International Workshop
Pragmatics of SAT (POS'25)

Revisiting Clause Vivification

Florian Pollitt[1] Mathias Fleury[1]
Armin Biere[1] Karem Sakallah[3]
Marijn Heule[2] Jiawei Chen[3]
Yonathan Fisseha[3]

[1] University of Freiburg
[2] Carnegie Mellon University
[3] University of Michigan

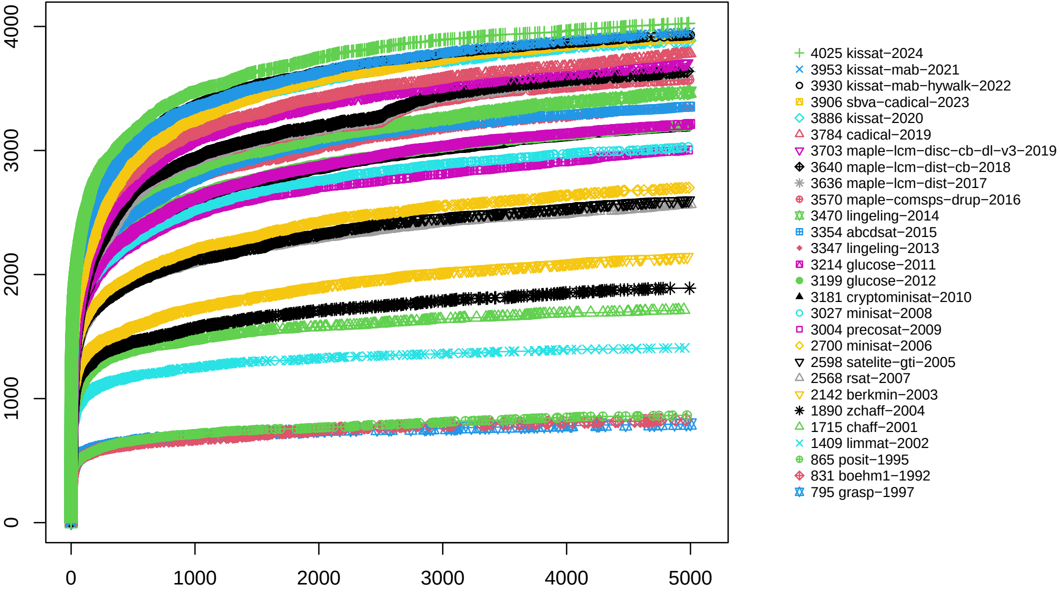
Glasgow, Scotland
August 11, 2025

Outline

- Motivation: museum, regression, inprocessing
- Benchmarks: objectives, factoring benchmarks
- Vivification: idea, history, vivification 4.0
- Experiments: run-time plots, details

Motivation

Legacy Solvers on SAT Competition Anniversary Benchmarks



How do SAT solvers work?

- Ongoing debate among the senior authors of this paper:

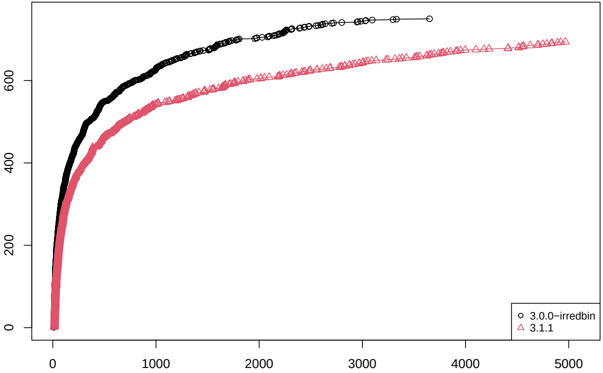
    1. What happened since learning was added to SAT solving?
    2. Can we understand why state-of-the-art solvers are getting faster and faster?
    3. Are there untravelled paths which lead to improve them further?

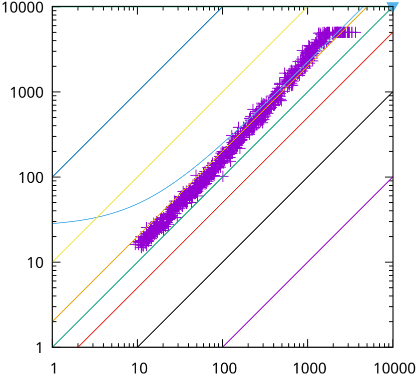
- Some joint understanding evolved about:

  learning, *unlearning*, *branching*, *restarts*, *preprocessing*, *inprocessing*, *portfolio*, . . .

- But what method / benchmarks should be used to investigate that?

