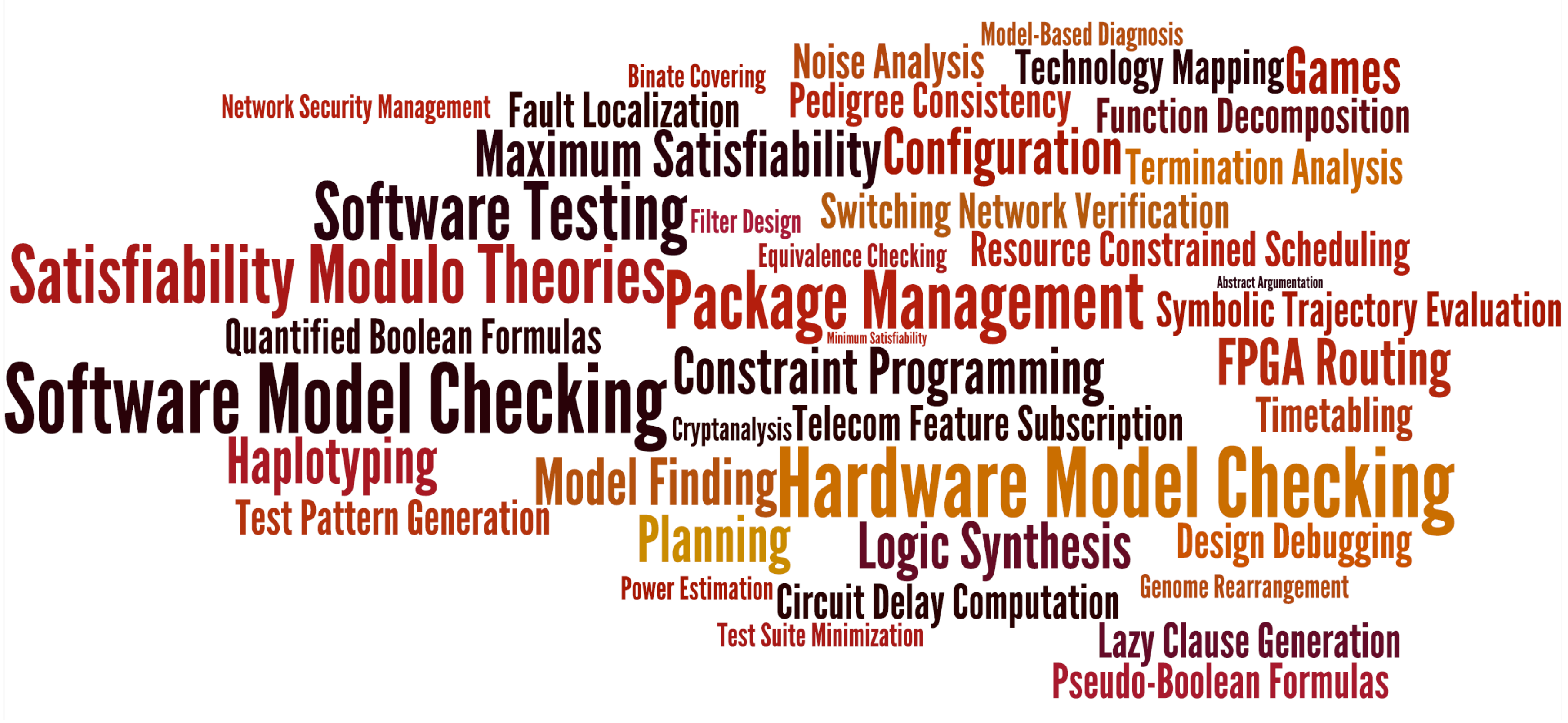
Using Backdoors to estimate the hardness of Boolean formulas w.r.t. SAT solving algorithms

