On the non-Degeneracy of Unsatisfiability Proof Graphs produced by SAT Solvers

Rohan Fossé* Laurent Simon*

Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR-5800

Preliminaries

What is SAT ?

Definition

Let $\Phi(a, b, ..)$ be a boolean formula.

Is there an **interpretation** of $(a, b, ..)$ that satisfies Φ ?

In theory

Cook-Levin's Theorem: SAT is **NP-Complete**.

$\Rightarrow$ First problem proved **NP-Complete**

$\Rightarrow$ $P = NP$?

Notations

Literals

A literal $(a, b, \ldots)$ is either a boolean variable x or the negation of a boolean variable $\neg x$

Clauses

A clause C is a disjunction of literals *i.e*:
$C = a \vee b \vee .. \vee z$

Formula

A formula Φ is a conjunction of clauses *i.e*:
$\Phi = C_1 \wedge C_2 \wedge ... \wedge C_m$

Example

$\Phi = (a \vee \neg b) \wedge b \wedge (\neg a \vee \neg b)$

Resolution rule [Robinson '65]

Let C_1 and C_2 two clauses such that:

$$C_1 = a \lor b \lor c \lor \mathrm{d}$$

$$C_2 = \neg d \lor e \lor f$$

We apply the resolution rule on d:

$$\overbrace{(a \lor b \lor c \lor d)}^{C_1} \land \overbrace{(\neg d \lor e \lor f)}^{C_2}$$
$$\vdash a \lor b \lor c \lor e \lor f$$

Resolution rule [Robinson '65]

More formally,

Let C_1 and C_2 be two clauses, the resolution rule gives us:

$$(C_1 \vee x) \wedge (C_2 \vee \neg x) \vdash C_1 \vee C_2$$

We call $C_1 \vee C_2$ the resolvent of $C_1 \vee x$ and $C_2 \vee \neg x$.

Resolution rule [Robinson '65]

More formally,

Let C_1 and C_2 be two clauses, the resolution rule gives us:

$$(C_1 \vee x) \wedge (C_2 \vee \neg x) \vdash C_1 \vee C_2$$

We call $C_1 \vee C_2$ the resolvent of $C_1 \vee x$ and $C_2 \vee \neg x$.

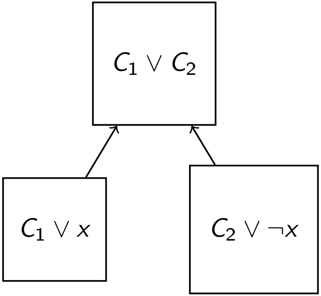


Figure 1: Graphical representation of the resolution

Inconsistency

Correction and completeness of the resolution

→ Φ is an inconsistent formula: Empty clause ($\Box$) can be derivated from the clauses of Φ;

→ An inconsistent proof represents the sequence of resolutions.

In this talk

We will consider only inconsistent proof (Satisfiability proof are trivial).

Modern SAT solver

Φ: Set of initials clauses

Σ: Set of learnts clauses

Algorithm 1 Modern SAT solver

While $\square \notin \Phi \cup \Sigma$ **do**
  $C \leftarrow \mathit{learntClauses}()$
  $\Sigma = \Sigma \cup C$
  If $\mathit{overfull}(\Sigma)$ **Then**
    $\Delta = \mathit{clausesToDelete}(\Sigma)$
    $\Sigma = \Sigma \backslash \Delta$
  End If
End While

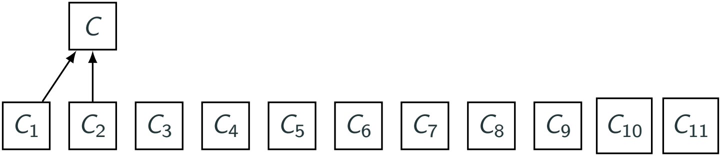
Sequence of resolution with a graph representation

Formula

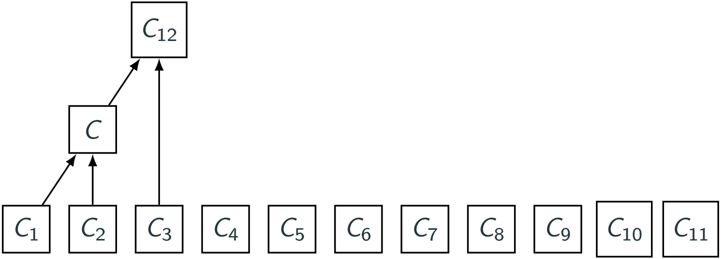
Let $\Phi = C_1 \wedge C_2 \wedge .. \wedge C_{11}$, s.t. $\forall i \in [1..11]$, C_i any clause.