Kissat 3.0.0 vs. 3.1.1 Performance Regression on Factoring Benchmarks





accidentally run the “wrong” version 3.0.0-irredbin

(in Nov. 2023 while working on factoring benchmarks)

3.1.1 *y*-axis, i.e., dots *above* diagonal mean 3.0.0-irredbin is faster

Inprocessing and Vivification 4.0

- 20 years ago **preprocessing** gave a huge boost
- 15 years ago **inprocessing** turbo-charged it (as in the last 5 years again)
  - inprocessing allows to preempt preprocessing (pre-solving) and resume it later
  - since 10 years we have proofs for this combination of search and simplification
- evaluated these trade-offs with SAT solvers Satch and TabularaSAT
  - non-satisfactory results as they are far behind Kissat
  - so we are back to to evaluating Kissat with things switched off/on
- focus in this paper on **vivification** in its inprocessing version
- here we discuss newest version (vivification 4.0) in Kissat and CaDiCaL
- performance regression actually due to a subtle change in vivification

Benchmarks

Benchmark Objectives

- similar application (family)
- ideally of real practical value
- different sizes and hardness
- scalable: hardness (solving time) increases with size / parameter
- can generate many of them (not just *ph10*, *ph11*, *ph12*, *ph13*, . . .)
- still in reach of current (CDCL) solvers
- allows us to study effects of techniques / configurations / regressions

Unsatisfiable Factoring Benchmarks

- generated 750 primes p as bit-vector constants
- 50 for each of the 15 bit-widths $n = 33 \ldots 47$
- “equally spaced” (next prime picked after constant delta $2^n/50$)
- generated simple SMT bit-vector formula $p = x \cdot y$
- assuming $x, y \neq 1$ and no multiplication overflow   but not $x \leq y$
- bit-blasted to CNF with Bitwutzla

Pseudocode Benchmark Generator

```
create-benchmarks (lower-bitwidth, upper-bitwidth, primes-per-bitwidth)
    for current-bitwidth from lower-bitwidth to higher-bitwidth
        low = (1 $\ll$ current-bitwidth)    // “$\ll$” = bit-shifting
        high = (1 $\ll$ (current-bitwidth + 1))
        increment = (high - low) / primes-per-bitwidth    // uniform distribution
        for $k$ from 1 to primes-per-bitwidth
            lower-limit = (1 $\ll$ current-bitwidth) + increment * ($k$ - 1)
            upper-limit = (1 $\ll$ current-bitwidth) + increment * $k$
            prime = find-smallest-prime-between (lower-limit, upper-limit)
            if prime generate-factoring-smt (prime, current-bitwidth)
```

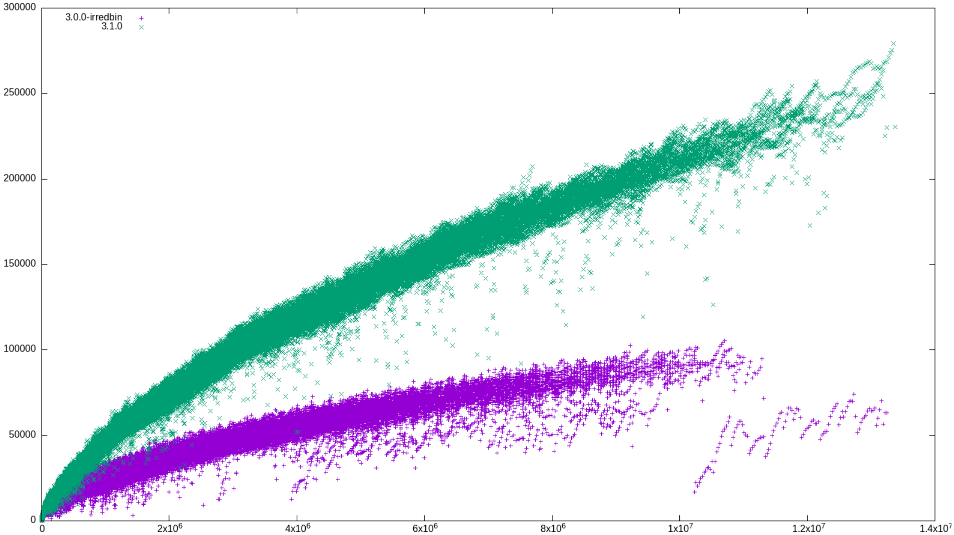
factoring-47-130885865177141.smt2

```
(set-info :smt-lib-version 2.6)
(set-logic QF_BV)
(set-option :produce-models true)
(set-info :status unsat)
(declare-fun a () (_ BitVec 47))
(declare-fun c () (_ BitVec 47))
(declare-fun d () (_ BitVec 47))
(assert (= a (bvmul c d)))
(assert (= a #b11101110000101000111101100000000000000000110101))
(assert (not (= c #b00000000000000000000000000000000000000000000001)))
(assert (not (= d #b00000000000000000000000000000000000000000000001)))
(assert (not (bvumulo c d))) ; ensure no overflow
(check-sat)
(exit)
```

