Clausal Equivalence Sweeping

Armin Biere[1] Katalin Fazekas[2] Mathias Fleury[1] Nils Froleyks[3]

[1] universität freiburg [2] TU WIEN [3] JKU

October 18, 2024, Prague, Czech Republic

24th International Conference on
Formal Methods in Computer-Aided Design

FMCAD 2024

https://cca.informatik.uni-freiburg.de/biere/talks/Biere-FMCAD24-talk.pdf

supported by FWF DFG Intel

Home
Venue
Slides
Panels
Pictures
Registration
Final Program
Invited Speakers
Accepted Papers
Call For Papers
Organization
Submission
Contact Info
FMCAD.org
Sponsors

Semiconductor Panel

Frontline users speak up!
What works, What doesn't, and What are they doing about it?

Ken Albin, AMD
Alan Carlin, Freescale
Velu Durairaj, TI
Alan Hunter, ARM
Tushar Ringe, Analog Devices
Dan Smith, NVIDIA

Moderator: Adnan Aziz, University of Texas at Austin

EDA Vendors Lunch Panel

What will be the next breakthrough solutions in formal?

Harry Foster, Mentor
Ziyad Hanna, Jasper
Kevin Harrer, Synopsys
Axel Scherer, Cadence

Moderator: JL Gray, Verilab

Hardware Equivalence Checking (EC)

- compare original (golden) with variant circuit (optimized) aka. *miter* [Brand'93]
- widespread industrial use since 20+ years

  EDA provides EC tools: Siemens (Mentor), Cadence, Synopsys, ...
- combinational EC (focus here) and sequential EC (model checking)
- original seminal work used **BDD sweeping**

  Kuehlmann and Krohm "Equivalence checking using cuts and heaps" DAC'97
- mostly replaced by **SAT sweeping** (e.g., Fully Reduced AIGs = FRAIGs)

  Kuehlmann, Paruthi, Krohm, Ganai
  "Robust boolean reasoning for equivalence checking and functional property verification" TCAD'02

  Mishchenko, Chatterjee, Jiang, Brayton
  "FRAIGs: A unifying representation for logic synthesis and verification" ERL TR 2005
- state-of-the-art uses **hybrid** solving (clausal + circuit)

  Zhang, Jiang, Amarù, Mishchenko, Brayton
  "Deep integration of circuit simulator and SAT solver" DAC'21

  Zhang, Jiang, Mishchenko, Amarù "Improved large-scale SAT sweeping" IWLS'22

Can we do this directly on CNF?

- clausal equivalence sweeping
  - just encode miter (comparison of both circuits) into CNF via. Tseitin
  - disadvantage: circuit structure lost (gates, models, topological order)
  - advantage: easily combined with powerful SAT techniques (inprocessing etc.)
- submitted in 2013 CNF miters to the SAT competition
  - CDCL works really badly on these (restarts essential)
  - even fail on “isomorphic miters” (copies of identical circuits)
  - structural hashing (strash) aka. hash-consing alone solves them instantly *but* only on the original circuit representation
- hyper-binary resolution can “strash” for AIGs in CNF (still does not scale)

  Heule, Järvisalo, Biere “Revisiting hyper binary resolution” CPAIOR’13
- new recent SAT’24 algorithm finally successfully does “strash” CNF

  Biere, Fazekas, Fleury, Froleyks “Clausal Congruence Closure” SAT’24
- this FMCAD’24 paper is about how to do the rest, i.e., semantic CNF sweeping

buddy@company.com, Jan 16, 2023, 5:14 PM

to me, colleague@company.com

Hi Armin,

My colleague (in CC) has encountered an unsatisfiable benchmark formula from the 2014 SAT competition that is solved immediately by lingeling (including a verified proof) but takes much longer by other solvers like CaDiCaL, kissat, or even Gimsatul (the formula is attached to this email if you are interested).

It turns out that lingeling solves the formula during failed-literal probing. This is interesting because CaDiCaL and kissat perform failed-literal probing too, but they must be doing it differently. Even if I explicitly tell CaDiCaL to perform one or more rounds of preprocessing (with the -P command-line option), it still takes long to solve.

We do not want you to spend any time investigating this, but we wanted to hear whether you can think of an obvious explanation for why this is happening? Is it maybe because lingeling is using a different heuristic for choosing the literals to probe on? Or because of other heuristics related to probing? Or is it maybe something completely different?