$$\boxed{C_1} \quad \boxed{C_2} \quad \boxed{C_3} \quad \boxed{C_4} \quad \boxed{C_5} \quad \boxed{C_6} \quad \boxed{C_7} \quad \boxed{C_8} \quad \boxed{C_9} \quad \boxed{C_{10}} \quad \boxed{C_{11}}$$

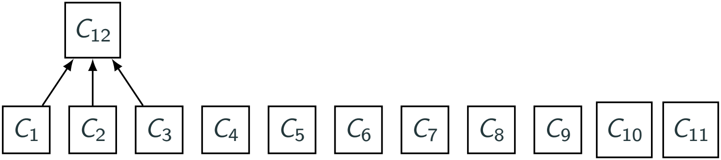
Sequence of resolution with a graph representation



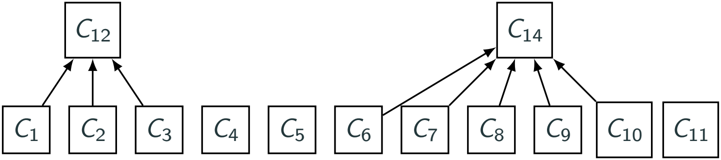
Sequence of resolution with a graph representation



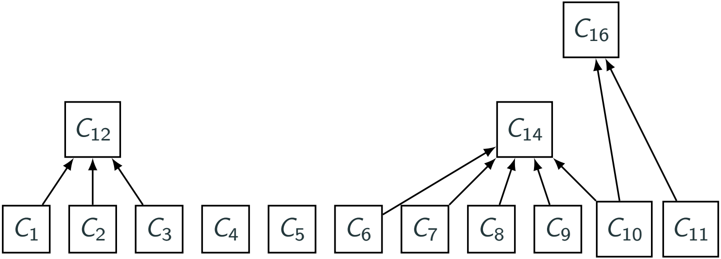
Sequence of resolution with a graph representation



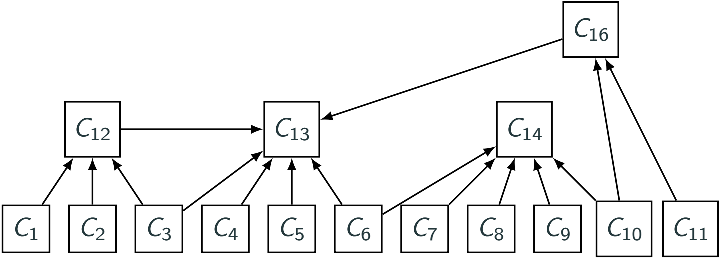
Sequence of resolution with a graph representation



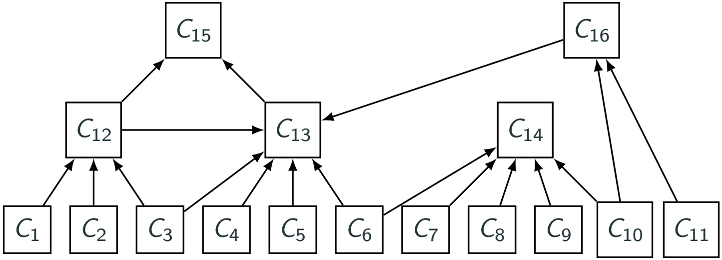
Sequence of resolution with a graph representation



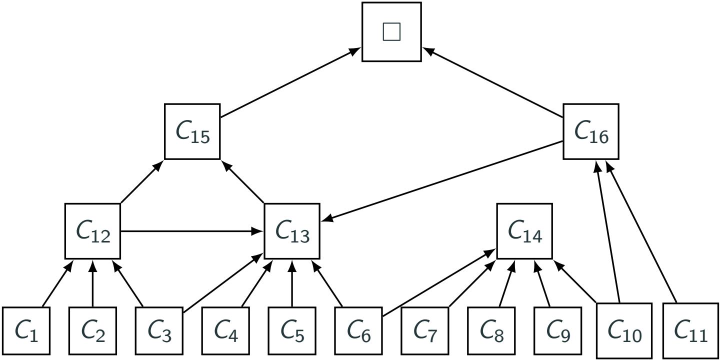
Sequence of resolution with a graph representation