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



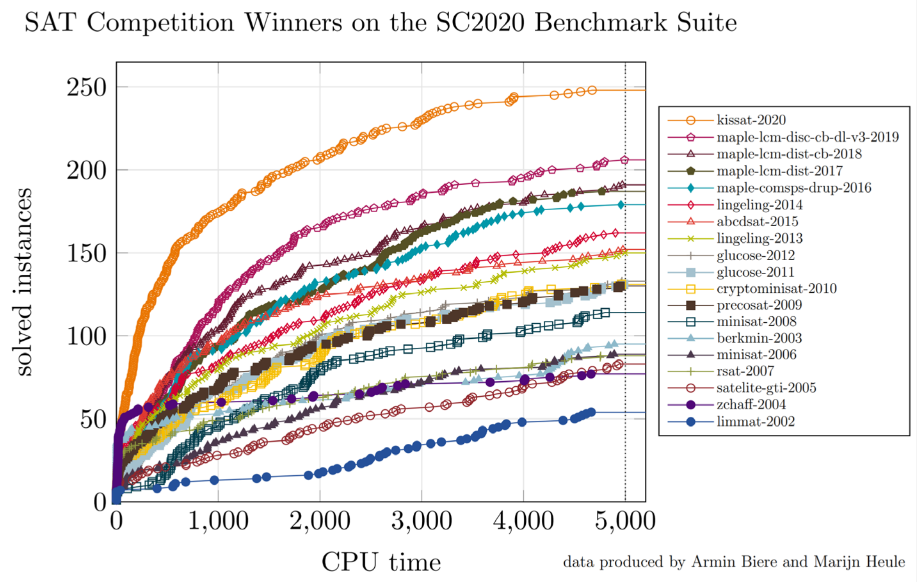
Surprising fact: state-of-the-art SAT solvers can successfully tackle Boolean formulas with millions variables and clauses.

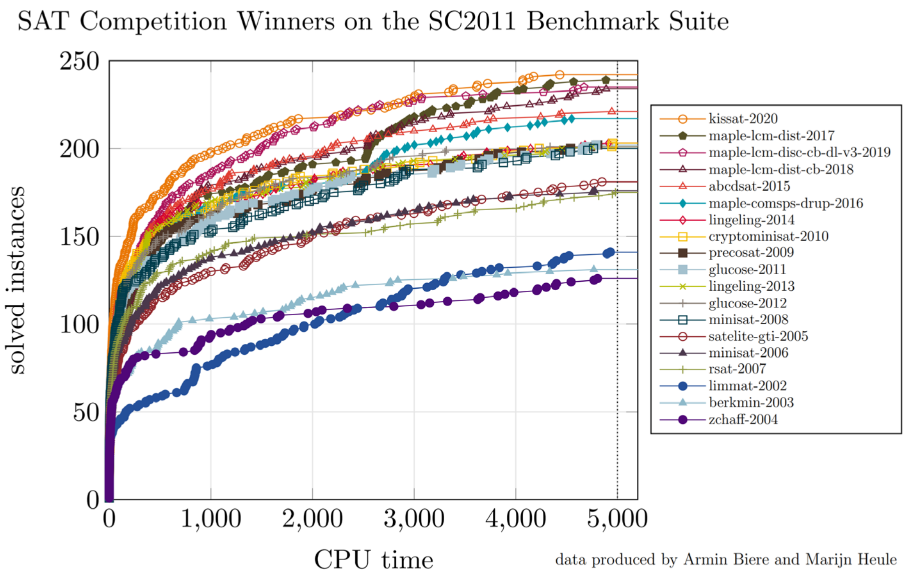
The picture is from the presentation by Joao Marques-Silva at SAT-SMT Summer school 2019

