Using Backdoors to Generate Learnt Information in SAT Solving

Alexander Andreev, **Konstantin Chukharev**, Stepan Kochemazov, Alexander Semenov

24 October 2024

27th European Conference on Artificial Intelligence — ECAI-2024 — Santiago de Compostela, Spain

Contents

Background: Backdoors

Backdoors by Williams (2003)

Consider a SAT problem for a CNF C over the set of variables X.

A **strong backdoor** (*with respect to an algorithm* A) is a subset of variables $B \subseteq X$ such that for **each** assignment α of these variables, $C[\alpha/B]$ is *easy*, that is, polynomially decidable using A. [1]

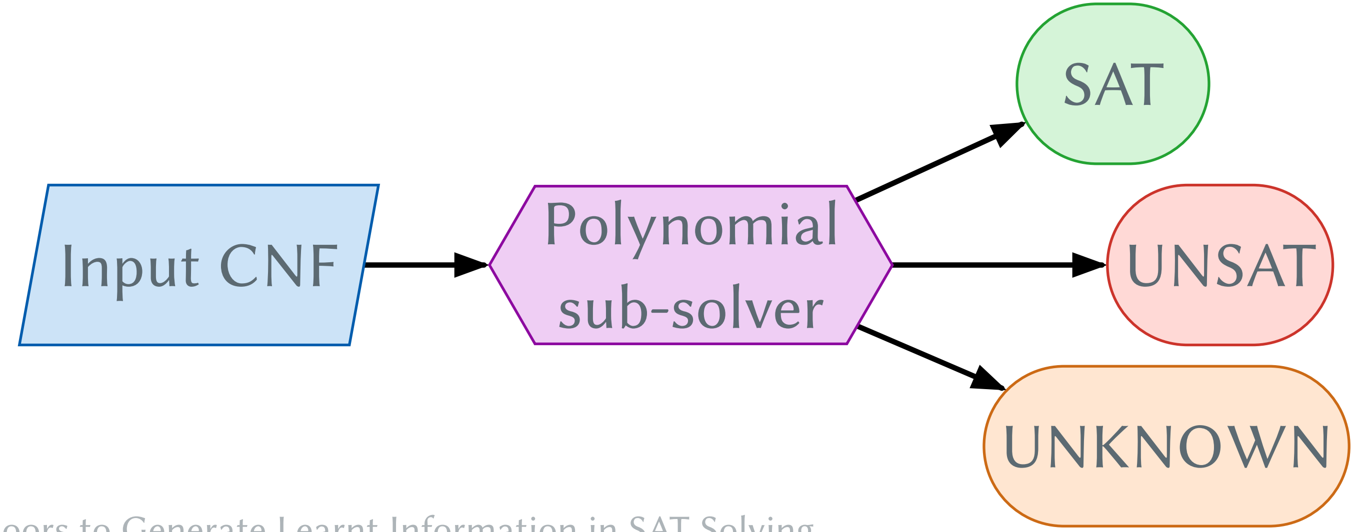
Overall, strong backdoor is a complete **unsatisfiability certificate** which can be verified in $O\left(p(|C|) \cdot 2^{|B|}\right)$.

[1] R. Williams, C. P. Gomes, and B. Selman, "Backdoors to Typical Case Complexity," in IJCAI, 2003.

Sub-solver

A **sub-solver** is a (SAT/CSP) solving algorithm that, for a given input C, in *polynomial time* (i.e. $O(p(|C|))$), either "*determines*" C correctly (UNSAT or SAT with a model) or "*rejects*" the input ("UNKNOWN" result).

For a **sub-solver**, one can employ the **unit propatation (UP)** rule.



Strong backdoors: Visualized

⚠ Hereinafter, assume that the considered CNF C is **unsatisfiable**.

For each assignment $\alpha \in \{0,1\}^{|B|}$, unit propagation on $C[\alpha/B]$ can return either UNSAT (**conflict** under assignment), or UNKNOWN (**no conflict**).

