 Strong Backdoors	UP	$2^{ B }$	$= 1$ if $C[\beta/B]$ is solved by $A$ , 0 otherwise	$0^n$	Portion of <b>all</b> subproblems solved by $A$
 d-hardness backdoors	SAT solver	$\leq \max(2^{ B }, 10^6)$	=runtime of $A$ on $C[\beta/B]$	$1^n$	Estimated runtime of $A$ on all subproblems
 $\rho$ – backdoors	UP	$\leq \max(2^{ B }, 10^6)$	$= 1$ if $C[\beta/B]$ is solved by $A$ , 0 otherwise	<i>any</i>	<b>Estimated</b> portion of all subproblems solved by $A$



# UP: example

Given unit literal  $a$ :

1. Remove all clauses with  $a$
2. Remove  $\neg a$  from all clauses with it
3. Apply UP to derived unit literals

PHP (4,3):

$$\begin{aligned} & (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_7 \vee x_8 \vee x_9) \wedge (x_{10} \vee x_{11} \vee x_{12}) \\ & (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_7) \wedge (\neg x_1 \vee \neg x_{10}) \wedge (\neg x_4 \vee \neg x_7) \wedge (\neg x_4 \vee \neg x_{10}) \wedge (\neg x_7 \vee \neg x_{10}) \\ & (\neg x_2 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_8) \wedge (\neg x_2 \vee \neg x_{11}) \wedge (\neg x_5 \vee \neg x_8) \wedge (\neg x_5 \vee \neg x_{11}) \wedge (\neg x_8 \vee \neg x_{11}) \\ & (\neg x_3 \vee \neg x_6) \wedge (\neg x_3 \vee \neg x_9) \wedge (\neg x_3 \vee \neg x_{12}) \wedge (\neg x_6 \vee \neg x_9) \wedge (\neg x_6 \vee \neg x_{12}) \wedge (\neg x_9 \vee \neg x_{12}) \end{aligned}$$

Decide I:  $x_1 = 1$

# UP: example

Given unit literal  $a$ :

1. Remove all clauses with  $a$
2. Remove  $\neg a$  from all clauses with it
3. Apply UP to derived unit literals

PHP (4,3):

$$\begin{aligned} & (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_7 \vee x_8 \vee x_9) \wedge (x_{10} \vee x_{11} \vee x_{12}) \\ & (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_7) \wedge (\neg x_1 \vee \neg x_{10}) \wedge (\neg x_4 \vee \neg x_7) \wedge (\neg x_4 \vee \neg x_{10}) \wedge (\neg x_7 \vee \neg x_{10}) \\ & (\neg x_2 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_8) \wedge (\neg x_2 \vee \neg x_{11}) \wedge (\neg x_5 \vee \neg x_8) \wedge (\neg x_5 \vee \neg x_{11}) \wedge (\neg x_8 \vee \neg x_{11}) \\ & (\neg x_3 \vee \neg x_6) \wedge (\neg x_3 \vee \neg x_9) \wedge (\neg x_3 \vee \neg x_{12}) \wedge (\neg x_6 \vee \neg x_9) \wedge (\neg x_6 \vee \neg x_{12}) \wedge (\neg x_9 \vee \neg x_{12}) \end{aligned}$$

Decide I:  $x_1 = 1$

Propagate I:  $x_4 = 0, x_7 = 0, x_{10} = 0$

# UP: example

Given unit literal  $a$ :

1. Remove all clauses with  $a$
2. Remove  $\neg a$  from all clauses with it
3. Apply UP to derived unit literals

PHP (4,3):

$$\begin{aligned} & (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_7 \vee x_8 \vee x_9) \wedge (x_{10} \vee x_{11} \vee x_{12}) \\ & (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_7) \wedge (\neg x_1 \vee \neg x_{10}) \wedge (\neg x_4 \vee \neg x_7) \wedge (\neg x_4 \vee \neg x_{10}) \wedge (\neg x_7 \vee \neg x_{10}) \\ & (\neg x_2 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_8) \wedge (\neg x_2 \vee \neg x_{11}) \wedge (\neg x_5 \vee \neg x_8) \wedge (\neg x_5 \vee \neg x_{11}) \wedge (\neg x_8 \vee \neg x_{11}) \\ & (\neg x_3 \vee \neg x_6) \wedge (\neg x_3 \vee \neg x_9) \wedge (\neg x_3 \vee \neg x_{12}) \wedge (\neg x_6 \vee \neg x_9) \wedge (\neg x_6 \vee \neg x_{12}) \wedge (\neg x_9 \vee \neg x_{12}) \end{aligned}$$

Decide I:  $x_1 = 1$

Propagate I:  $x_4 = 0, x_7 = 0, x_{10} = 0$

Decide II:  $x_5 = 1$

# UP: example

Given unit literal  $a$ :