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

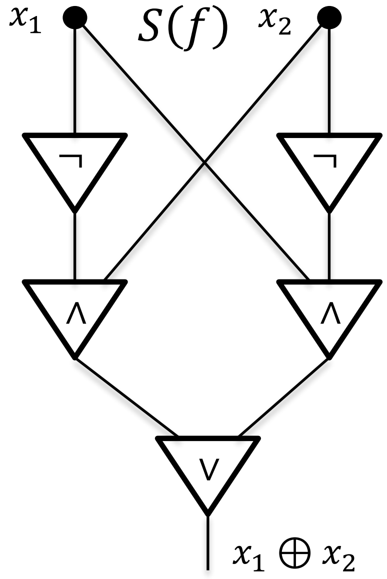


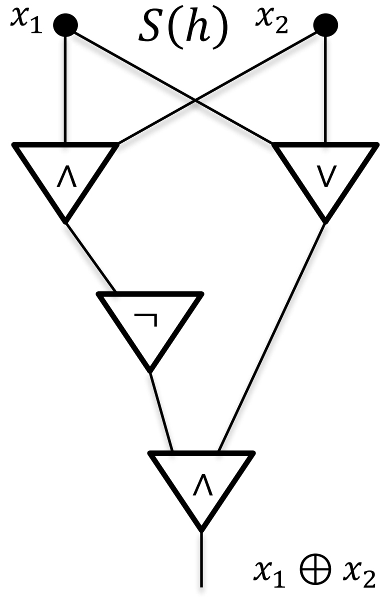


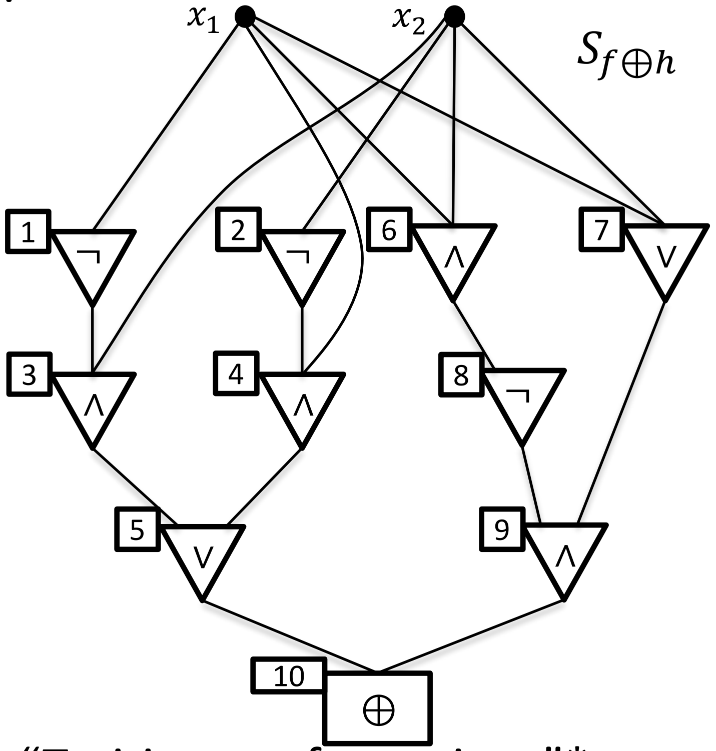
The plots are from http://fmv.jku.at/kissat/

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

α	γ	β
β	α	γ
γ	β	α

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

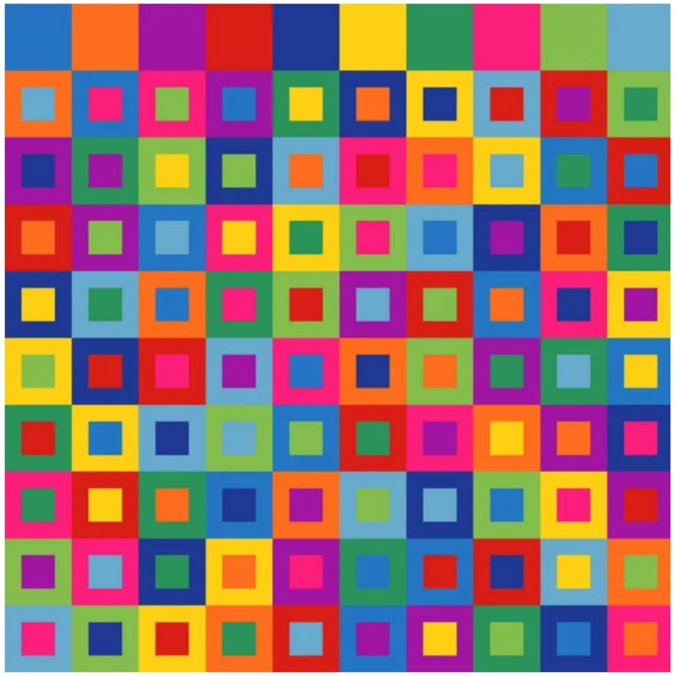
Latin squares exist for any order ≥ 2 (it follows from the properties of Z_n group). The question is, for what orders do Graeco-Latin squares exist?

