Clausal Congruence Closure

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

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

August 21, 2024, Pune, India

27th International Conference on
Theory and Applications of Satisfiability Testing

SAT 2024

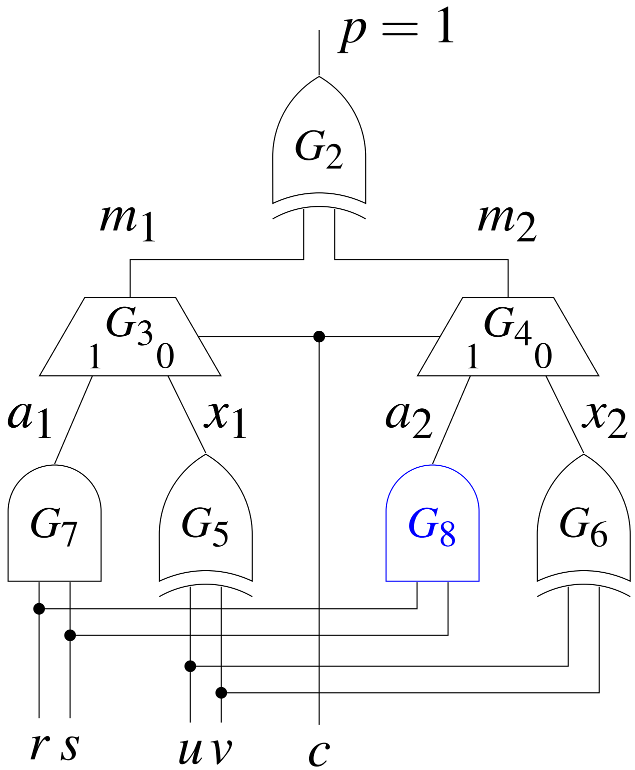
https://cca.informatik.uni-freiburg.de/biere/talks/Biere-SAT24-congruence-talk.pdf

supported by FWF DFG Intel

Circuit Equivalence Checking — Isomorphic Miter

$$p \underset{1}{=} 1$$
$$p \underset{2}{=} m_1 \oplus m_2$$
$$m_1 \underset{3}{=} c \mathbin{?} a_1 : x_1$$
$$m_2 \underset{4}{=} c \mathbin{?} a_2 : x_2$$
$$x_1 \underset{5}{=} u \oplus v$$
$$x_2 \underset{6}{=} u \oplus v$$
$$a_1 \underset{7}{=} r \wedge s$$
$$a_2 \underset{8}{=} r \wedge s$$

(a) gates $G_1, \ldots, G_8$



(b) miter circuit

$$(p)_1$$
$$(\bar{p}\, m_1 m_2)_2\ (\bar{p}\, \bar{m}_1 \bar{m}_2)_3\ (p\, \bar{m}_1 m_2)_4\ (p\, m_1 \bar{m}_2)_5$$
$$(\bar{m}_1 \bar{c}\, a_1)_6\ (\bar{m}_1 c\, x_1)_7\ (m_1 \bar{c}\, \bar{a}_1)_8\ (m_1 c\, \bar{x}_1)_9$$
$$(\bar{m}_2 \bar{c}\, a_2)_{10}\ (\bar{m}_2 c\, x_2)_{11}\ (m_2 \bar{c}\, \bar{a}_2)_{12}\ (m_2 c\, \bar{x}_2)_{13}$$
$$(\bar{x}_1 \bar{u} \bar{v})_{14}\ (\bar{x}_1 u v)_{15}\ (x_1 \bar{u} v)_{16}\ (x_1 u \bar{v})_{17}$$
$$(\bar{x}_2 \bar{u} \bar{v})_{18}\ (\bar{x}_2 u v)_{19}\ (x_2 \bar{u} v)_{20}\ (x_2 u \bar{v})_{21}$$
$$(\bar{a}_1 r)_{22}\ (\bar{a}_1 s)_{23}\ (a_1 \bar{r} \bar{s})_{24}$$
$$(\bar{a}_2 r)_{25}\ (\bar{a}_2 s)_{26}\ (a_2 \bar{r} \bar{s})_{27}$$

(c) CNF with clauses $C_1, \ldots, C_{27}$

XOR of the output of two (isomorphic) circuits is constant 0

Miter circuit assumes that the outputs can be different (XOR is 1)

Thus Tseitin Encoding into CNF of the miter is unsatisfiable

Circuit Equivalence Checking — Hyper Binary Resolution [CPAIOR'13]

$p \underset{1}{=} 1$

$p \underset{2}{=} m_1 \oplus m_2$

$m_1 \underset{3}{=} c\ ?\ a_1 : x_1$

$m_2 \underset{4}{=} c\ ?\ a_2 : x_2$

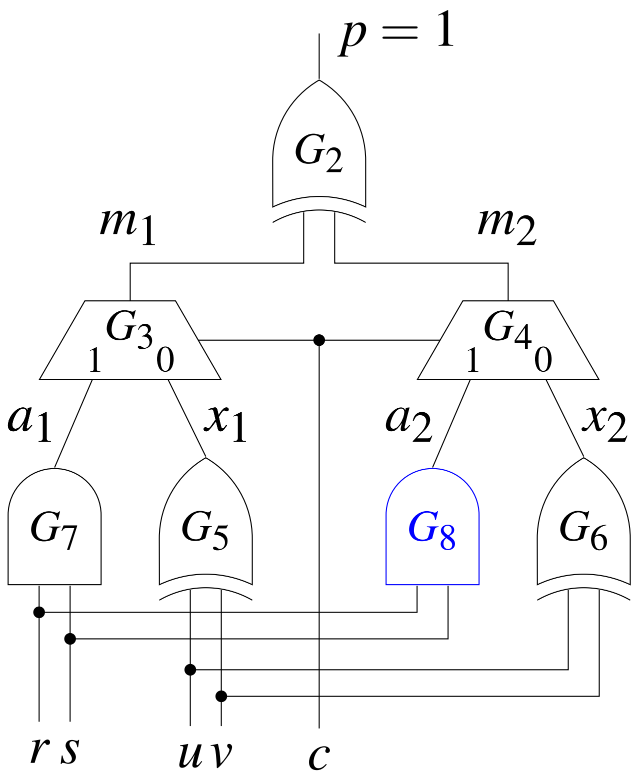
$x_1 \underset{5}{=} u \oplus v$

$x_2 \underset{6}{=} u \oplus v$

$a_1 \underset{7}{=} r \wedge s$

$a_2 \underset{8}{=} r \wedge s$

(a) gates $G_1, \ldots, G_8$



(b) miter circuit

$(p)_1$

$(\overline{p}\, m_1 m_2)_2\ (\overline{p}\, \overline{m}_1 \overline{m}_2)_3\ (p\, \overline{m}_1 m_2)_4\ (p\, m_1 \overline{m}_2)_5$