For a strong backdoor, **all** assignments lead to a conflict:

$$C \wedge (x_1 \wedge x_2 \wedge \ldots \wedge x_n) \vdash_{\text{UP}} \bot \text{ (conflict)}$$
$$C \wedge (\overline{x}_1 \wedge x_2 \wedge \ldots \wedge x_n) \vdash_{\text{UP}} \bot \text{ (conflict)}$$
$$\ldots$$
$$C \wedge (\overline{x}_1 \wedge \overline{x}_2 \wedge \ldots \wedge \overline{x}_n) \vdash_{\text{UP}} \bot \text{ (conflict)}$$

Main matter: ρ-backdoors

Desired properties of "better" backdoors

Our goal is to construct **a thing** with the following properties:

- It is similar to a strong backdoor.
- It is a *partial* unsatisfiability certificate.
- It is small and easy to find.
- It can be used to obtain logical entailments of the original formula, which might be beneficial for a SAT solver.

We are going to show that "***ρ*-backdoors**" have these properties.

ρ-backdoors: Visualized

$$B = \{x_1, x_2, x_3\}$$

$$C \wedge (\overline{x}_1 \wedge \overline{x}_2 \wedge \overline{x}_3) \vdash_{\mathrm{UP}} \bot$$
$$C \wedge (\overline{x}_1 \wedge \overline{x}_2 \wedge x_3) \vdash_{\mathrm{UP}} \bot$$
$$C \wedge (\overline{x}_1 \wedge x_2 \wedge \overline{x}_3) \not\vdash_{\mathrm{UP}} \bot$$
$$C \wedge (\overline{x}_1 \wedge x_2 \wedge x_3) \not\vdash_{\mathrm{UP}} \bot$$
$$C \wedge (x_1 \wedge \overline{x}_2 \wedge \overline{x}_3) \vdash_{\mathrm{UP}} \bot$$
$$C \wedge (x_1 \wedge \overline{x}_2 \wedge x_3) \vdash_{\mathrm{UP}} \bot$$
$$C \wedge (x_1 \wedge x_2 \wedge \overline{x}_3) \vdash_{\mathrm{UP}} \bot$$
$$C \wedge (x_1 \wedge x_2 \wedge x_3) \not\vdash_{\mathrm{UP}} \bot$$

Legend: easy hard

$$\rho = 1 - 3/8 = 0.625$$

ρ is the proportion of "easy" tasks

Properties of ρ-backdoors

- ρ-backdoor is a **partial** unsatisfiability certificate: *not for all* assignments $\alpha \in \{0,1\}^{|B|}$ the formula $C[\alpha/B]$ has an "easy" proof.
- Generally, there are **many** small ρ-backdoors with ρ close to 1. 80% of SAT Comp instances have ρ-backdoors with $|B| < 20$ and $\rho > 0.8$
- ρ can be calculated very fast using **tree-based** unit propagation. [AAAI'23]
- Multiple small ρ-backdoors can be combined into larger ρ-backdoors with **strictly** greater ρ. [AAAI'22, **ECAI-2024**]
- New **learnt clauses** can be derived from a ρ-backdoor. [**ECAI-2024**]

Properties of ρ-backdoors

- ρ-backdoor is a **partial** unsatisfiability certificate: *not for all* assignments $\alpha \in \{0,1\}^{|B|}$ the formula $C[\alpha/B]$ has an "easy" proof.
- Generally, there are **many** small ρ-backdoors with ρ close to 1.
  80% of SAT Comp instances have ρ-backdoors with $|B| < 20$ and $\rho > 0.8$
- ρ can be calculated very fast using **tree-based** unit propagation. [AAAI'23]
- Multiple small ρ-backdoors can be combined into larger ρ-backdoors with **strictly** greater ρ. [AAAI'22, **ECAI-2024**]
- New **learnt clauses** can be derived from a ρ-backdoor. [**ECAI-2024**]