factoring-47-130885865177141.cnf

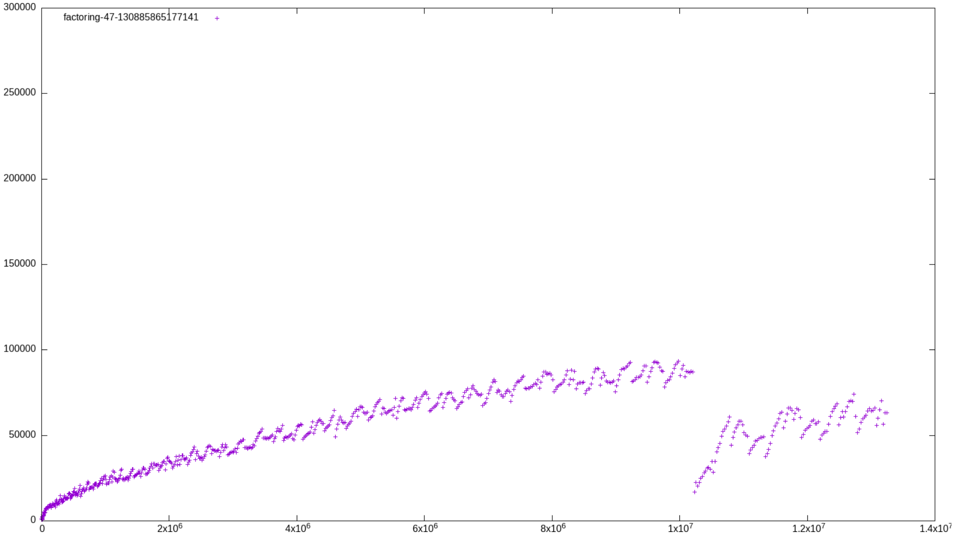
```
c CNF dump 1 start
c Bitwuzla version main-3ea759df11285e722b565c0b5c132dc0bb77066f
p cnf 8926 26635
1 0
47 48 49 0
-49 -47 0
-49 -48 0
46 -49 50 0
-50 -46 0
-50 49 0
45 -50 51 0
-51 -45 0
-51 50 0
...
```

Remaining Learned Clauses after Reductions on Factoring Benchmarks



x-axis number of conflicts where reduction occurred

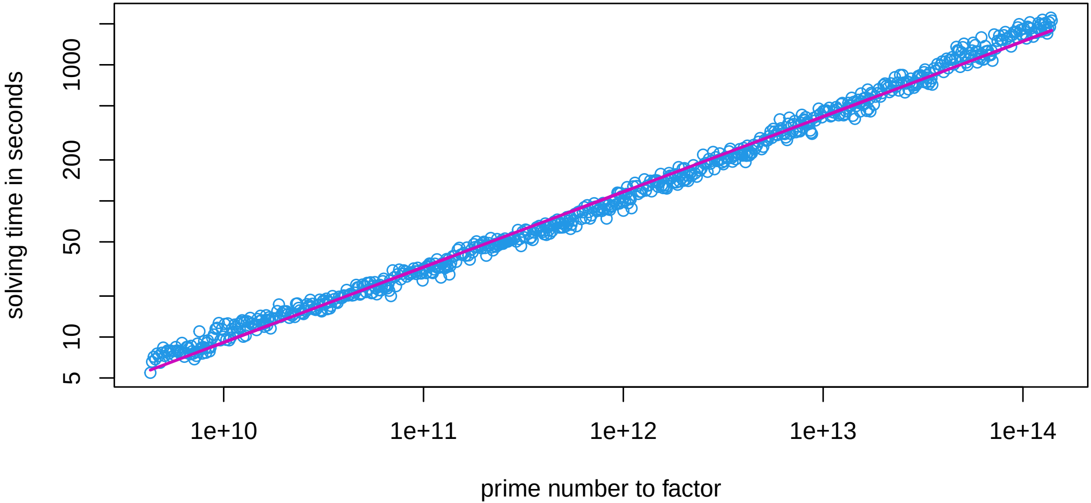
Remaining Learned Clauses for factoring-47-130885865177141