$(\overline{m}_1 \overline{c}\, a_1)_6\ (\overline{m}_1 c\, x_1)_7\ (m_1 \overline{c}\, \overline{a}_1)_8\ (m_1 c\, \overline{x}_1)_9$

$(\overline{m}_2 \overline{c}\, a_2)_{10}\ (\overline{m}_2 c\, x_2)_{11}\ (m_2 \overline{c}\, \overline{a}_2)_{12}\ (m_2 c\, \overline{x}_2)_{13}$

$(\overline{x}_1 \overline{u}\, \overline{v})_{14}\ (\overline{x}_1 u\, v)_{15}\ (x_1 \overline{u}\, v)_{16}\ (x_1 u\, \overline{v})_{17}$

$(\overline{x}_2 \overline{u}\, \overline{v})_{18}\ (\overline{x}_2 u\, v)_{19}\ (x_2 \overline{u}\, v)_{20}\ (x_2 u\, \overline{v})_{21}$

$(\overline{a}_1 r)_{22}\ (\overline{a}_1 s)_{23}\ (a_1 \overline{r}\, \overline{s})_{24}$

$(\overline{a}_2 r)_{25}\ (\overline{a}_2 s)_{26}\ (a_2 \overline{r}\, \overline{s})_{27}$

(c) CNF with clauses $C_1, \ldots, C_{27}$

Two hyper binary resolution steps yield the equivalence $a_1 = a_2$

$$\frac{(a_1 \overline{r}\, \overline{s})_{24} \quad (\overline{a}_2 r)_{25} \quad (\overline{a}_2 s)_{26}}{(a_1 \overline{a}_2)_{28}}\ \mathrm{HBR}_1 \qquad \frac{(a_2 \overline{r}\, \overline{s})_{27} \quad (\overline{a}_1 r)_{22} \quad (\overline{a}_1 s)_{23}}{(a_2 \overline{a}_1)_{29}}\ \mathrm{HBR}_2$$

Circuit Equivalence Checking — Linear Resolution Chains — RUP

$p \underset{1}{=} 1$

$p \underset{2}{=} m_1 \oplus m_2$

$m_1 \underset{3}{=} c \mathbin{?} a_1 : x_1$

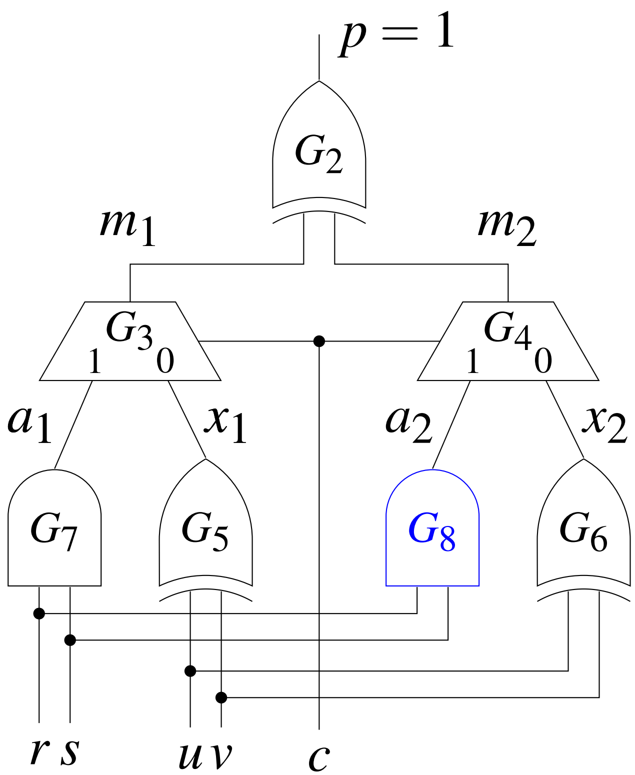
$m_2 \underset{4}{=} c \mathbin{?} a_2 : x_2$

$x_1 \underset{5}{=} u \oplus v$

$x_2 \underset{6}{=} u \oplus v$

$a_1 \underset{7}{=} r \wedge s$

$a_2 \underset{8}{=} r \wedge s$



$(p)_1$

$(\overline{p}\, m_1 m_2)_2\ (\overline{p}\, \overline{m}_1 \overline{m}_2)_3\ (p\, \overline{m}_1 m_2)_4\ (p\, m_1 \overline{m}_2)_5$

$(\overline{m}_1 \overline{c}\, a_1)_6\ (\overline{m}_1 c\, x_1)_7\ (m_1 \overline{c}\, \overline{a}_1)_8\ (m_1 c\, \overline{x}_1)_9$

$(\overline{m}_2 \overline{c}\, a_2)_{10}\ (\overline{m}_2 c\, x_2)_{11}\ (m_2 \overline{c}\, \overline{a}_2)_{12}\ (m_2 c\, \overline{x}_2)_{13}$

$(\overline{x}_1 \overline{u}\, \overline{v})_{14}\ (\overline{x}_1 u\, v)_{15}\ (x_1 \overline{u}\, v)_{16}\ (x_1 u\, \overline{v})_{17}$

$(\overline{x}_2 \overline{u}\, \overline{v})_{18}\ (\overline{x}_2 u\, v)_{19}\ (x_2 \overline{u}\, v)_{20}\ (x_2 u\, \overline{v})_{21}$

$(\overline{a}_1 r)_{22}\ (\overline{a}_1 s)_{23}\ (a_1 \overline{r}\, \overline{s})_{24}$

$(\overline{a}_2 r)_{25}\ (\overline{a}_2 s)_{26}\ (a_2 \overline{r}\, \overline{s})_{27}$

(a) gates $G_1, \ldots, G_8$ (b) miter circuit (c) CNF with clauses $C_1, \ldots, C_{27}$