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

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 β 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:

<u>Definition 2</u> (directly follows from*).
The backdoor hardness (***b-hardness***) of C is the following value:

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

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

<u>Definition 3</u>.
Decomposition hardness (***d-hardness***) of C w.r.t. A and $B \in 2^X$ is the value

$$\mu_{C,A}(B) = \sum\nolimits_{\beta \in \{0,1\}^{|B|}} \mu_{C,A}\left(\beta \in \{0,1\}^{|B|}\right)$$

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 ξ_B with finite expected value and variance, such that the following holds:

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

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

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

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

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

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

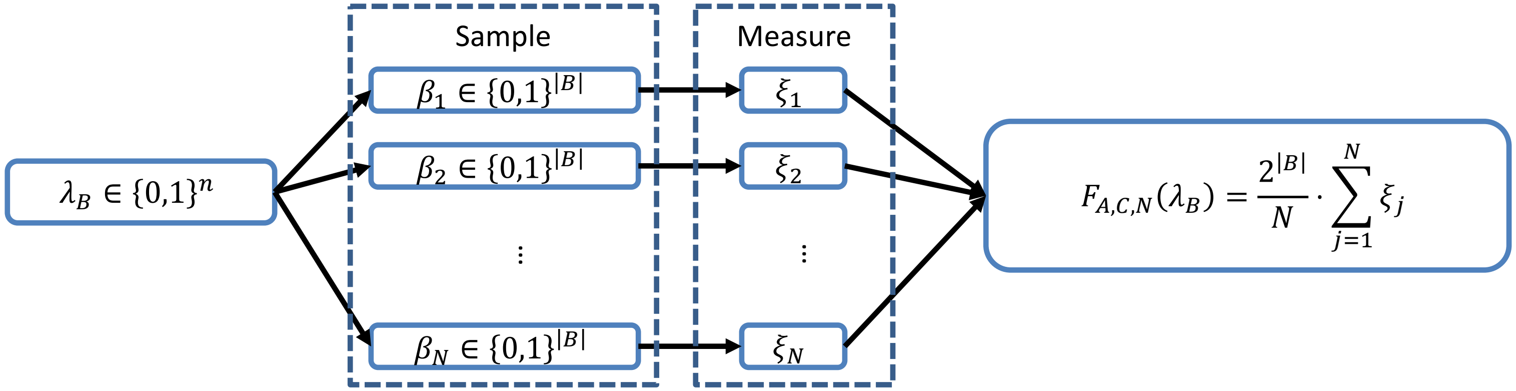
*R. Karp, M. Luby, N. Madras Monte Carlo approximation algorithms for enumeration problems. J. Algorithms, 10(3): 429 – 448, 1989

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

$$F: \{0,1\}^{|X|} \to \mathrm{R} \qquad (3)$$

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



Sample
Measure
$\lambda_B \in \{0,1\}^n$
$\beta_1 \in \{0,1\}^{|B|}$
$\beta_2 \in \{0,1\}^{|B|}$
$\vdots$
$\beta_N \in \{0,1\}^{|B|}$
ξ_1
ξ_2
$\vdots$
ξ_N
$F_{A,C,N}(\lambda_B) = \frac{2^{|B|}}{N} \cdot \sum_{j=1}^{N} \xi_j$

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 ρ.

We also reduced the problem of finding $\rho -$backdoors with ρ 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.