x-axis number of conflicts where reduction occurred

```
$ kissat factoring-47-130885865177141.cnf
...
c - 1657.96 24 14 95 613 315630 1 10106593 87479 54% 11 10415 1815 20%
c - 1662.11 24 14 95 614 315642 1 10131354 87279 54% 11 10415 1815 20%
c - 1666.21 24 14 95 615 315655 1 10156134 86889 54% 11 10415 1815 20%
c - 1670.39 24 13 95 616 315658 1 10180933 87319 54% 11 10415 1815 20%
c - 1674.59 24 13 95 617 315671 1 10205752 87083 54% 10 10415 1815 20%
c s 1676.07 21 13 95 617 315673 1 10214679 95895 54% 10 10278 1815 20%
c e 1676.07 17 13 95 617 315673 1 10214679 95895 54% 10 10148 1768 20%
c
c   seconds   switched      rate                trail     variables
c          MB   reductions      conflicts            glue       remaining
c            level      restarts          redundant   irredundant
c
c s 1676.07 17 13 95 617 315673 1 10214679 95895 54% 10 10064 1768 20%
c e 1676.08 17 13 95 617 315673 1 10214679 95895 54% 10 10057 1765 20%
c - 1677.03 13 14 95 618 315680 1 10230592 17003 54% 11 10057 1765 20%
c - 1678.82 15 14 95 619 315689 1 10255452 22564 55% 11 10057 1765 20%
...
c conflicts:                      13264941             6644.79 per second
c reductions:                          734            18072    interval
...
c maximum-resident-set-size:      30498816 bytes          29 MB
c process-time:                     33m 16s          1996.29 seconds
```

Benchmark Scalability



Vivification

Vivification in a Nutshell

- given CNF F and a candidate clause $\boxed{C = a \vee b \vee c \vee d}$ to be vivified
- assume negations of literals in clause, i.e., $\boxed{\neg a}$, $\boxed{\neg b}$, $\boxed{\neg c}$ and $\boxed{\neg d}$, one by one
- inbetween assumptions propagate them on F ignoring C

  1. on $\boxed{\text{conflict}}$ the candidate clause C is unit implied and can be **removed**
  2. if literal, say d, becomes $\boxed{\text{true}}$ clause C is also unit implied and can be **removed**
  3. if literal, say d, becomes $\boxed{\text{false}}$ during propagation **shrink** C (by removing d)

- first two outcomes: *asymmetric tautology* (AT) or *reverse unit propagated* (RUP)
- goal is to apply on all clauses of CNF until completion (costly)

Vivification/Distillation History

Vivification 1.0

distillation [HanSomenzi-DAC’07] with *trie* to reuse propagations
vivification [PietteHamadiSais-ECAI’08] independently

Vivification 2.0

CaDiCaL 2017 inprocessing version + simulating trie

Vivification 3.0

Maple-LCM-dist-2017 winner SC 2017 [LuoLiXiaoManyàLü-IJCAI’17]
focusing on redundant/learned clauses

Vivification 4.0

this paper new inprocessing version revisited precisely

Scheduling Vivification 4.0

```
vivify(CNF F)    // CNF updated in place / passed by reference
    ticks-budget = search-ticks-since-last-vivification$_{\text{stats}}$ × relative-vivification-effort$_{\text{option}}$
    tier-1-budget = ticks-budget × relative-tier-1-budget$_{\text{option}}$
    tier-2-budget = ticks-budget × relative-tier-2-budget$_{\text{option}}$
    tier-3-budget = ticks-budget × relative-tier-3-budget$_{\text{option}}$
    irredundant-budget = ticks-budget × relative-irredundant-budget$_{\text{option}}$
    remaining-ticks = vivify-tier(F, tier-1 clauses of F, tier1-budget)
    remaining-ticks = vivify-tier(F, tier-2 clauses of F, tier2-budget + remaining-ticks)
    remaining-ticks = vivify-tier(F, tier-3 clauses of F, tier3-budget + remaining-ticks)
    vivify-tier(F, irredundant clauses of F, irredundant-budget + remaining-ticks)
```

Tier Vivification 4.0

```
    vivify-tier (CNF $F$, CNF $G$, ticks-budget)   // update subset of clauses in original CNF in place
 1      limit = $\text{ticks}_{\text{stats}} + \text{ticks-budget}$   // global variable “$\text{ticks}_{\text{stats}}$” updated during propagation
 2      sort literals in clauses $C \in G$ by number of occurrences (more occurrences first)
 3      let $G_1$ be the sub-set of clauses of $G$ which were *not* tried during vivification last time
 4      let $G_2 = G \backslash G_1$   // new clauses or clauses already tried last time
 5      sort $G_1$ and separately $G_2$ lexicographically *w.r.t.* literal occurrences (more first)
        // decision level set to zero at this point
 6      **for all** clauses $C$ in the sequence $G_1$, $G_2$ sorted as in line 5 as long $\text{ticks}_{\text{stats}} < \text{limit}$
 7          **if** *vivify-clause* ($F$, $C$) **then** increment $\text{vivified}_{\text{stats}}$
 8      backtrack to decision level zero
 9      **if** $\text{ticks}_{\text{stats}} > \text{limit}$ **return** 0   // incomplete – remember untried clauses
10      **return** $\text{limit} - \text{ticks}_{\text{stats}}$   // return unused ticks budget – no untried clauses remembered
```