1. Remove all clauses with  $a$
2. Remove  $\neg a$  from all clauses with it
3. Apply UP to derived unit literals

PHP (4,3):

$$\begin{aligned} & (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_7 \vee x_8 \vee x_9) \wedge (x_{10} \vee x_{11} \vee x_{12}) \\ & (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_7) \wedge (\neg x_1 \vee \neg x_{10}) \wedge (\neg x_4 \vee \neg x_7) \wedge (\neg x_4 \vee \neg x_{10}) \wedge (\neg x_7 \vee \neg x_{10}) \\ & (\neg x_2 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_8) \wedge (\neg x_2 \vee \neg x_{11}) \wedge (\neg x_5 \vee \neg x_8) \wedge (\neg x_5 \vee \neg x_{11}) \wedge (\neg x_8 \vee \neg x_{11}) \\ & (\neg x_3 \vee \neg x_6) \wedge (\neg x_3 \vee \neg x_{12}) \wedge (\neg x_6 \vee \neg x_{12}) \wedge (\neg x_9 \vee \neg x_{12}) \wedge (\neg x_9 \vee \neg x_{11}) \end{aligned}$$

Decide I:  $x_1 = 1$

Propagate I:  $x_4 = 0, x_7 = 0, x_{10} = 0$

Decide II:  $x_5 = 1$

Propagate II:  $x_2 = 0, x_8 = 0, x_{11} = 0; x_9 = 1; x_3 = 0, x_6 = 0, x_{12} = 0; x_{11} = 1$

**Conflict!**

# Computing fitness for $\rho$ -backdoor

We have CNF  $C$  and want to evaluate the fitness of set  $B = \{x_{i_1}, \dots, x_{i_k}\}$

For  $\beta \in \{0,1\}^k$ ,  $\beta = (\beta_1, \dots, \beta_k)$  we apply UP to  $C[\beta/B]$

If UP derives a conflict when propagating  $x_{i_1} = \beta_1, x_{i_2} = \beta_2, \dots, x_{i_k} = \beta_k$  then  $\xi = 1$ , else  $\xi = 0$ .

**Observation 1\*:** For small  $B$  ( $|B| \leq 20$ ) we can traverse all  $\{0,1\}^{|B|}$  possible  $\beta$  in a tree-like fashion:

- We propagate  $x_{i_1} = \beta_1$ , then propagate  $x_{i_2} = \beta_2$ , etc. **Step by step.**
- If we derive conflict after propagating  $x_{i_l} = \beta_l$ ,  $l < |B|$  we do not need to process any other  $\beta$  with the same  $\beta_1, \dots, \beta_l$

**Observation 2:** By construction, the fitness function prefers  $B$  for which the portion of  $\beta$  on which UP derives conflict is high: for sets with good fitness the depth is usually small

# Search space reduction for $\rho$ -backdoors

## Observation 3:

- If we have a value of a current record, we can stop processing  $B$  if we understand that the resulting fitness function value will be worse than the current record
- Does not agree well with all metaheuristic algorithms

## Observation 4\*:

- The chances of conflict are higher the more literals are propagated.
- We can limit the search space to variables that (individually) result in most propagated literals
- We can evaluate every variable once before finding backdoors, and choose  $h$  (e.g. 200) variables with the highest metric above

\*Credit for the idea goes to Daniil Chivilikhin, ITMO

# Time required to find backdoors comparison

	Baseline	With search space reduction	With search space reduction and tweaks
d-hardness backdoors	Up to 12 hours on 36 CPU cores*	-	-
$\rho$ – backdoors	Up to 12 hours on 36 CPU cores**	20 to 60 minutes on 36 CPU cores	3 seconds on 1 CPU core

$C$	$ B $	$\rho$	$r_{B,ks}$	$r_{B,cd}$
$PvS_{7,4}$	11.4	$0.997 \pm 0.001$	$0.62 \pm 0.08$	$0.68 \pm 0.10$
$BvP_{7,6}$	11.4	$0.997 \pm 0.001$	$0.85 \pm 0.12$	$0.72 \pm 0.11$
$BvP_{8,4}$	11.2	$0.997 \pm 0.002$	$0.87 \pm 0.14$	$0.71 \pm 0.11$
$BvS_{7,7}$	11.4	$0.997 \pm 0.002$	$0.99 \pm 0.16$	$0.78 \pm 0.11$
$PHP_{13,12}$	12.0	$0.997 \pm 0.000$	$0.55 \pm 0.00$	$< 0.36 \pm 0.04$
$par9$	12.2	$0.995 \pm 0.004$	$0.65 \pm 0.02$	$0.48 \pm 0.04$
$pmg12$	13.4	$0.969 \pm 0.014$	$0.12 \pm 0.01$	$0.10 \pm 0.01$
$sgen_{150}^{100}$	13.3	$0.978 \pm 0.011$	$0.29 \pm 0.04$	$0.21 \pm 0.03$

\*A. Semenov, D. Chivilikhin, A. Pavlenko, I. Otpuschennikov, V. Ulyantsev, A. Ignatiev Evaluating the hardness of SAT instances using evolutionary optimization algorithms. CP 2021.