Buddy

to buddy@company.com, colleague@company.com, Jan 16, 2023, 5:20 PM

Very cool, thanks. I will have a look! Maybe it is 'simple probing', where we had started experiments with Norbert Manthey once but it never gave a paper. This simulates structural hashing on AIGs on the CNF level (fast - because other methods do that too but more and slower).

Armin

to buddy@company.com, colleague@company.com, Jan 16, 2023, 5:24 PM

Yep, so it is probably actually a benchmark I submitted in that year ;-) Those are miters of identical circuits, which can be trivially solved if you have the AIGs: just read the input. For SAT it is much harder even though we know there is a simple resolution proof. See our CPAIOR'13 paper (Knuth called this issue a dead body in the cellar). I have not found a way to make this fast in all cases and worse it can not be preempted as variable elimination destroys the nice structure for this simple probing to work. The SAT sweeper in Kissat can do it with Kitten as sub-solver, but you have to give more time.

With '--no-prbsimple' you can check that it is indeed 'simple probing' to make Lingeling fast on this one.

Armin

to buddy@company.com, colleague@company.com, Jan 16, 2023, 5:30 PM

BTW, I guess you used this one

/data/cnf/sc2022/main/6s184.cnf.xz

which is a benchmark I regularly use for testing now ....
(Kissat solves it in 800 seconds or so).

It is good that the organizer's procedure seems to pick up those trivial benchmarks ;-)

Armin

buddy@company.com, Jan 16, 2023, 5:30 PM

to me, colleague@company.com

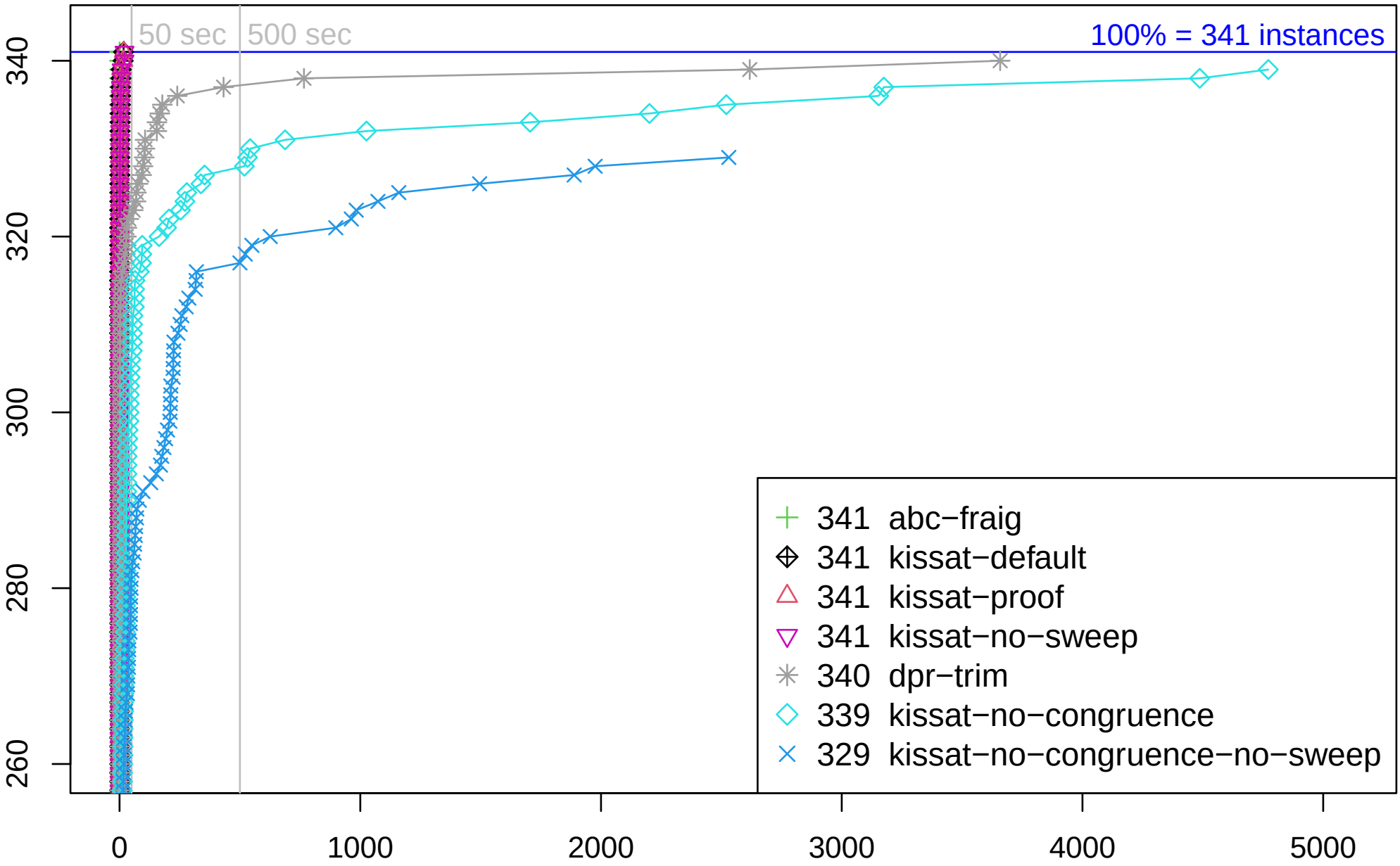
Haha, so I guess lingeling was the only solver solving that formula efficiently back then. :-D