Properties of ρ-backdoors

- ρ-backdoor is a **partial** unsatisfiability certificate: *not for all* assignments $\alpha \in \{0,1\}^{|B|}$ the formula $C[\alpha/B]$ has an "easy" proof.
- Generally, there are **many** small ρ-backdoors with ρ close to 1.
  80% of SAT Comp instances have ρ-backdoors with $|B| < 20$ and $\rho > 0.8$
- ρ can be calculated very fast using **tree-based** unit propagation. [AAAI'23]
- Multiple small ρ-backdoors can be combined into larger ρ-backdoors with **strictly** greater ρ. [AAAI'22, **ECAI-2024**]
- New **learnt clauses** can be derived from a ρ-backdoor. [**ECAI-2024**]

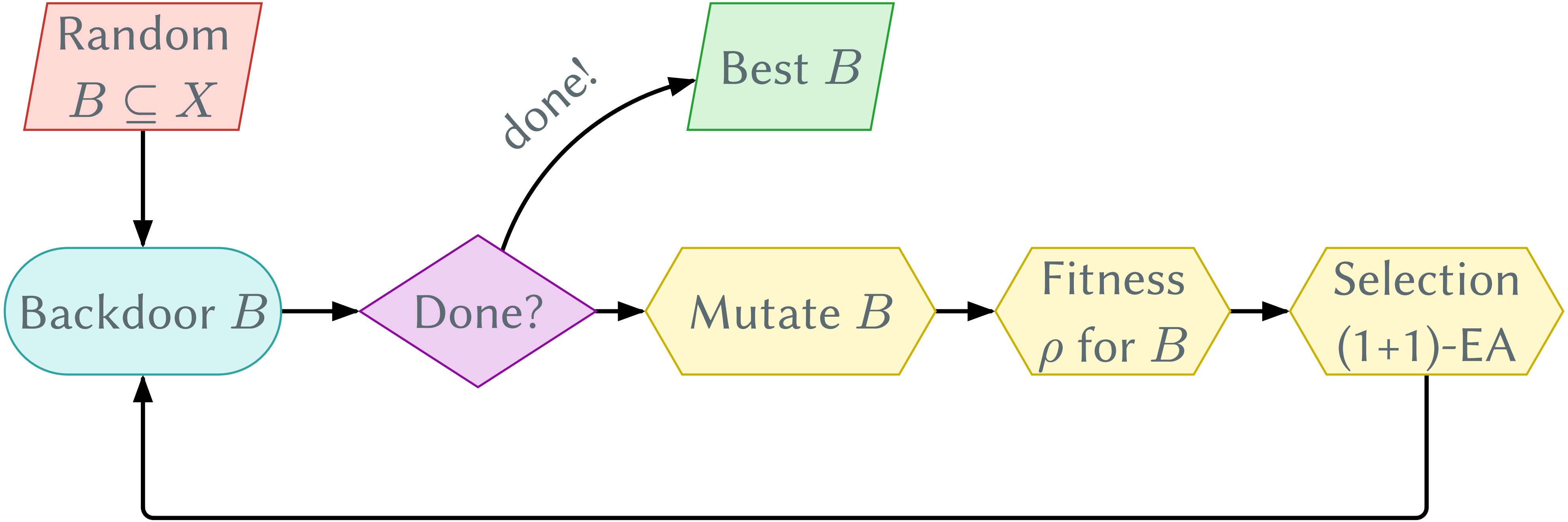
Properties of ρ-backdoors

- ρ-backdoor is a **partial** unsatisfiability certificate: *not for all* assignments $\alpha \in \{0,1\}^{|B|}$ the formula $C[\alpha/B]$ has an "easy" proof.
- Generally, there are **many** small ρ-backdoors with ρ close to 1. 80% of SAT Comp instances have ρ-backdoors with $|B| < 20$ and $\rho > 0.8$
- ρ can be calculated very fast using **tree-based** unit propagation. [AAAI'23]
- Multiple small ρ-backdoors can be combined into larger ρ-backdoors with **strictly** greater ρ. [AAAI'22, **ECAI-2024**]
- New **learnt clauses** can be derived from a ρ-backdoor. [**ECAI-2024**]