Sequence of resolution with a graph representation



Sequence of resolution with a graph representation

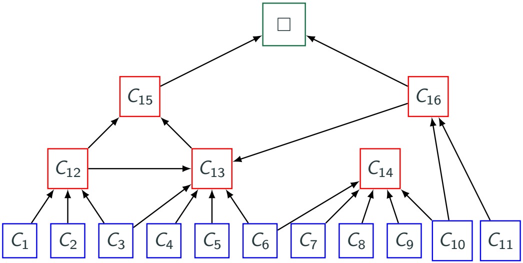


Resolution graph

Definition

The resolution graph is a directed acyclic graph (or **DAG**) such that:

- Leaves are **initials** clauses;
- Internal nodes are **learnts** clauses;
- The root is the **empty** clause.



We call it the proof produced by the SAT solver.

Characterization of K-Cores

Representation of a real proof

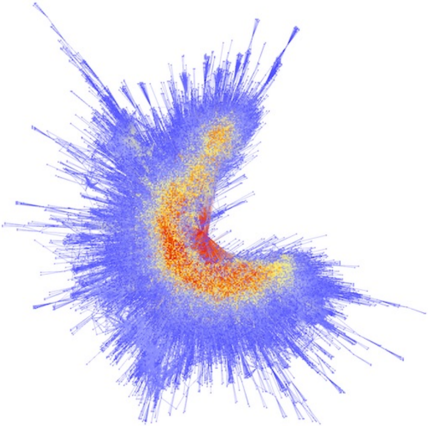


Figure 2: Force-Directed layout of the Dependency Graph for the benchmark `een-pico-prop-05`. The color shows the **degree** of each node.

Information

Formula

\# clauses: 55585

\# variables: 50076

Conflicts

\# conflicts: 59792

CPU time: **6s**

Graph

\# vertices: 51274

\# edges: **960620**

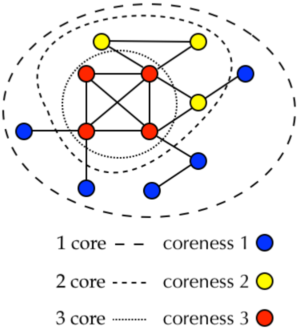
Definitions

K-Core

A *k*-core of a graph G is an undirected subgraph in which all vertices have degree at least *k*.

Coreness

The coreness of a vertex is *k* *iff* he belongs to a *k*-core but not to any *k+1*-core.



Experiments conditions

Selection of `UNSAT` problems

- 60 problems from the 2012-2017 `SAT` competitions;
- Select at least **two** benchmarks per family of problems;
- Need less than one million conflicts to be solved on the original formula.

Conditions

- Cluster of Xeon E7-4870 processors from the *Mesocentre Aquitain de Calcul Intensif*;

Characterization of K-Cores

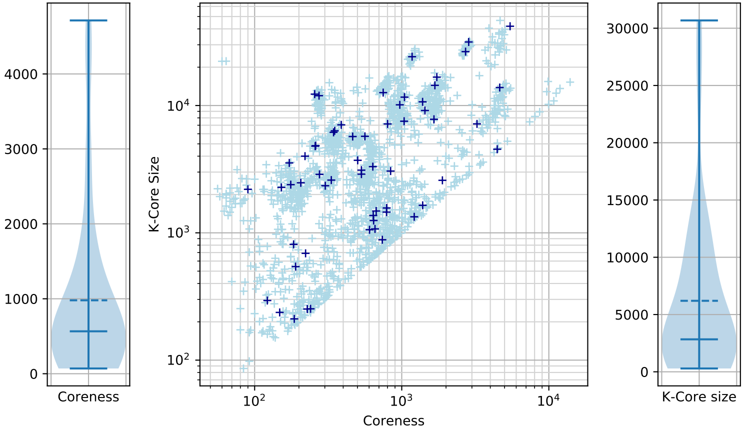


Figure 3: On **left** and **right**, we show the distribution of median values over shuffled instances. In the **middle**, blue darker plots are original problems, lighter shuffled problems.

Characterization of K-Cores

We call **useful** a clause necessary for the proof, and **useless** otherwise.