Thanks a lot for responding so quickly! I just started a run of lingeling with '--no-prbsimple', and after more than two minutes it is still running. Nice!

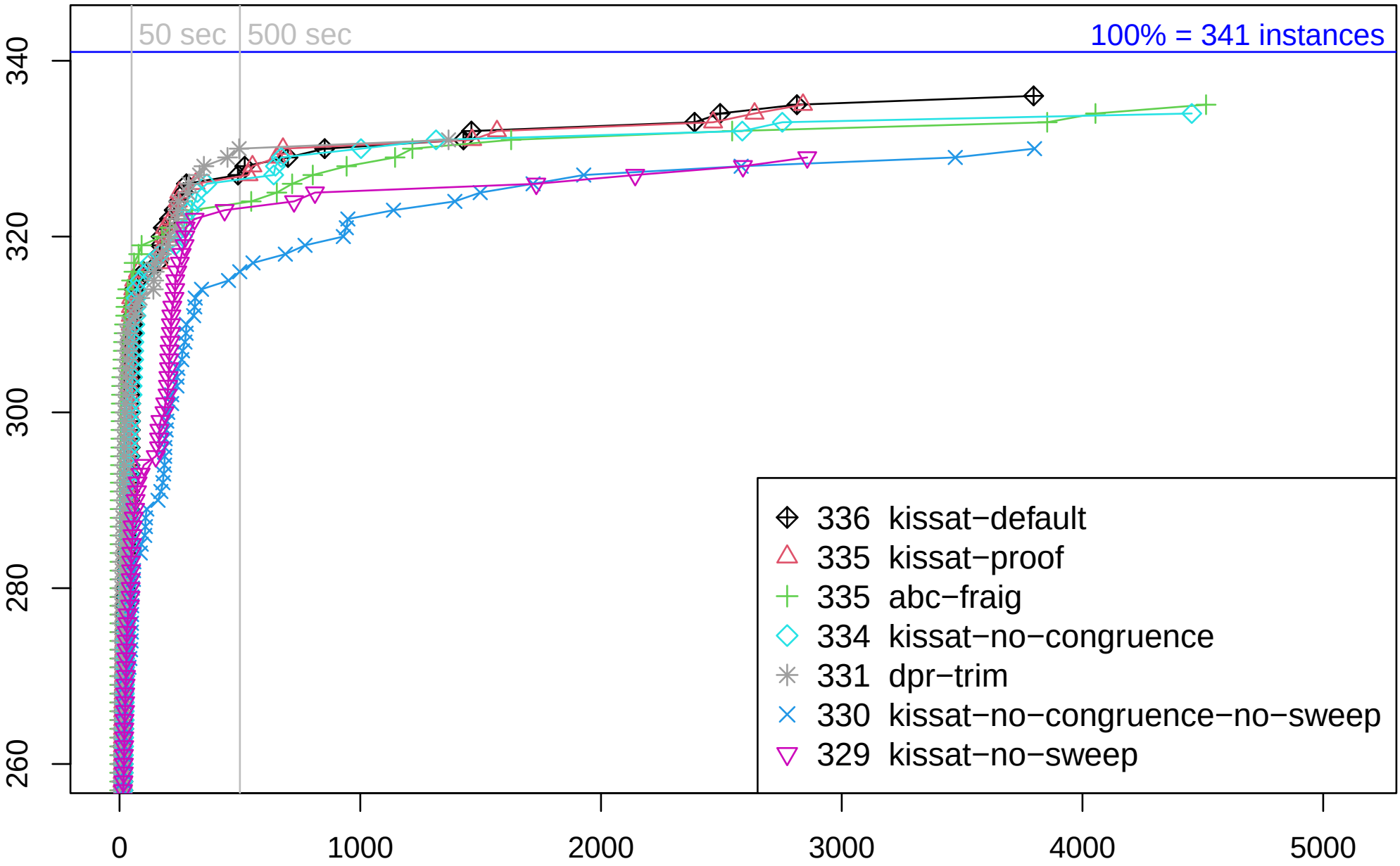
Thanks a lot,
Buddy

```
$ date
Tue Oct 15 17:51:51 CEST 2024

$ lscpu | grep Model.name
Model name:                           AMD Ryzen 9 7950X 16-Core Processor

$ grep p cnf 6s184.cnf
p cnf 33368 97516

$ time -p kissat 6s184.cnf -q  # with SAT'24 (congruence), FMCAD'24 (sweep)
s UNSATISFIABLE
real 0.02
user 0.01
sys 0.01

$ time -p kissat 6s184.cnf -q --no-congruence             # no SAT'24
s UNSATISFIABLE
real 15.96
user 15.86
sys 0.09

$ time -p kissat 6s184.cnf -q --no-congruence --no-sweep  # no FMCAD'24
s UNSATISFIABLE
real 107.04
user 106.59
sys 0.43
```