Properties of ρ-backdoors

- ρ-backdoor is a **partial** unsatisfiability certificate: *not for all* assignments $\alpha \in \{0,1\}^{|B|}$ the formula $C[\alpha/B]$ has an "easy" proof.
- Generally, there are **many** small ρ-backdoors with ρ close to 1.
  80% of SAT Comp instances have ρ-backdoors with $|B| < 20$ and $\rho > 0.8$
- ρ can be calculated very fast using **tree-based** unit propagation. [AAAI'23]
- Multiple small ρ-backdoors can be combined into larger ρ-backdoors with **strictly** greater ρ. [AAAI'22, **ECAI-2024**]
- New **learnt clauses** can be derived from a ρ-backdoor. [**ECAI-2024**]

Searching for ρ-backdoors



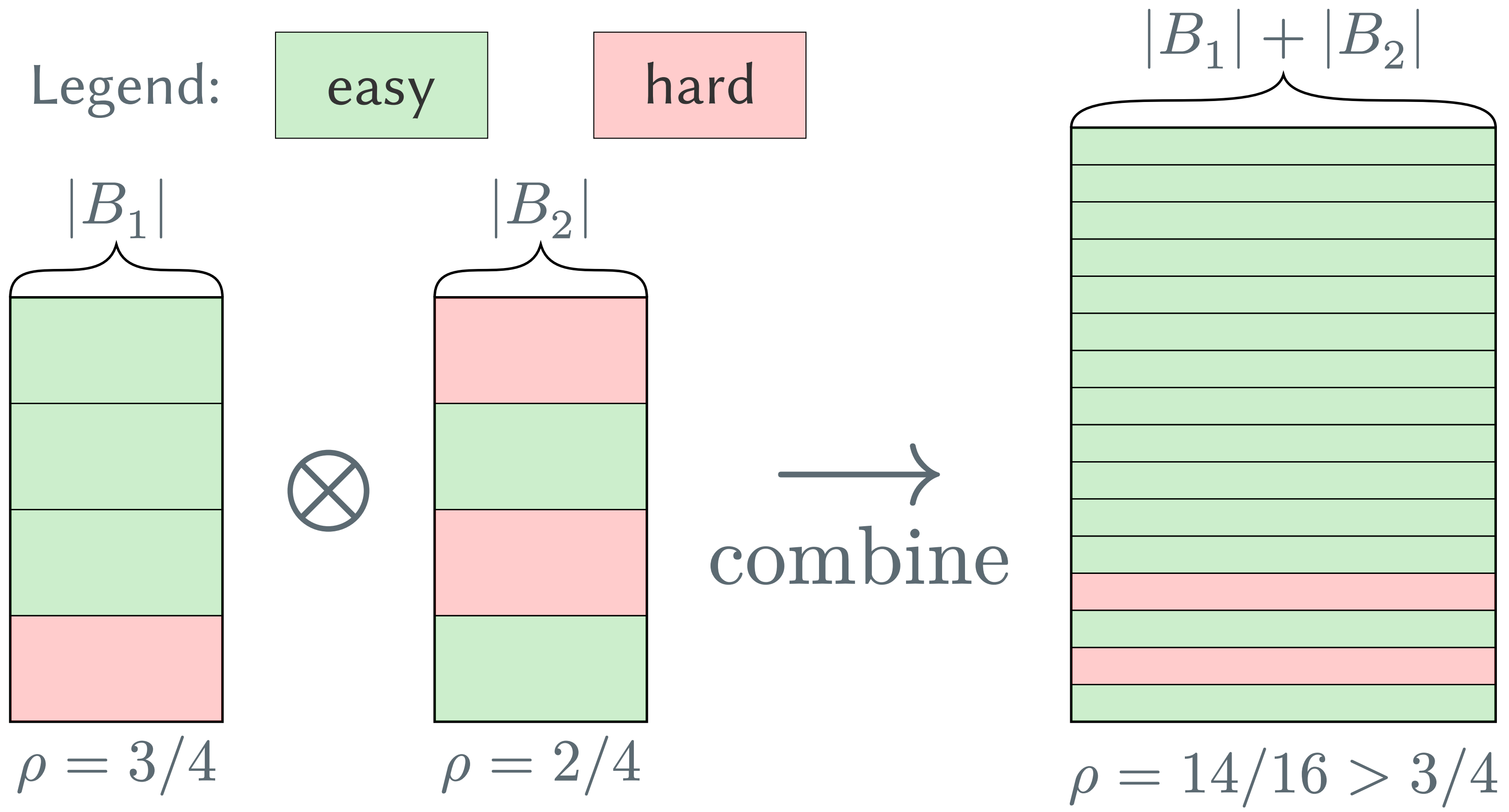
Combining ρ-backdoors

Theorem. It is possible to use two small ρ-backdoors B_1 and B_2 (for simplicity, assume $B_1 \cap B_2 = \varnothing$) to construct a ρ-backdoor $B' = B_1 \cup B_2$ of larger size $|B'| = |B_1| + |B_2|$ with $\rho_3 > \max(\rho_1, \rho_2)$.

For this, just perform a **Cartesian product** of the two sets of hard tasks, **concatenating** the cubes and filtering out all trivially conflicting ones.

Note: no need to check all $2^{|B_1| + |B_2|}$ cubes, since most of them (ρ_1 and ρ_2) have been proven to be conflicting. We get **larger** ρ-backdoor **for free**!

Combination of ρ-backdoors: Visualized



Deriving clauses from ρ-backdoors

Hard cubes:

$\overline{x}_1\ x_2\ \overline{x}_3\ \overline{x}_4\ \overline{x}_5$

$\overline{x}_1\ x_2\ x_3\ \overline{x}_4\ \overline{x}_5$

$x_1\ \overline{x}_2\ \overline{x}_3\ x_4\ x_5$

$x_1\ \overline{x}_2\ x_3\ x_4\ \overline{x}_5$