Clause Vivification 4.0 Part 1

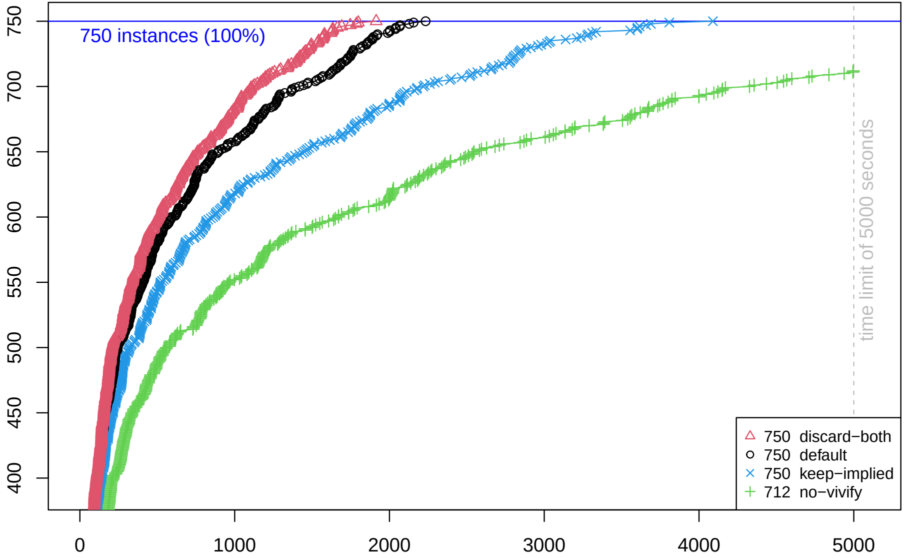
```
vivify-clause (CNF $F$, clause $C$)    // update $F$ and $C$ in place
  mark $C$ as having been tried    // puts it in $G_2$ next time
  let $C = \ell_1 \vee \cdots \vee \ell_n$ sorted by number of occurrences (more occurrences first)
  find maximal $m$ such that $\ell_i$ is assigned to false at decision level $i$ for all $i < m$  // reuse trail
  if $m > 0$ and decision level larger than $m - 1$ backtrack to decision level $m - 1$
  add $m - 1$ to both probes$_{\text{stats}}$ and reused$_{\text{stats}}$    // reused $m$ decisions / probes
  literal implied = $\bot$,  clause conflict = $\bot$    // initialize both to be undefined denoted as "$\bot$"
  for $i = m \ldots n$ as long conflict $= \bot$    // and implied $= \bot$
    if $\ell_i$ is assigned to false continue
    if $\ell_i$ is assigned to true then implied = $\ell_i$ and break
    increase decision level and assign $\ell_i$ to false,   increment probes$_{\text{stats}}$
    // temporarily disable propagation over $C$, i.e., $C$ is simply skipped during propagation
    conflict = propagate ($F$, $C$)    // update global assignment and ticks$_{\text{stats}}$
  // now we have either implied $\neq \bot$, conflict $\neq \bot$, or $C$ is falsified by the current assignment
  (subsuming, learned, irredundant) = vivify-analyze ($C$, conflict, implied)
⋮
```