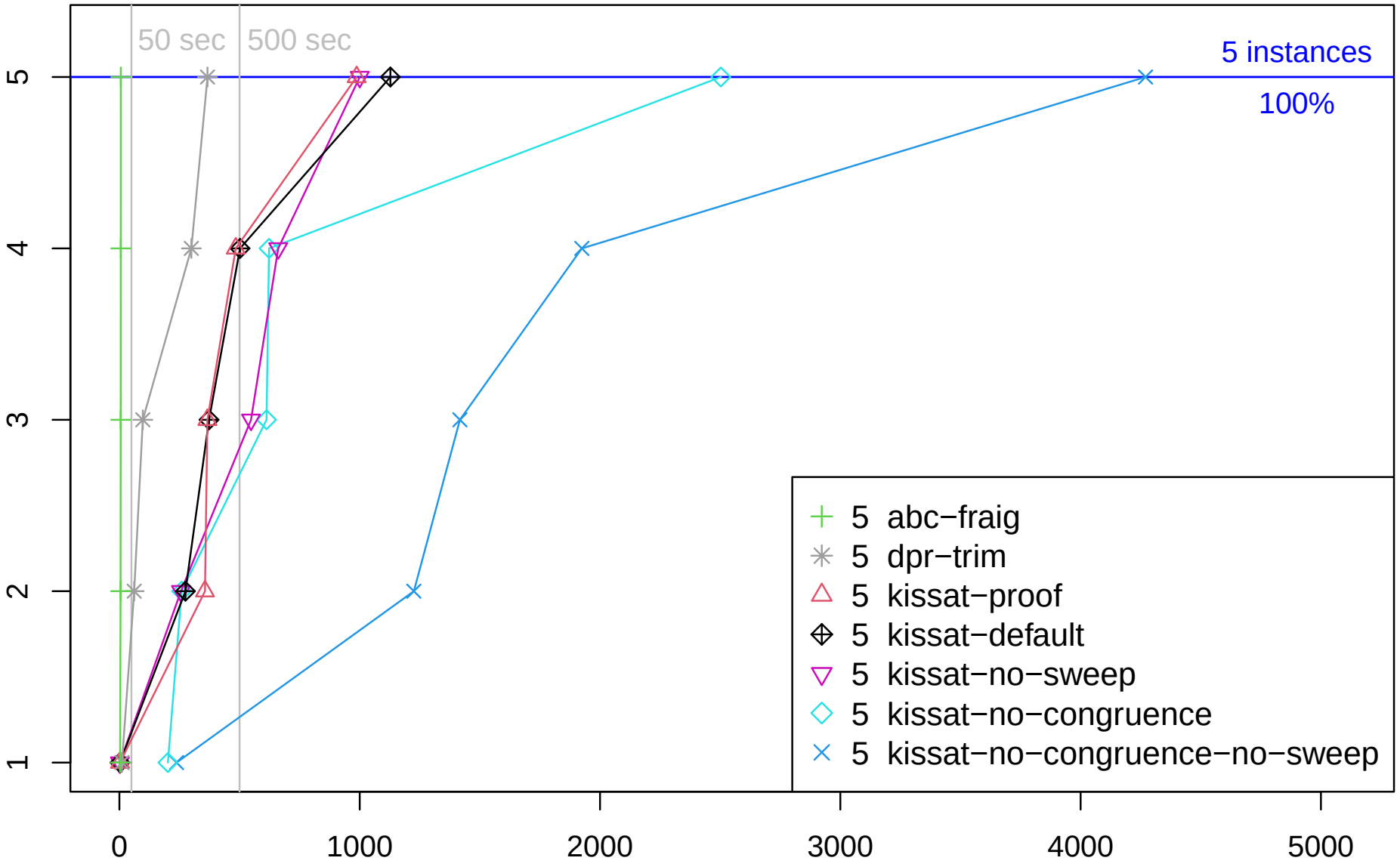
Isomorphic 341 Miters from HWMCC'12 Benchmarks