$x_1\ x_2\ \overline{x}_3\ x_4\ x_5$

$$\rho = 1 - 5/32 = 0.844$$

Count table:

(x_i, x_j)	$x_i x_j$	$x_i \overline{x}_j$	$\overline{x}_i x_j$	$\overline{x}_i \overline{x}_j$	Derived clauses
(x_1, x_2)	1	2	2	0	(x_1, x_2)
(x_1, x_3)	1	2	1	1	
(x_1, x_4)	3	0	0	2	$(\overline{x}_1, x_4)$, $(x_1, \overline{x}_4)$
(x_1, x_5)	2	1	0	2	$(x_1, \overline{x}_5)$
(x_2, x_3)	1	2	1	1	
(x_2, x_4)	1	2	2	0	(x_2, x_4)
(x_2, x_5)	1	2	1	1	
(x_3, x_4)	1	1	2	1	
(x_3, x_5)	0	2	2	1	$(\overline{x}_3, \overline{x}_5)$
(x_4, x_5)	2	1	0	2	$(x_4, \overline{x}_5)$

Interleave procedure

Overview of the INTERLEAVE procedure

- **Alternate** between the two phases:
  1. **ρ-backdoor phase**: identify and combine ρ-backdoors, derive clauses.
  2. **CDCL solving phase**: just run CaDiCaL for some time.
- Each phase is granted an initial conflict **budget**, which increases incrementally (e.g., by a factor of 1.1) after each round.

Step-by-Step process (7 steps)

0. **Initialize** the CDCL SAT solver on the given CNF.
1. **Pre-solve** the CNF using a limited conflict budget (e.g., 10 000 conflicts).
2. **Iterative process**: alternate ρ-backdoor phase and plain CDCL solving.

Iterative process: ρ-backdoor identification

3. **Identify ρ-backdoor** using Evolutionary Algorithm.
   - Determine hard tasks for the ρ-backdoor using UP.
   - Augment CaDiCaL to propagate multiple cubes in a tree-like manner.
4. **Combine ρ-backdoors** using Cartesian product.
   - Combine previously found hard tasks with those for the new backdoor.
   - Reset the large set of hard tasks (e.g, once it exceeds 10 000 cubes).