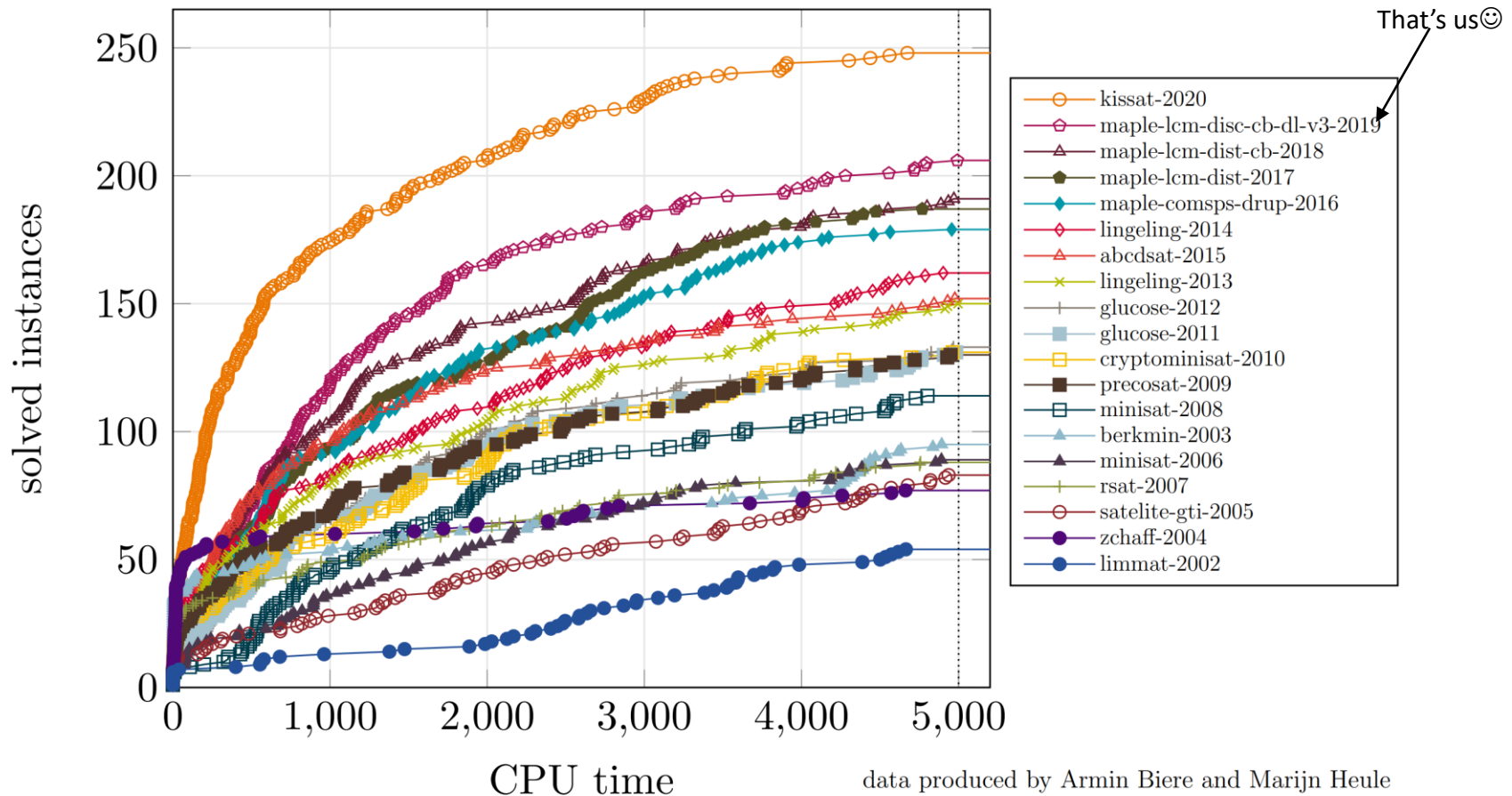
\*\*A. Semenov, A. Pavlenko, D. Chivilikhin, S. Kochemazov On probabilistic generalization of Backdoors in Boolean Satisfiability. AAAI 2022.

# Hopes for the future

CDCL SAT solvers evolve by incorporating new heuristics.

We plan to see whether  $\rho$ -backdoors will make a valuable contribution.

SAT Competition Winners on the SC2020 Benchmark Suite





Thank you for your attention!