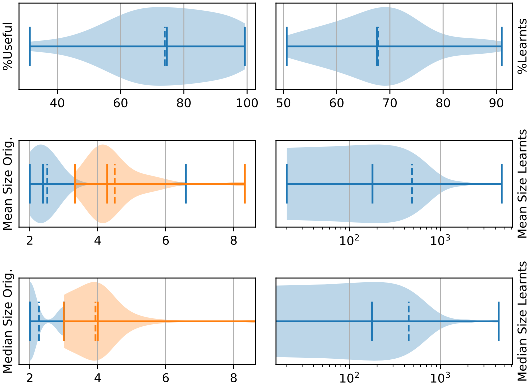
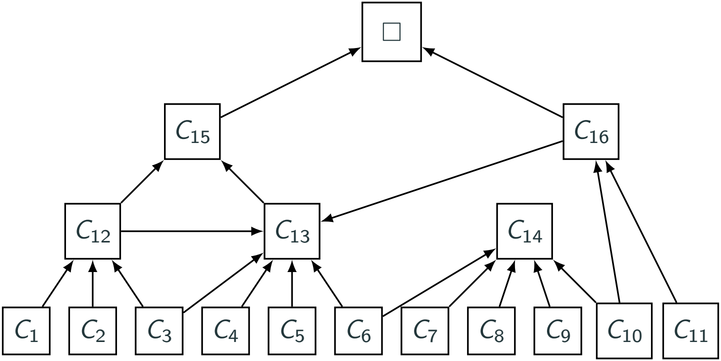
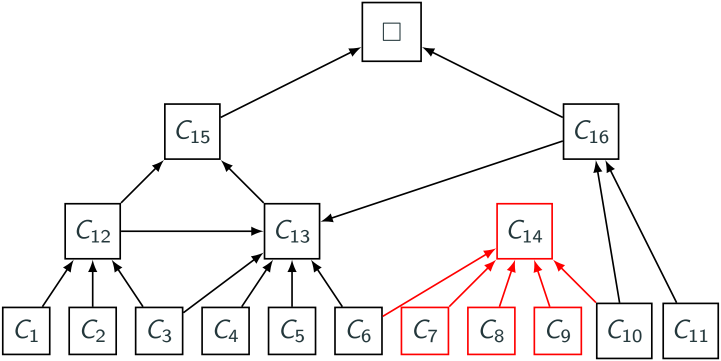


Figure 4: Violin plots summarizing some information about **K-Cores**. All numbers are median values over 50 shuffled and original problems over our set of 60 problems.

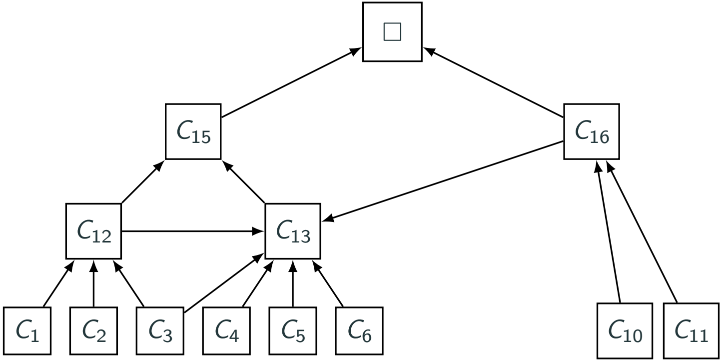
Exemple of useless clauses



Exemple of useless clauses



Exemple of useless clauses



Characterization of K-Cores

We call **useful** a clause necessary for the proof, and **useless** otherwise.

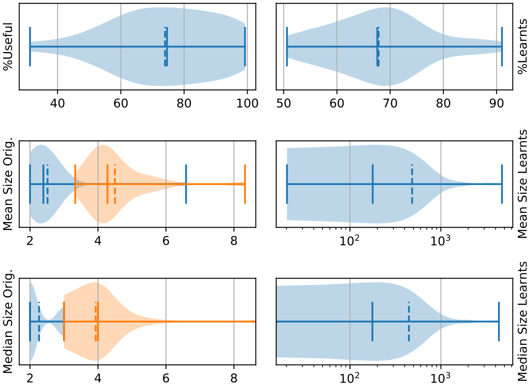


Figure 5: Violin plots summarizing some information about **K-Cores**. All numbers are median values over 50 shuffled and original problems over our set of 60 problems.

Characterization of K-Cores

Summary

- K-Cores can be very large (median larger than **2000**);
- Surprisingly not entirely composed of useful clauses;
- Original K-Core clauses are brief (median of the clauses are binary or ternary clauses);
- However, large disparity in the size of K-Core learnt clauses.