Clause Vivification 4.0 Part 2

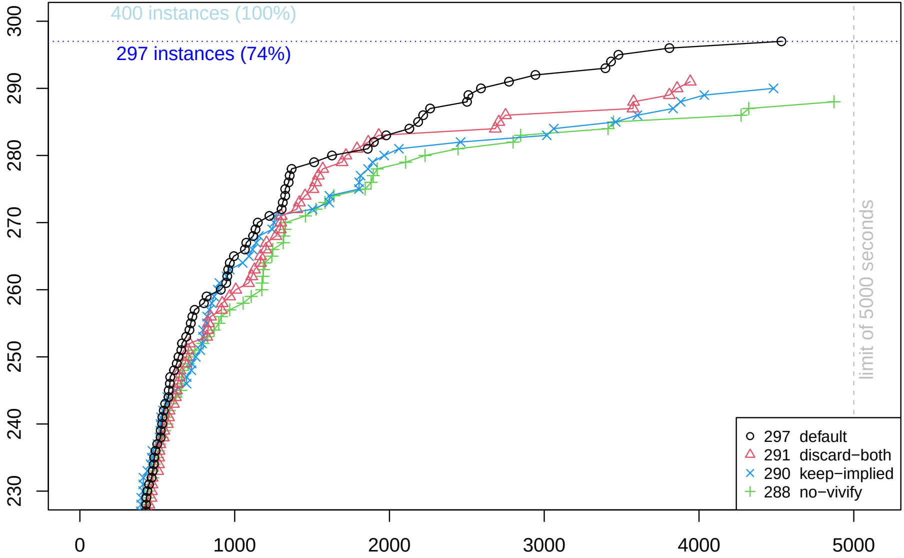
```
⋮
(subsuming, learned, irredundant) = vivify-analyze (C, conflict, implied)
if subsuming $\neq \bot$
   remove $C$ from $F$,  increment subsumed$_{\text{stats}}$  and  $\boxed{\textbf{return } \textit{true}}^{1}$
   // . . . and need to make “subsuming” irredundant if it was redundant but $C$ not
if $|\text{learned}| < |C|$    // actually “learned $\subset C$” as it is a decision learned clause
   replace $C$ in $F$ by learned,  increment shrunken$_{\text{stats}}$  and  $\boxed{\textbf{return } \textit{true}}^{2}$
if implied $\neq \bot$ and $C$ redundant
   // regression version “without-implied” would only return false but the “default” version has:
   remove $C$ from $F$,  increment implied$_{\text{stats}}$  and  $\boxed{\textbf{return } \textit{true}}^{3}$
conflicting  =  conflict $\neq \bot \vee$ implied $\neq \bot$
if conflicting and $C$ irredundant as well as analysis resolved only irredundant clauses
   remove $C$ from $F$,  increment asymmetric$_{\text{stats}}$  and  $\boxed{\textbf{return } \textit{true}}^{4}$
if implied $\neq \bot$ and vivify-instantiate $(F, C, \ell_n)$    // $C$ falsified at decision level $n$
   remove $\ell_n$ from $C$,  increment instantiated$_{\text{stats}}$  and  $\boxed{\textbf{return } \textit{true}}^{5}$
return false
```