Iterative process: Limited filtering

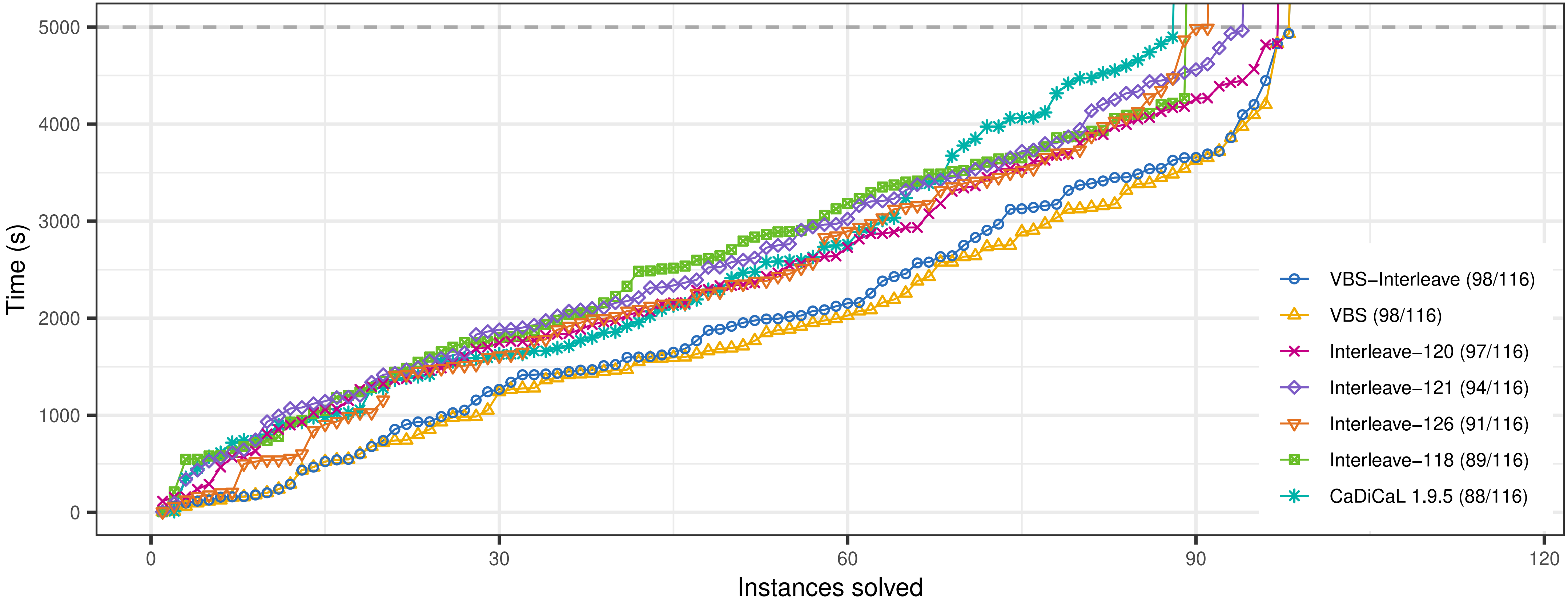
5. **Filter** the set of hard tasks using *limited* solver (yet polynomial).
   - Use conflict budget of, e.g., 1000 conflicts per cube, and 100 000 total.
   - Use heuristic to select the most promising cubes.
6. **Derive clauses** from the set of hard tasks.