On predictions based on Dependency Graph analysis

On predictions based on Dependency Graph analysis

Objectives

- Identify during the analysis which clauses will be useful;
- Guess which variables will occur in the last learnt clauses, just before deriving the final contradiction.

Conditions

In both, we report the analysis of the **DG** (Dependency Graph) after **20,000 conflicts** and at **half-run**, by simply removing from the **DG** all the nodes and edges added after the limit.

A simple flow algorithm

The idea is to measure the clauses that occur in the **maximal** number of paths to a root node (clause without descendant).

Predicting useful clauses

Predicting useful clauses

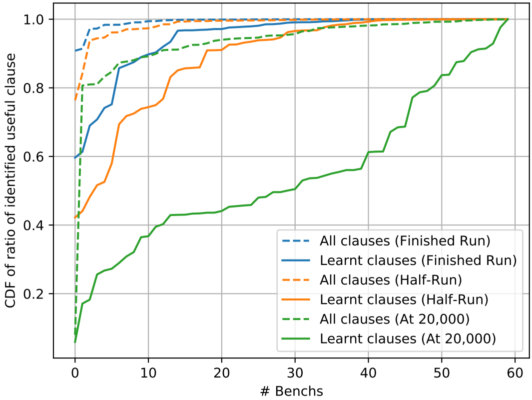


Figure 6: All clauses include 10,000 clauses that can be original or learnt. Learnt clauses restrict the computation to only learnt clauses. A **prediction** is made after 20,000 conflicts, at half-run, and at the end of the run.

Predicting useful clauses

Some results...

- A simple **flow** algorithm can already predict usefulness with a good precision;

Predicting useful clauses

Some results...

- A simple **flow** algorithm can already predict usefulness with a good precision;
- We can identify useful original clauses at the very beginning of the computation;

Predicting useful clauses

Some results...

- A simple **flow** algorithm can already predict usefulness with a good precision;
- We can identify useful original clauses at the very beginning of the computation;
- At half of the computation, we correctly guess at least ***90%*** of useful learnt clauses for **half of the problems**;

Predicting useful clauses

Some results...

- A simple **flow** algorithm can already predict usefulness with a good precision;
- We can identify useful original clauses at the very beginning of the computation;
- At half of the computation, we correctly guess at least ***90%*** of useful learnt clauses for **half of the problems**;

... But no improvement yet

Despite the importance of being able to detect a **useful** clause early, it did not allowed us to improve `Glucose` **yet**.

Detecting future learnt clauses

Detecting future learnt clauses

Detection of variables in the last learnt clause

- We want to guess which **variables** will occur in the last learnt clauses;
- We studied the variables occurring in the K-Core at **half of the computation**;
- We only consider **the most frequent variables** in the K-Core.

Detecting future learnt clauses

Results of our prediction for literals occurring over 60 problems

Table - Top-Y variables (*rows*) w.r.t the last X learnt clauses (*columns*)

	20	**50**	**100**	**1000**
5	27	37	45	53
10	42	47	50	54
20	49	51	51	55

Detecting future learnt clauses

Results of our prediction for literals occurring over 60 problems

Table - Top-Y variables (*rows*) w.r.t the last X learnt clauses (*columns*)

	20	**50**	**100**	**1000**
5	**27**	37	45	53
10	42	47	50	54
20	49	51	51	55

Results of our prediction for literals occurring over 60 problems

Table - Top-Y variables (*rows*) w.r.t the last X learnt clauses (*columns*)

	20	**50**	**100**	**1000**
5	27	37	45	53
10	42	47	50	54
20	**49**	51	51	55

Conclusion

Conclusion

- We have highlighted the existence of a very dense subgraph in the proofs, the K-Core;
- We are capable of identifying a set of useful learnt clauses at **half** of the run;
- We can also identify a very small set of variables that will occur in the very last learnt clauses;

Further work

- We will try to improve `SAT` solvers thanks to the detection of useful clauses early in computation;
- Take into account the existence of K-Core even for the parallelization;