The HBR steps correspond to two linear chains of resolution (RES) steps:

$$\dfrac{\dfrac{(a_1 \overline{r}\, \overline{s})_{24} \quad (\overline{a}_2 r)_{25}}{(a_1 \overline{a}_2 \overline{s})}\ \text{RES} \qquad (\overline{a}_2 s)_{26}}{(a_1 \overline{a}_2)_{28}}\ \text{RES}$$

$$\dfrac{\dfrac{(a_2 \overline{r}\, \overline{s})_{27} \quad (\overline{a}_1 r)_{22}}{(a_2 \overline{a}_1 \overline{s})}\ \text{RES} \qquad (\overline{a}_1 s)_{23}}{(a_2 \overline{a}_1)_{29}}\ \text{RES}$$

Circuit Equivalence Checking — Substitution

$p \underset{1}{=} 1$

$p \underset{2}{=} m_1 \oplus m_2$

$m_1 \underset{3}{=} c \,?\, a_1 : x_1$

$m_2 \underset{4}{=} c \,?\, a_2 : x_2$

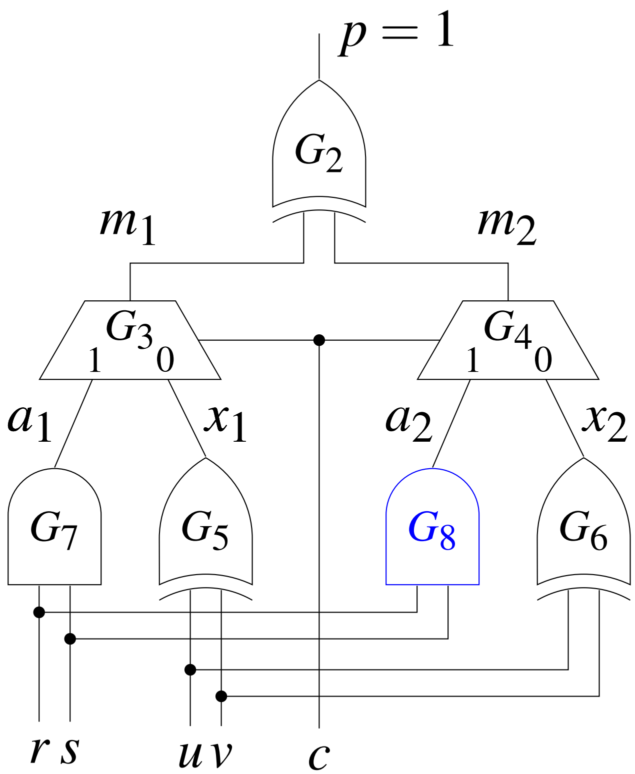
$x_1 \underset{5}{=} u \oplus v$

$x_2 \underset{6}{=} u \oplus v$

$a_1 \underset{7}{=} r \wedge s$

$a_2 \underset{8}{=} r \wedge s$

(a) gates $G_1, \ldots, G_8$



(b) miter circuit

$(p)_1$

$(\overline{p}\, m_1 m_2)_2\ (\overline{p}\, \overline{m}_1 \overline{m}_2)_3\ (p\, \overline{m}_1 m_2)_4\ (p\, m_1 \overline{m}_2)_5$

$(\overline{m}_1 \overline{c}\, a_1)_6\ (\overline{m}_1 c\, x_1)_7\ (m_1 \overline{c}\, \overline{a}_1)_8\ (m_1 c\, \overline{x}_1)_9$

$(\overline{m}_2 \overline{c}\, a_2)_{10}\ (\overline{m}_2 c\, x_2)_{11}\ (m_2 \overline{c}\, \overline{a}_2)_{12}\ (m_2 c\, \overline{x}_2)_{13}$

$(\overline{x}_1 \overline{u}\, \overline{v})_{14}\ (\overline{x}_1 u v)_{15}\ (x_1 \overline{u} v)_{16}\ (x_1 u \overline{v})_{17}$

$(\overline{x}_2 \overline{u}\, \overline{v})_{18}\ (\overline{x}_2 u v)_{19}\ (x_2 \overline{u} v)_{20}\ (x_2 u \overline{v})_{21}$

$(\overline{a}_1 r)_{22}\ (\overline{a}_1 s)_{23}\ (a_1 \overline{r}\, \overline{s})_{24}$

$(\overline{a}_2 r)_{25}\ (\overline{a}_2 s)_{26}\ (a_2 \overline{r}\, \overline{s})_{27}$

(c) CNF with clauses $C_1, \ldots, C_{27}$

Next we have to substitute (w.l.o.g.) a_2 by a_1 in the formula:

$$\frac{(\overline{m}_2 \overline{c}\, a_2)_{10} \qquad (a_1 \overline{a}_2)_{28}}{(\overline{m}_2 \overline{c}\, a_1)_{30}}\ \text{RES} \qquad\qquad \frac{(m_2 \overline{c}\, \overline{a}_2)_{12} \qquad (a_2 \overline{a}_1)_{29}}{(m_2 \overline{c}\, \overline{a}_1)_{31}}\ \text{RES}$$

Structural Hashing through Hyper-Binary Resolution Summary

The following two hyper binary resolution steps yield the equivalence $a_1 = a_2$

$$\frac{(a_1\overline{r}\,\overline{s})_{24} \quad (\overline{a}_2 r)_{25} \quad (\overline{a}_2 s)_{26}}{(a_1\overline{a}_2)_{28}}\,\mathsf{HBR}_1 \qquad \frac{(a_2\overline{r}\,\overline{s})_{27} \quad (\overline{a}_1 r)_{22} \quad (\overline{a}_1 s)_{23}}{(a_2\overline{a}_1)_{29}}\,\mathsf{HBR}_2$$

They correspond to the following two linear chains of resolution (RES) steps:

$$\frac{\dfrac{(a_1\overline{r}\,\overline{s})_{24} \quad (\overline{a}_2 r)_{25}}{(a_1\overline{a}_2\overline{s})}\,\mathsf{RES} \qquad (\overline{a}_2 s)_{26}}{(a_1\overline{a}_2)_{28}}\,\mathsf{RES} \qquad \frac{\dfrac{(a_2\overline{r}\,\overline{s})_{27} \quad (\overline{a}_1 r)_{22}}{(a_2\overline{a}_1\overline{s})}\,\mathsf{RES} \qquad (\overline{a}_1 s)_{23}}{(a_2\overline{a}_1)_{29}}\,\mathsf{RES}$$