Experiments

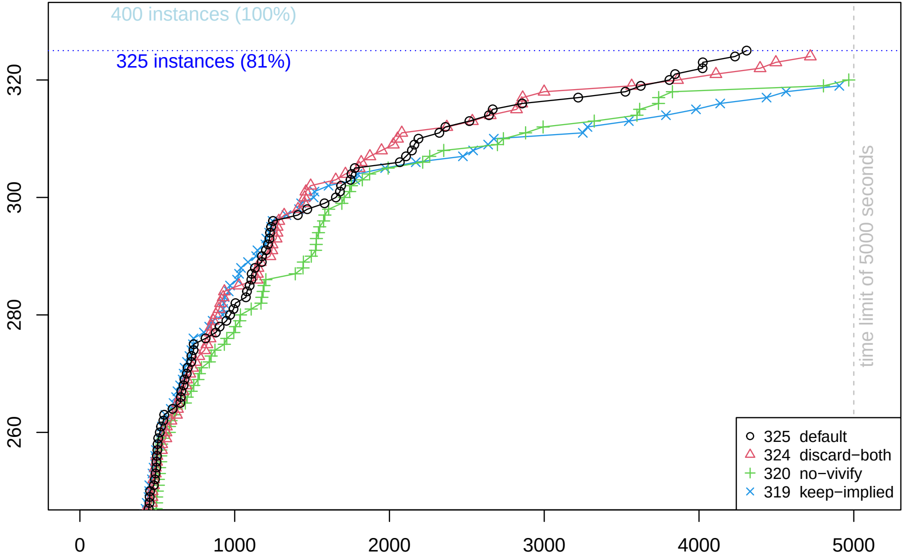
Factoring Benchmarks



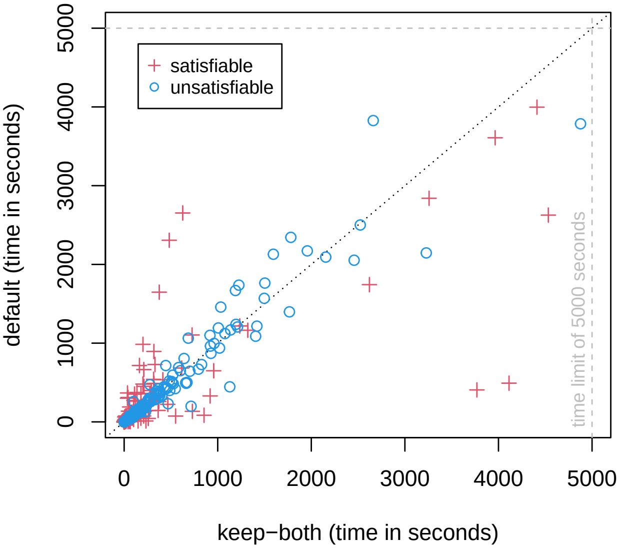
SAT Competition 2023



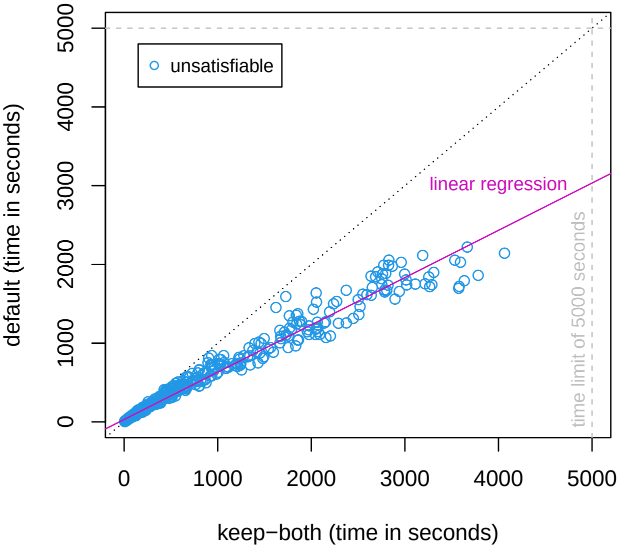
SAT Competition 2024



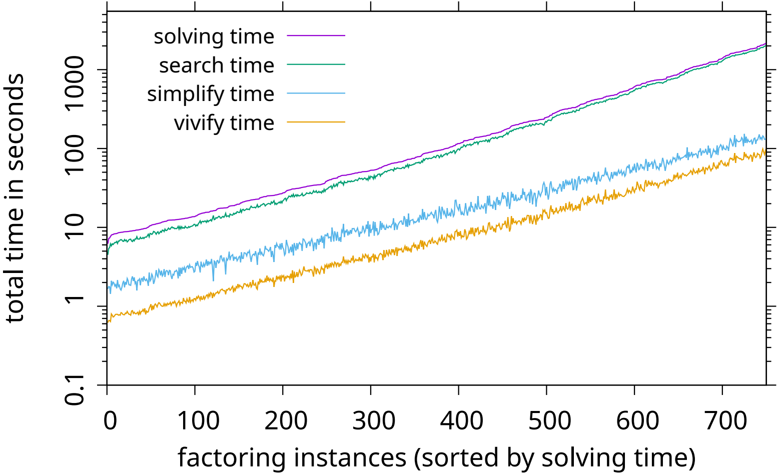
SAT Competition 2024



Factoring Benchmarks



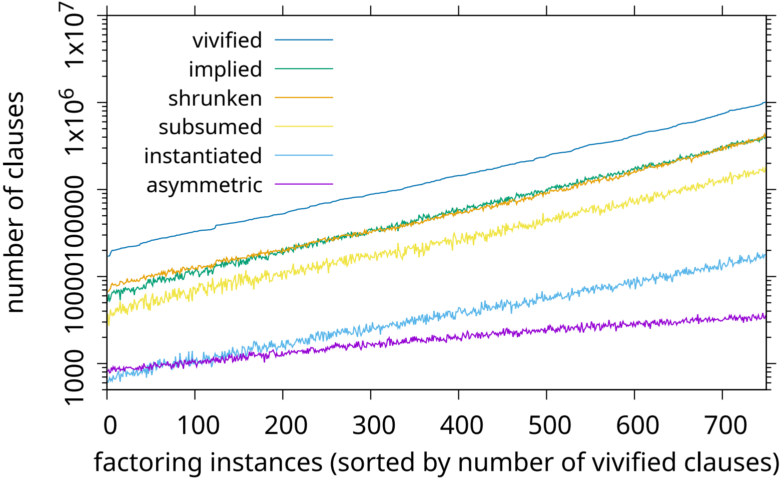
Factoring Benchmarks Times



Clause Vivification 4.0 Part 2

```
⋮
(subsuming, learned, irredundant) = vivify-analyze ($C$, conflict, implied)
if subsuming $\neq \bot$
  remove $C$ from $F$,  increment $\text{subsumed}_{\text{stats}}$  and  $\boxed{\textbf{return}\ \textit{true}}$[1]
  // . . . and need to make “subsuming” irredundant if it was redundant but $C$ not
if $|\text{learned}| < |C|$    // actually “learned $\subset C$” as it is a decision learned clause
  replace $C$ in $F$ by learned,  increment $\text{shrunken}_{\text{stats}}$  and  $\boxed{\textbf{return}\ \textit{true}}$[2]
if implied $\neq \bot$ and $C$ redundant
  // regression version “without-implied” would only return false but the “default” version has:
  remove $C$ from $F$,  increment $\text{implied}_{\text{stats}}$  and  $\boxed{\textbf{return}\ \textit{true}}$[3]