Thank you

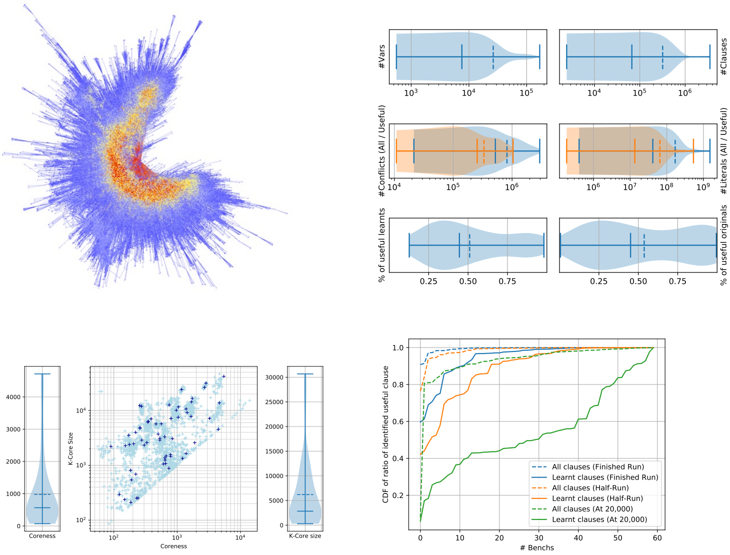


Figure 7: Overview of our results

Analysis of Parallel Proofs

Analysis of Parallel Proofs

- How does the proof evolve *w.r.t* the number of threads ?
- We simulated a Round Robin parallel `SAT` solver, based on `Glucose`

Modern parallel SAT solver

Φ: Set of initials clauses

Σ: Set of learnts clauses

Algorithm 2 Modern parallel SAT solver

While $\square \notin \Phi \cup \Sigma$ **do**

  $C = \mathit{learntClauses}()$

  $\Sigma = \Sigma \cup C$

  $\mathit{Export}(C)$

  $\Sigma = \Sigma \cup \mathit{importClause}()$

  If $\mathit{overfull}(\Sigma)$ **Then**

    $\Delta = \mathit{clausestoDelete}(\Sigma)$

    $\Sigma = \Sigma \backslash \Delta$

  End If

End While

Analysis of Parallel Proofs

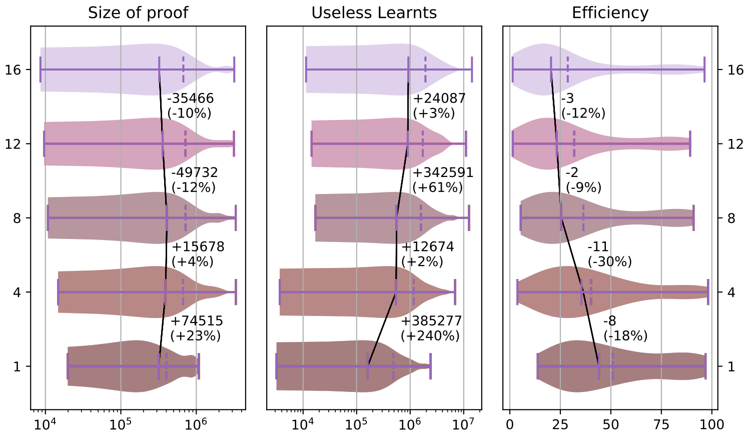


Figure 8: The evolution of metrics on the proofs according to the number of solvers

Analysis of Parallel Proofs

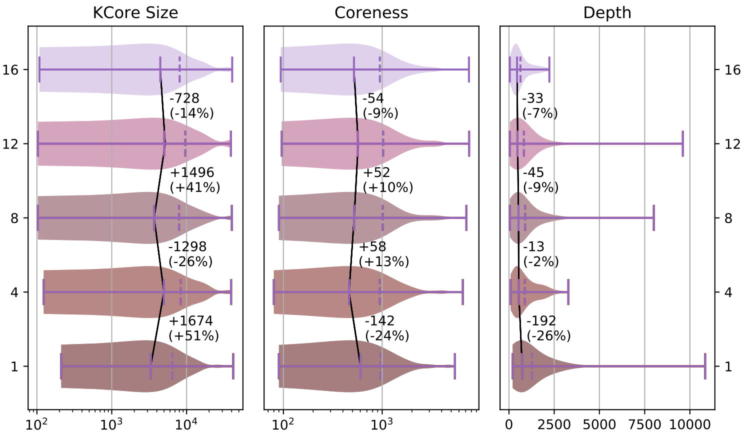


Figure 9: The evolution of metrics on the K-Cores according to the number of solvers

Analysis of Parallel Proofs

Summary

- Parallelization does not seem to have a big impact on **K-Core**;
- It does not seem to have a big impact of parallelization on the K-Core;