Such linear resolution chains correspond to reverse-unit propagation (RUP)

Next we have to substitute (w.l.o.g.) a_2 by a_1 in the formula:

$$\frac{(\overline{m}_2\overline{c}\,a_2)_{10} \quad (a_1\overline{a}_2)_{28}}{(\overline{m}_2\overline{c}\,a_1)_{30}}\,\mathsf{RES} \qquad \frac{(m_2\overline{c}\,\overline{a}_2)_{12} \quad (a_2\overline{a}_1)_{29}}{(m_2\overline{c}\,\overline{a}_1)_{31}}\,\mathsf{RES}$$

Structural Hashing vs. Congruence Closure

- hash-consing (LISP)
- common-sub-expression elimination (compilers)
- unique-table in BDD and AIG libraries (`strash` in ABC)
  - often wrongly assumed to require an acyclic circuit representation (DAG)
- structural rule in Stålmarck's procedure
  - works on "sea of triples" (gate equations)
- **congruence** axiom (core rule in SMT solvers)

$$\frac{x = f(a,b) \quad y = f(c,d) \quad a = c \quad b = d}{x = y}$$

  - obviously does not require an acyclic (circuit) representation
- common implementation
  - hash right-hand-side of equations to left-hand-side variables
  - replace matching larger left-hand-size variable with smaller one
- congruence closure requires order on variables but equations can be cyclic

However ...

- structural hashing finds identical gates
  - applied recursively on the circuit solves isomorphic miters
  - but isomorphic parts emerge during inprocessing
  - and many instances are only given in CNF
- hyper binary resolution works for AND/OR gates (and negations)
  - one strategy to solve isomorphic AND/OR miters:
    simple-probing tries to simulate structural hashing
- XOR/ITE gates need additional intermediate clauses
  - which are still RUP clauses though
  - simple-probing alone does not work
  - hyper binary resolution alone neither
  - in earlier [CPAIOR'13] work we proposed to use ternary resolution
    - can in principle solve isomorphic miters with binary XOR/ITE gates
- CDCL does not find the right clauses

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

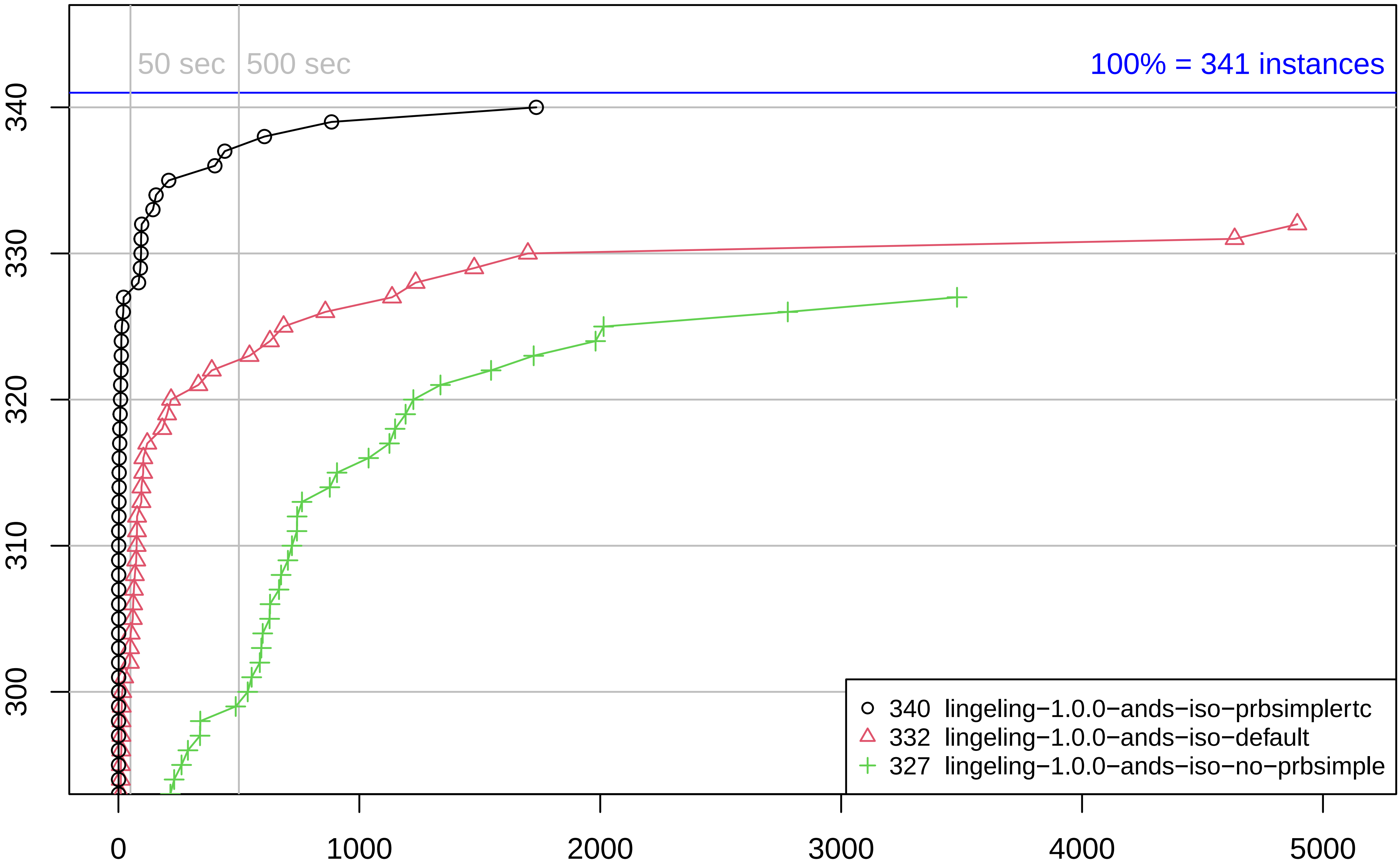
Haha, so I guess lingeling was the only solver solving that formula efficiently back then. :-D

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!