Optimized 341 Miters from HWMCC'12 Benchmarks



State-of-the-Art Circuit Approach on 5 Hard Miters [IWLS'22,DAC'21]



[IWLS'22] "Improved large-scale SAT sweeping"
He-Teng Zhang, Jie-Hong R. Jiang, Alan Mishchenko, and Luca Amarù.

[DAC'21] "Deep integration of circuit simulator and SAT solver"
Hee-Teng Zhang, Jie-Hong R. Jiang, Luca G. Amarù, Alan Mishchenko, and Robert K. Brayton

How does it work?

clausal-equivalence-sweeping (CNF F)

1 working set W initialized to all variables of F
2 **while** $W \neq \emptyset$ — call to embedded SAT solver Kitten
3 pick and remove x from W
4 determine *bounded* CNF environment G of x with $G \subseteq F$ and $x \in V(G)$
5 **if** G is unsatisfiable [1] then also F is unsatisfiable and **return**
6 backbone candidate set $B \leftarrow \{l \mid \sigma(l) = 1\}$ where σ is a model of G
7 equivalent literal partition $P \leftarrow \{B\}$
8 **while** $B \neq \emptyset$
9 pick and remove l from B
10 **if** $G \wedge \bar{l}$ unsatisfiable [2] add l as unit to F and G, propagate and **continue**
11 refine B and P based on model of $G \wedge \bar{l}$
12 **while** exists $L \in P$ with $l, k \in L$ and $l \neq k$
13 **if** $G \wedge l \wedge \bar{k}$ is unsatisfiable [3] and $G \wedge \bar{l} \wedge k$ is unsatisfiable [4]
14 **then** substitute l by k in F and G and propagate
15 **else** refine P based on satisfying model
16 add back to W all the variables in reduced or substituted clauses

Slide invited talk at FDL'22

```
kitten *kitten_init (void);
void kitten_clear (kitten *);
void kitten_release (kitten *);

void kitten_track_antecedents (kitten *);

void kitten_shuffle_clauses (kitten *);
void kitten_flip_phases (kitten *);
void kitten_randomize_phases (kitten *);

void kitten_assume (kitten *, unsigned lit);

void kitten_clause (kitten *, size_t size, unsigned *);
void kitten_unit (kitten *, unsigned);
void kitten_binary (kitten *, unsigned, unsigned);

void kitten_clause_with_id_and_exception (kitten *, unsigned id,
                                          size_t size, const unsigned *,
                                          unsigned except);

void kitten_no_ticks_limit (kitten *);
void kitten_set_ticks_limit (kitten *, uint64_t);
```

only clear memory aka std::vector.clear

reallocate if release every time

diversify solutions / cores

produce clausal cores in memory

limited solving

```
int kitten_solve (kitten *);
int kitten_status (kitten *);

signed char kitten_value (kitten *, unsigned);
bool kitten_failed (kitten *, unsigned);
bool kitten_flip_literal (kitten *, unsigned);

unsigned kitten_compute_clausal_core (kitten *, uint64_t * learned);
void kitten_shrink_to_clausal_core (kitten *);

void kitten_traverse_core_ids (kitten *, void *state,
                               void (*traverse) (void *state, unsigned id));

void kitten_traverse_core_clauses (kitten *, void *state,
                                   void (*traverse) (void *state,
                                                     bool learned, size_t,
                                                     const unsigned *));
```

new Secret Sauce for sweeping

clausal Cores !