conflicting = conflict $\neq \bot \vee$ implied $\neq \bot$
if conflicting and $C$ irredundant as well as analysis resolved only irredundant clauses
  remove $C$ from $F$,  increment $\text{asymmetric}_{\text{stats}}$  and  $\boxed{\textbf{return}\ \textit{true}}$[4]
if implied $\neq \bot$ and vivify-instantiate ($F$, $C$, $\ell_n$)    // $C$ falsified at decision level $n$
  remove $\ell_n$ from $C$,  increment $\text{instantiated}_{\text{stats}}$  and  $\boxed{\textbf{return}\ \textit{true}}$[5]
return false
```

Factoring Benchmarks Vivified Clauses



Summary

Conclusion

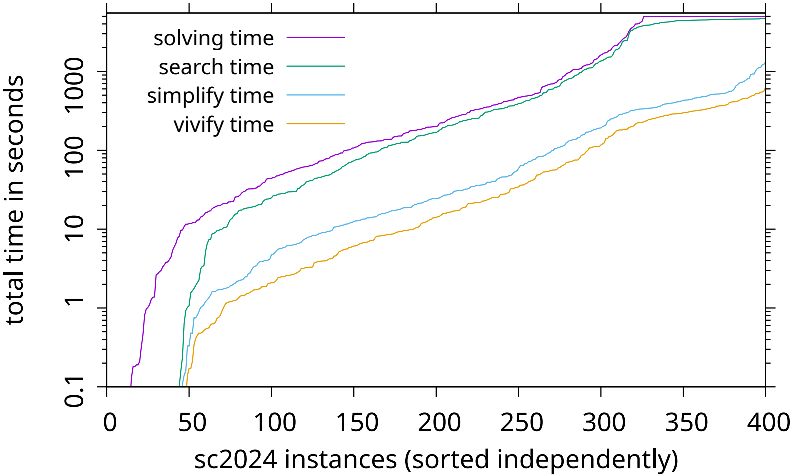
- need new benchmarks to understand why solvers get faster and faster
- found a simple *scalable* benchmark set
- triggered an interesting regression
- Vivification 4.0

Future Work

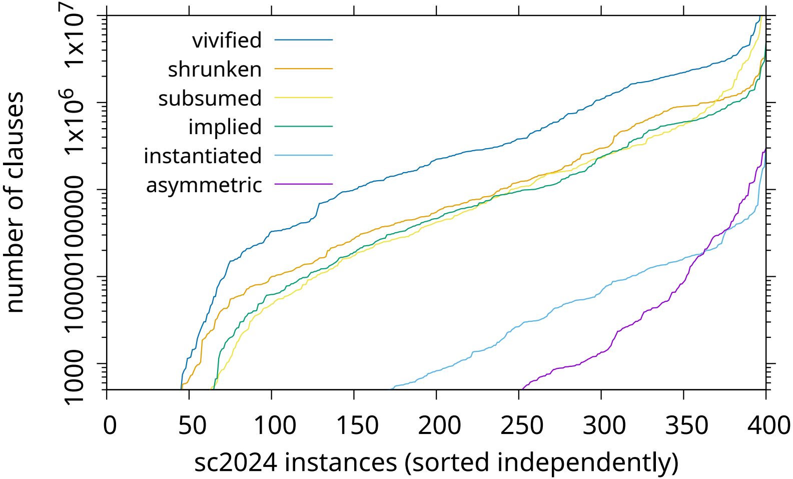
- more practical scalable benchmarks
- scalable satisfiable benchmarks

Supported by Baden-Württemberg bwHPC, DFG grant INST 35/1597-1 FUGG and gift from Intel.

SAT Competition 2024 Benchmarks Times



SAT Competition 2024 Benchmarks Vivified Clauses


vivified
shrunken
subsumed
implied
instantiated
asymmetric
number of clauses
1x10^7
1x10^6
sc2024 instances (sorted independently)