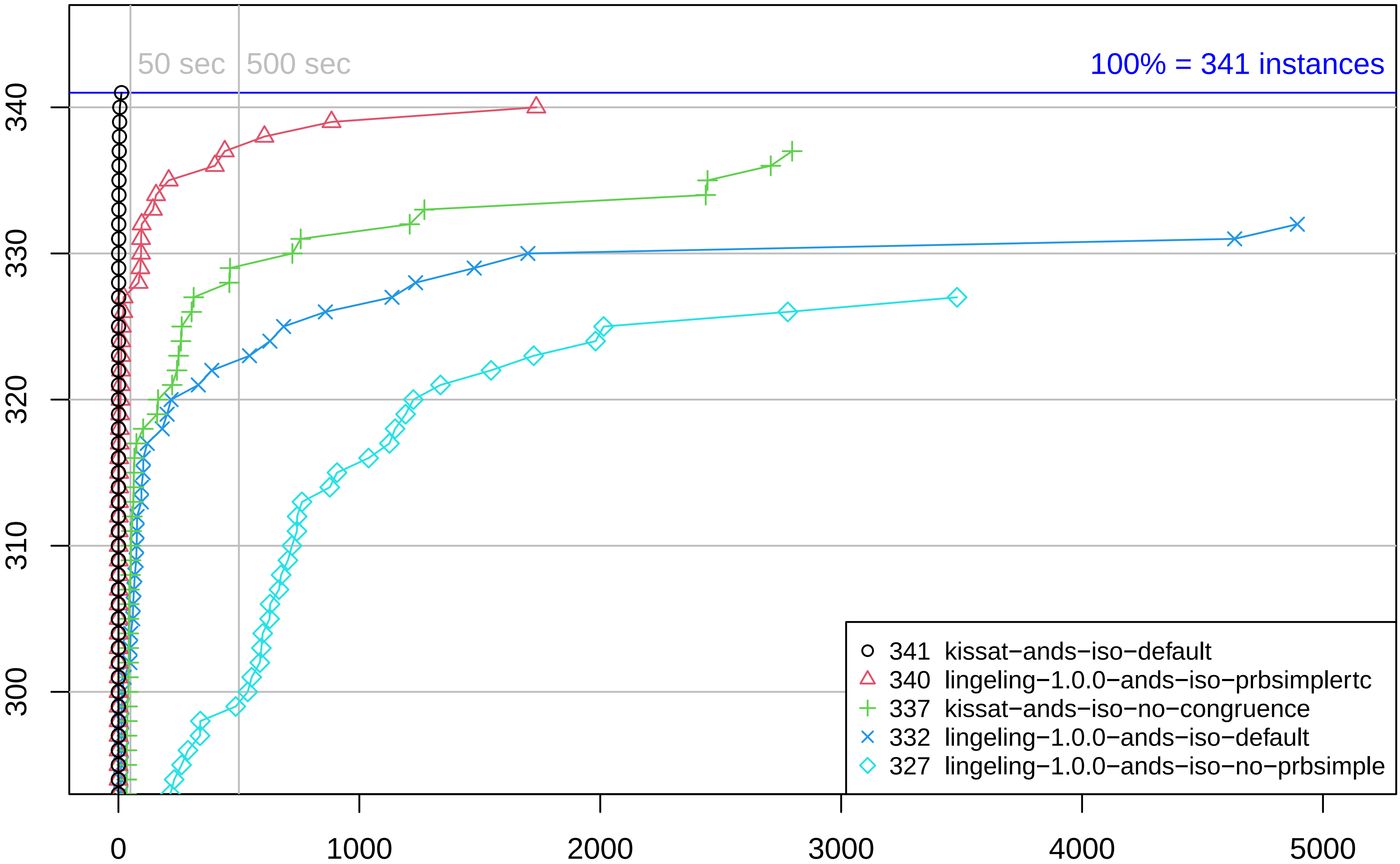
Thanks a lot,
Buddy

```
   simple-probing (CNF F)          // by reference, i.e., F updated in place
1    literals L = all literals in F
2    candidates Λ = L
3    while Λ ≠ ∅
4        pick and remove l ∈ Λ
5        for all "base" clauses C ∈ F with |C| > 2 and l ∈ C
6            for all literals k ∈ C
7                counts γ: L → ℕ initialized to γ ≡ 0
8                for all binary clauses (o ∨ k̄) ∈ F
9                    γ(o)++    // increment count of other literal o by one
10               for all r with γ(r̄) + 1 = |C| and |r| ≠ |l| and (r̄ ∨ l) ∉ F
11                   add (r̄ ∨ l) to F    // HBR
12                   if (r ∨ l̄) ∈ F       // checking for dual clause - ELS
13                       substitute l = r in all clauses D ∈ F with l or l̄ in D
14                       reschedule literals in resulting clauses by adding them to Λ
15                       continue with outer while loop at Line 3
```

Lingeling on AND Encoded Isomorphic HWMCC'12 Miters



Lingeling on AND Encoded Isomorphic HWMCC'12 Miters vs. Kissat



basic-and-gate-extraction (CNF F)

resulting AND gates $A = \emptyset$
literals L = all literals in F
for all clauses $C \in F$ with $|C| > 2$
    marks $\mu\colon L \to \mathbb{B}$ initialized to $\mu \equiv \bot$ // implemented as bit-map
    for all literals r with $\bar{r} \in C$
        $\mu(r) = \top$
    for all literals $l \in C$
        $n = 0$
        for all binary clauses $(\bar{l} \vee r) \in F$
            if $\mu(r)$ **then** $n{+}{+}$
        if $n = |C| - 1$
            let $(l \vee \bar{r}_1 \vee \ldots \vee \bar{r}_n) = C$ // structured binding
            add AND gate $(l = r_1 \wedge \cdots \wedge r_n)$ to A
return A

More sophisticated and faster version in the implementation

was in the appendix and will go into extended version of paper *basic-xor-gate-extraction* (CNF F)

1 resulting XOR gates $X = \emptyset$

2 let $\beta\colon \mathbb{N} \times \mathbb{N} \to \{0,1\}$ with $\beta(i,s) = (s/2^i) \bmod 2$ // extract i^{th} bit from s

3 let $\pi\colon \mathbb{N} \to \{0,1\}$ with $\pi(s) = |\{i \mid \beta(i,s) = 1\}| \bmod 2$ // parity of all "bits" in s