Conclusion

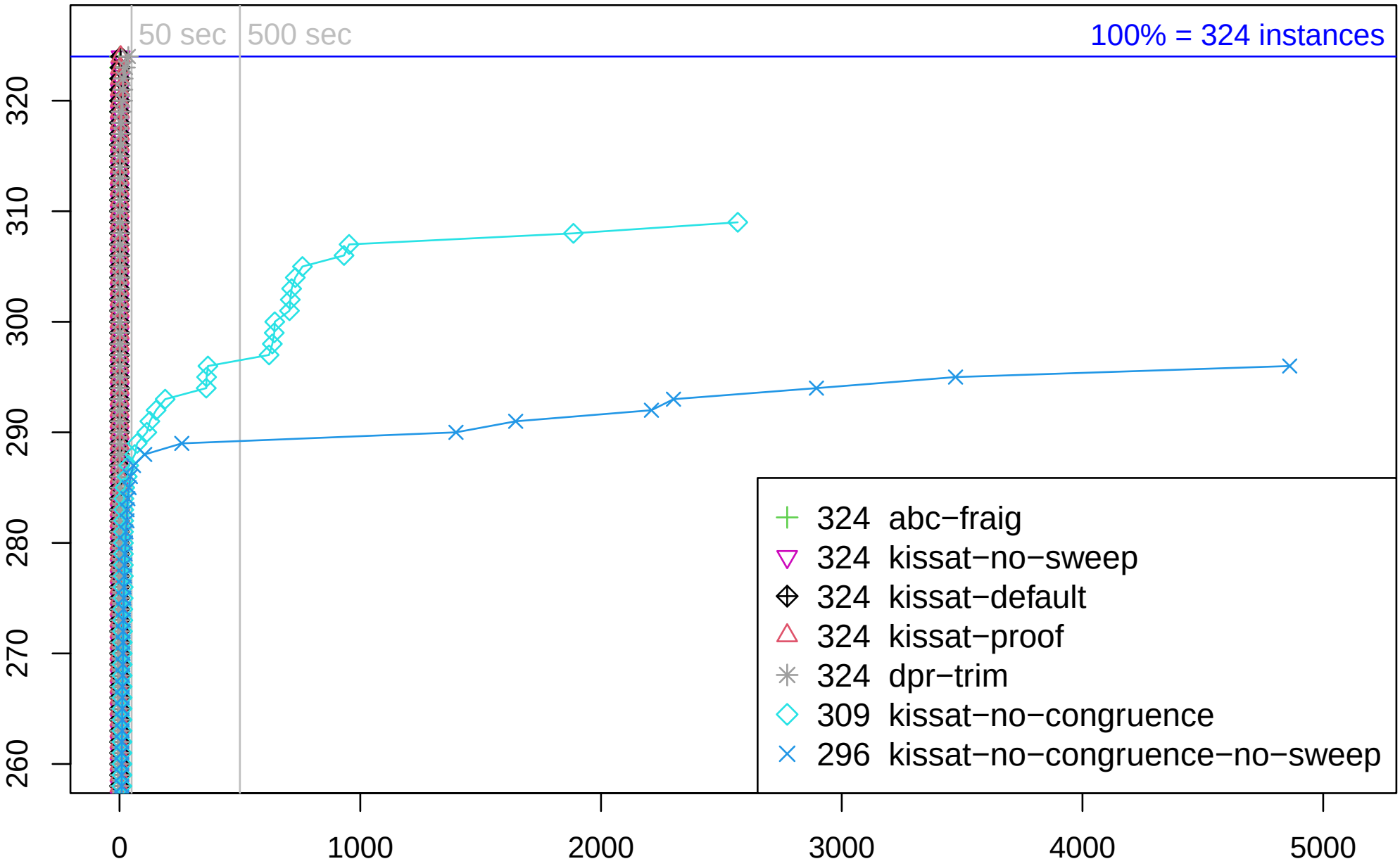
- Semantic SAT Sweeping directly on CNF [FMCAD'24]
- Complements syntactic Clausal Congruence Closure [SAT'24]
- Improves miter checking in state-of-the-art SAT Solver Kissat
- Useful for Optimized / Synthesized miters
- Helps in other SAT solving tasks too

Kissat dominated the SAT Competition 2024

Future Work

- How to maintain **global** Backbone / Equivalence candidates?
- How to get a good initial **global** Backbone / Equivalence candidate set?
- Article on merging SAT'24 and FMCAD'24 work
- Port clausal equivalence sweeping to CaDiCaL
- How to produce linear proofs (LRAT)?

Isomorphic 324 Miters from HWMCC'20 Benchmarks



Optimized 324 Miters from HWMCC'20 Benchmarks