Iterative process: Exit condition

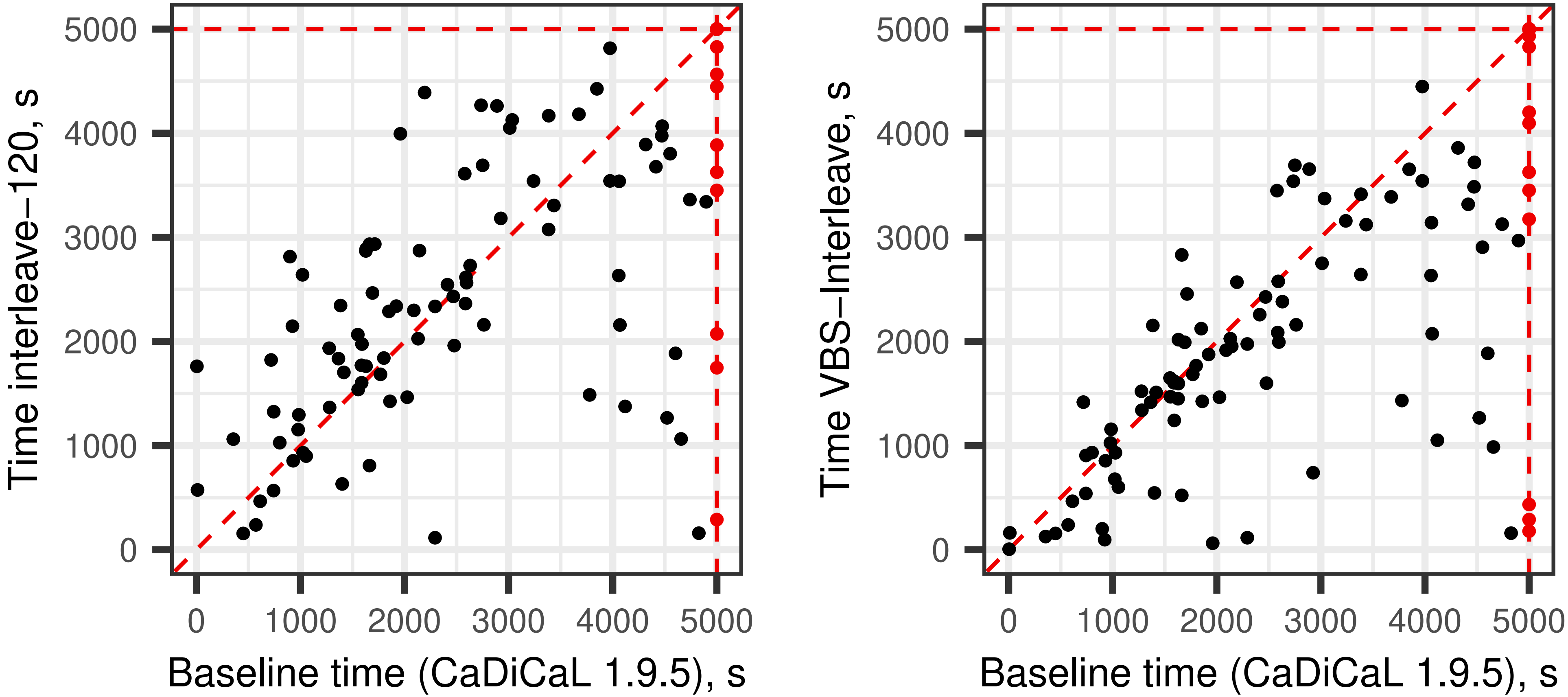
7. **Exit** the iterative process if:
   - The satisfying assignment is found — problem is SAT.
   - The set of hard tasks is empty, i.e., the strong backdoor is found — problem is UNSAT with a polynomial certificate.
   - **Else**, switch to other mode (plain CDCL) and continue.

Experimental evaluation

Results: Cactus plot on 116 instances from SAT Competition



Results: Scatter plot for the best configuration and VBS



Conclusion

Conclusion and Future plans

- In this work, we further explored the concept of ρ-backdoors:
  - Partial unsatisfiability certificates.
  - Easy to search for using evolutionary algorithms.
  - Can be used to derive new clauses that are beneficial for a SAT solver.
- Open-source implementation is available on GitHub:
  https://github.com/Lipen/backdoor-solver
- Future plans:
  - Parallel implementation.
  - Proofs!

Thank you for your attention!

kchukharev@itmo.ru

Lipen

Some questions you *might* want to ask:

- How long the EA runs comparing to solving? [*Very little.*]
- Have you tried BDDs for storing hard tasks? [*Yes! It exploded.*]
- Have you tried using a sub-solver other than UP? [*Yes, we tried.*]