4 **for all** clauses $C = (l_0 \vee \ldots \vee l_{m-1}) \in F$ with $|C| > 2$

5   **for** s = 2 **to** $2^m - 1$ with $\pi(s) = 0$ // flip even number of sign bits

6         $D = \{l_i \mid \beta(i,s) = 0\} \cup \{\bar{l}_i \mid \beta(i,s) = 1\}$ // negate l_i if i^{th} bit set

7   **if** $D \notin F$ continue with outer loop at Line 4 // clause missing

8   **for** i = 0 **to** $m-1$ // m XOR gates of arity $m-1$

9   let $(l_i \vee k_1 \vee \ldots \vee k_{m-1})$ = C and $l = \bar{l}_i$

10   add XOR gate $(l = k_1 \oplus \cdots \oplus k_{m-1})$ to X

11 **return** X

More sophisticated and faster version in the implementation too

described in the paper

basic-ite-gate-extraction (CNF F)

1 resulting ITE gates $I = \emptyset$

2 **for all** ternary clauses $C = (l_1 \vee l_2 \vee l_3) \in F$

3   **for** $i = 1 \ldots 3$

4   let $(\bar{c} \vee \bar{l} \vee t) = C$ with $c = \bar{l}_i$

5   **if** $(\bar{c} \vee l \vee \bar{t}) \notin F$ continue with next i at Line 3

6   **for all** ternary clauses $(c \vee \bar{l} \vee e) \in F$

7   **if** $(c \vee l \vee \bar{e}) \in F$

8   add ITE gate $(l = c \mathrel{?} t : e)$ to I

9 **return** I

This basic version is still slow!

looks qubic, but is quadratic

Faster version with conditional equivalences next two slides …

Conditional Equivalences

$$(l = c \,?\, t : e) \quad \equiv \quad (c \rightarrow l = t) \wedge (\bar{c} \rightarrow l = e)$$

- split on condition variables c
- find equivalences assuming c
- find equivalences assuming $\bar{c}$
- merge them to find matching left-hand-side l

find-conditional-equivalences (CNF F, literal c)

resulting conditional equivalences $E = \emptyset$

for all ternary clauses $C = (\bar{c} \vee \bar{l} \vee t) \in F$

if $(\bar{c} \vee l \vee \bar{t}) \in F$

add $l = t$ to E

return E

merge-conditional-equivalences (literal c, equivalences E^+, equivalences E^-)

resulting ITE gates I = 0

for all equivalences $l = t$ in E^+

for all equivalences $l = e$ in E^-

add ITE gate $(l = c\, ?\, t : e)$ to I

return I

fast-ite-gate-extraction (CNF F)

resulting ITE gates I = 0

for all variables v in F

E^+ = *find-conditional-equivalences* (F, v)

E^- = *find-conditional-equivalences* (F, $\bar{v}$)

add *merge-conditional-equivalences* (v, E^+, E^-) to I

return I

merge-literals (CNF F, queue Q, representatives ρ, literals l_1, l_2) // F, Q, ρ by reference

$r_1 = \rho(l_1)$, $r_2 = \rho(l_2)$

if $r_1 = \bar{r}_2$ **then** $F = \bot$ **and return** // inconsistent equivalence thus F unsatisfiable

select $r \in \{r_1, r_2\}$ with $|r| = \min(|r_1|, |r_2|)$ // pick representative with smaller variable

update $\rho(l_1) = \rho(l_2) = r$ and $\rho(\bar{l}_1) = \rho(\bar{l}_2) = \bar{r}$

if $r \neq r_1$ **then** enqueue l_1 to Q

if $r \neq r_2$ **then** enqueue l_2 to Q

clausal-congruence-closure (CNF F) // by reference, i.e., F updated in place

G = *extract-gates* (F)

literals L = all literals in F

representatives $\rho : L \to L$ initialized to $\rho(l) = l$

Q = empty literal queue

for all $(l_1 = rhs_1), (l_2 = rhs_2) \in G$ with $rhs_1 = rhs_2$

  merge-literals (F, Q, ρ, l_1, l_2)

while $F \neq \bot$ and Q not empty dequeue l from Q

  for all gates $(k = rhs) \in G$ where l or $\bar{l}$ occurs in rhs

    use ρ to rewrite $(k = rhs)$ to $(k' = rhs')$

    remove gate $(k = rhs)$ from G

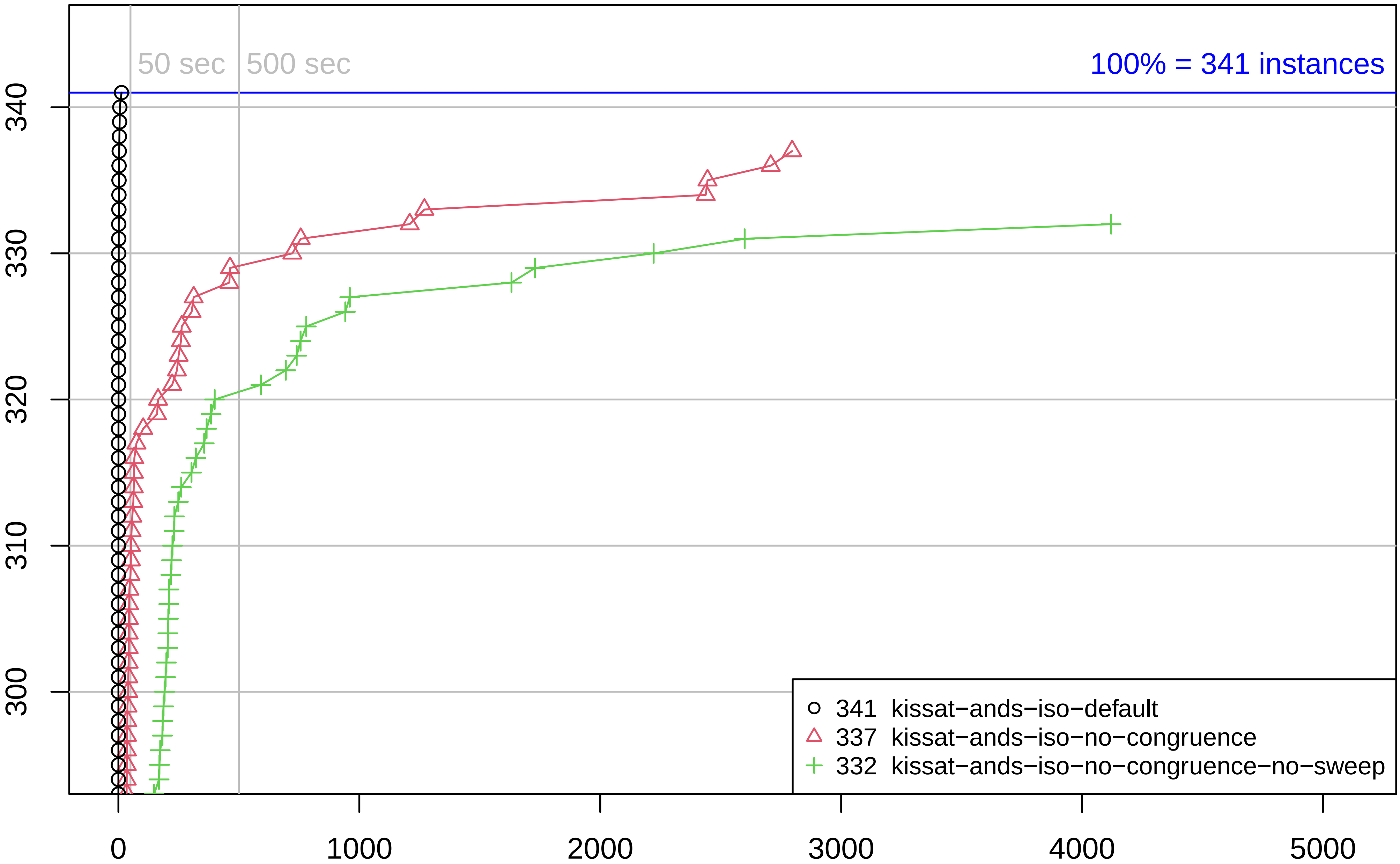
    if G contains $(k'' = rhs'')$ with $rhs' = rhs''$ **then** *merge-literals* (F, Q, ρ, k', k'')