(Very) brief history of SAT solvers: DPLL

For convenience SAT is considered only for formulas in Conjunctive Normal Form (CNF)

1961 – DPLL algorithm *

- Depth-first search in the search tree with *Unit Propagation* and *Backtracking*
- Picks a variable and splits the search tree into a True-branch and False-branch, etc.
- *Unit propagation*: given a unit literal, remove all clauses containing it and remove all complementary literals from CNF
- *Backtracking*: once a conflicting assignment of variables is produced (conflict) – backtrack to the previous level
- Complete algorithm

- **Usually can solve only SAT for instances with several dozen variables**

*Davis, Martin; Logemann, George; Loveland, Donald (1961). "A Machine Program for Theorem Proving". Communications of the ACM. 5 (7): 394–397

(Very) brief history of SAT solvers: BDD and SLS

1965 – Resolution method *

- Uses the resolution rule to derive new clauses
- Uses unit propagation to simplify clauses
- **Is very good for ATP but weaker than other concepts for SAT**

1986 – Binary Decision Diagrams (BDD) **

- The idea is to represent a formula as a graph of a special kind
- Reduced Ordered BDD (ROBDD) are canonical
- Constructing ROBDD allows to effectively solve #SAT problem
- **Can be useful when one needs to see the multitude of states of a discrete dynamic system**

*Davis, Martin; Putnam, Hilary (1960). "A Computing Procedure for Quantification Theory". J. ACM. 7 (3): 201–215.
*J. A. Robinson. 1965. A Machine-Oriented Logic Based on the Resolution Principle. J. ACM 12, 1965, 23–41.
**Randal E. Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. Comput. 35, 8, 1986, 677–691.

(Very) brief history of SAT solvers: BDD and SLS

1992 – Stochastic Local Search (SLS) *

- Uses local search techniques to traverse the space of all satisfying assignments
- Elementary operation is variable flip
- **Incomplete algorithm, can not prove unsatisfiability for most instances**

1996 – Lookahead **

- A variant of DPLL, which uses sophisticated decision heuristics to choose variables to branch on
- Quite effective on small hard instances

*B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In Proceedings of AAAI'92, pages 440–446. MIT Press, 1992.
*J. Gu. Efficient Local Search for Very Large-Scale Satisfiability Problems. SIGART Bulletin, 3:8–12, 1992.

**Max Bohm and Ewald Speckenmeyer. A fast parallel sat-solver - efficient workload balancing. Ann. Math. Artif. Intell., 17(3-4):381– 400, 1996.

(Very) brief history of SAT solvers: CDCL

1996 – Conflict Driven Clause Learning *

- Based on DPLL
- First main idea: use conflicts to add new clauses and direct future search
- Second main idea: use dynamic variable order based on heuristics
- Conflict analysis allows to determine the cause of conflict and perform backjumping: backtracking several levels simultaneously
- Heavily relies on memory use to store learnt clauses
- Requires effective heuristics to manage learnt clauses, variable selection heuristics, and other aspects
- Very reliant on implementation: several specific programming techniques (e.g. watched literals) boost the performance by tens of percents

- **Very effective in practice**
- **Predominant SAT solving concept today**

*J.P. Marques-Silva; Karem A. Sakallah (November 1996). "GRASP-A New Search Algorithm for Satisfiability". Digest of IEEE International Conference on Computer-Aided Design (ICCAD). pp. 220–227