    else add gate $(k' = rhs')$ to G

remove clauses C from F with $C \neq \rho(C) \wedge \rho(C) \in F$

replace F with $\rho(F)$

Kissat on AND encoded Isomorphic HWMCC'12 Miters



for "sweep" see our FMCAD'24 paper on "Clausal Equivalence Sweeping" with Kitten

Example of Optimized (Non-Isormophic) Miter

$p \underset{1}{=} 1$

$p \underset{2}{=} m_1 \oplus \overline{m_2}$

$m_1 \underset{3}{=} c_3 \mathbin{?} i_1 : i_3$

$m_2 \underset{4}{=} c_3 \mathbin{?} i_2 : i_4$

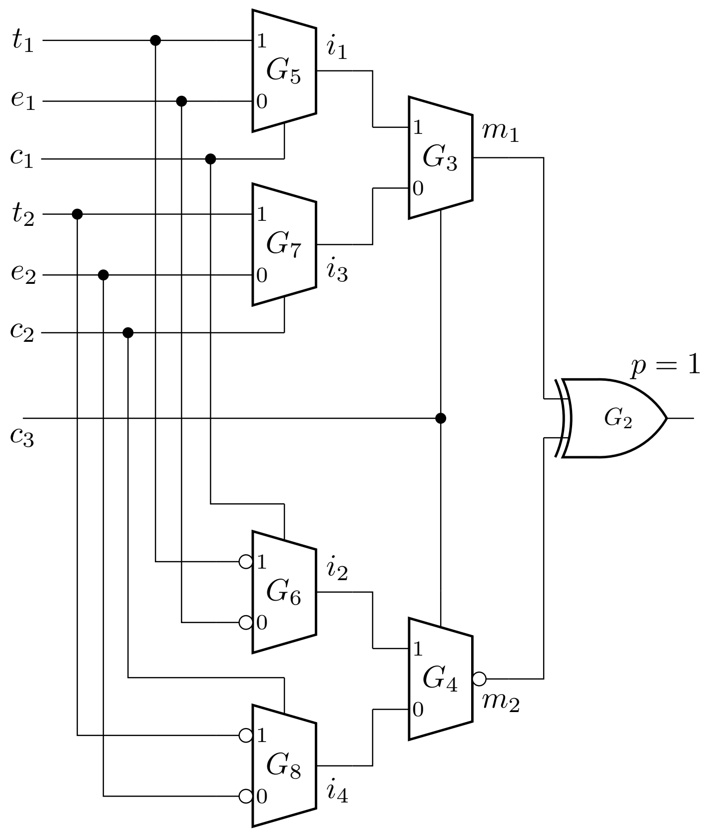
$i_1 \underset{5}{=} c_1 \mathbin{?} t_1 : e_1$

$i_2 \underset{6}{=} c_2 \mathbin{?} t_2 : e_2$

$i_3 \underset{7}{=} c_1 \mathbin{?} \overline{t_1} : \overline{e_1}$

$i_4 \underset{8}{=} c_2 \mathbin{?} \overline{t_2} : \overline{e_2}$

(a) gates $G_1, \ldots, G_8$



(b) miter circuit

$(p)_1$

$(\overline{p}\, m_1 \overline{m}_2)_2\ (\overline{p}\, \overline{m}_1 m_2)_3\ (p\, \overline{m}_1 \overline{m}_2)_4\ (p\, m_1 m_2)_5$

$(\overline{c}_3 \overline{m}_1 i_1)_6\ (\overline{c}_3 m_1 \bar{i}_1)_7\ (c_3 \overline{m}_1 i_3)_8\ (c_3 m_1 \bar{i}_3)_9$

$(\overline{c}_3 \overline{m}_2 i_2)_{10}\ (\overline{c}_3 m_2 \bar{i}_2)_{11}\ (c_3 \overline{m}_2 i_4)_{12}\ (c_3 m_2 \bar{i}_4)_{13}$

$(\overline{c}_1 \bar{i}_1 t_1)_{14}\ (\overline{c}_1 i_1 \overline{t}_1)_{15}\ (c_1 \bar{i}_1 e_1)_{16}\ (c_1 i_1 \overline{e}_1)_{17}$

$(\overline{c}_1 \bar{i}_2 \overline{t}_1)_{18}\ (\overline{c}_1 i_2 t_1)_{19}\ (c_1 \bar{i}_2 \overline{e}_1)_{20}\ (c_1 i_2 e_1)_{21}$