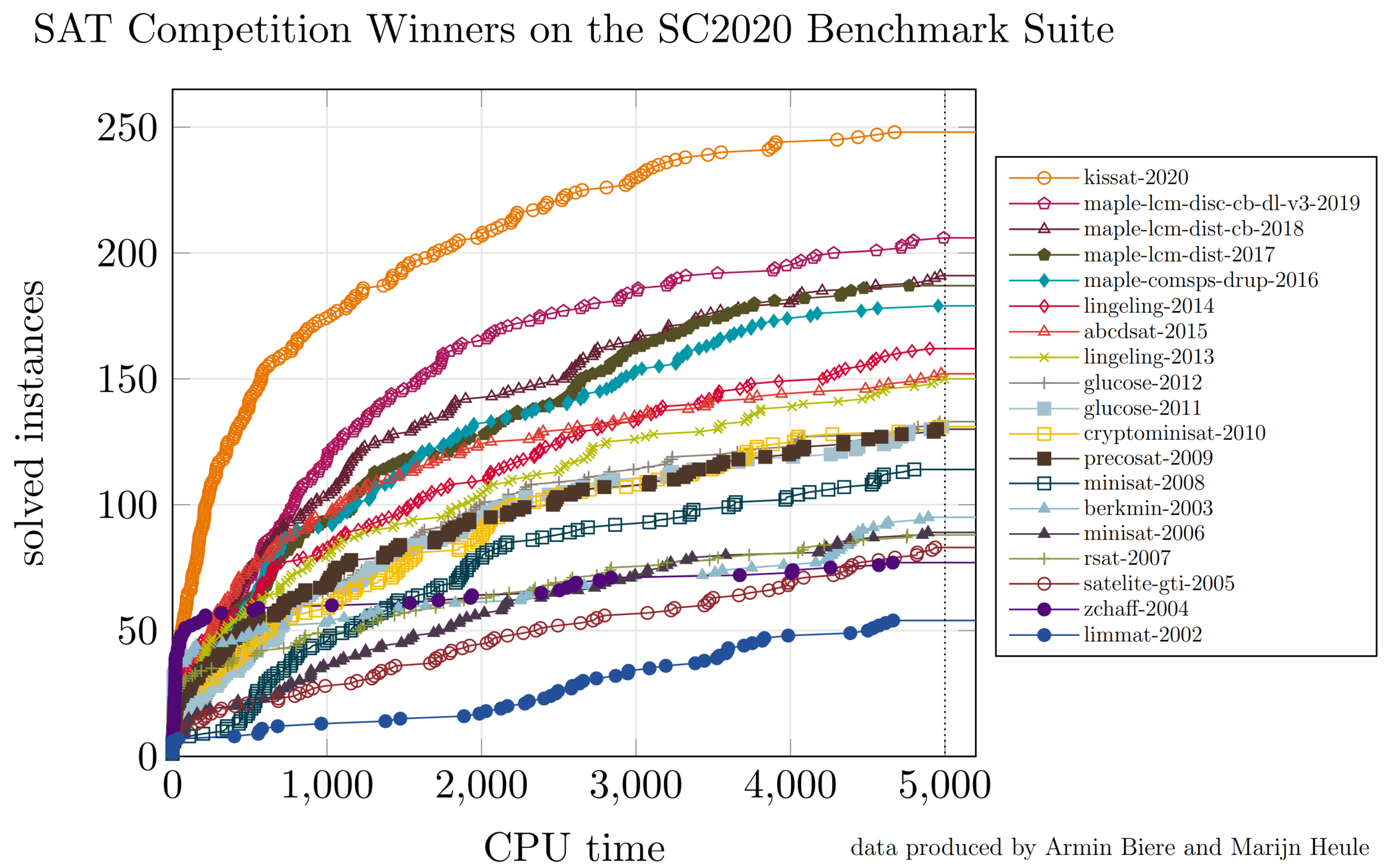
CDCL today

The evolution of CDCL solvers today moved towards using heuristics:

- Restarts: periodically backtrack to decision level 0 to perform a soft reset of an algorithm
- Preprocessing: use methods involving derivation rules different from DPLL to simplify the original CNF before feeding it to SAT solver
- Inprocessing: interleave CDCL phases with preprocessing-like methods (between restarts) to simplify the formula during solving
- Hybrid solving: incorporate an SLS component to work alongside CDCL
- Parallel and cloud SAT solvers
- Use ML methods in general and neural networks in particular (currently, not very successful direction, but a lot of potential)
- Decomposition-based methods: Cube-and-Conquer, **Backdoors**

SAT Competitions

SAT competition is an annual event established in 2002. Its goal is to promote the development of new SAT solving techniques and evaluate them in a competitive environment



Thank you for your attention!