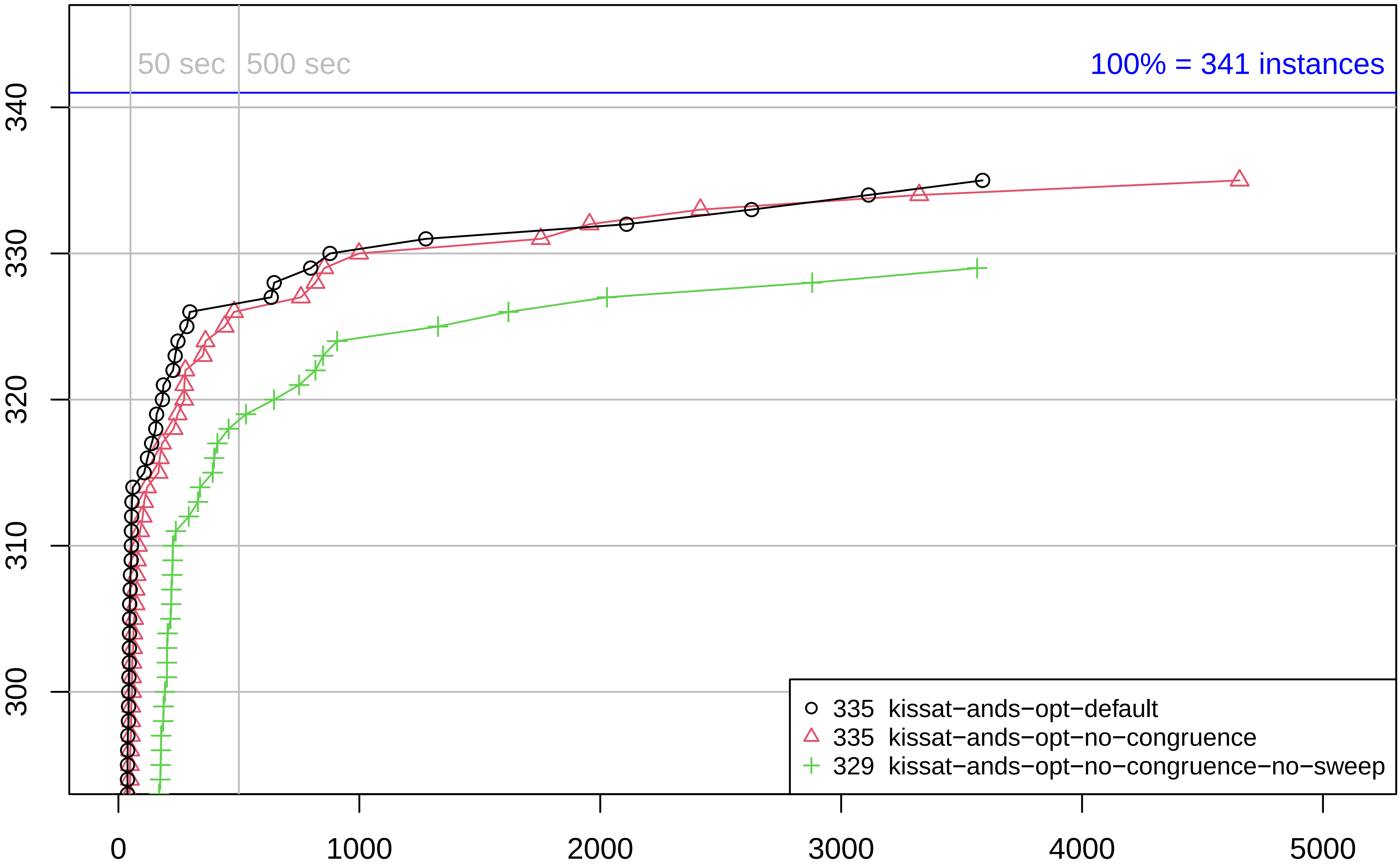
$(\overline{c}_2 \bar{i}_3 t_2)_{22}\ (\overline{c}_2 i_3 \overline{t}_2)_{23}\ (c_2 \bar{i}_3 e_2)_{24}\ (c_2 i_3 \overline{e}_2)_{25}$

$(\overline{c}_2 \bar{i}_4 \overline{t}_2)_{26}\ (\overline{c}_2 i_4 t_2)_{27}\ (c_2 \bar{i}_4 \overline{e}_2)_{28}\ (c_2 i_4 e_2)_{29}$

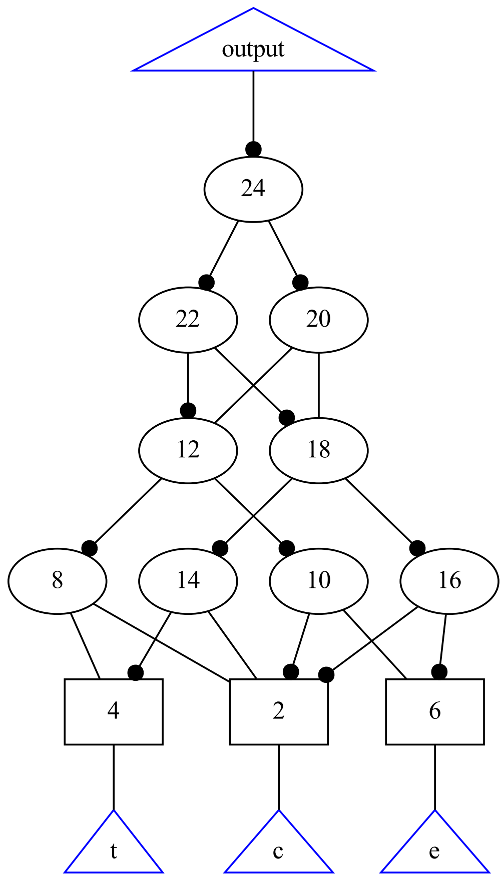
(c) CNF with clauses $C_1, \ldots, C_{29}$

To generate the optimized miters we used the ABC synthesis command `dc2` for optimization

Kissat on AND encoded Optimized HWMCC'12 Miters



XITS Encoding — Extracting XOR/ITE Gates in AIG Tseitin Encoding



Miter of two ITE gates in AIGER format.

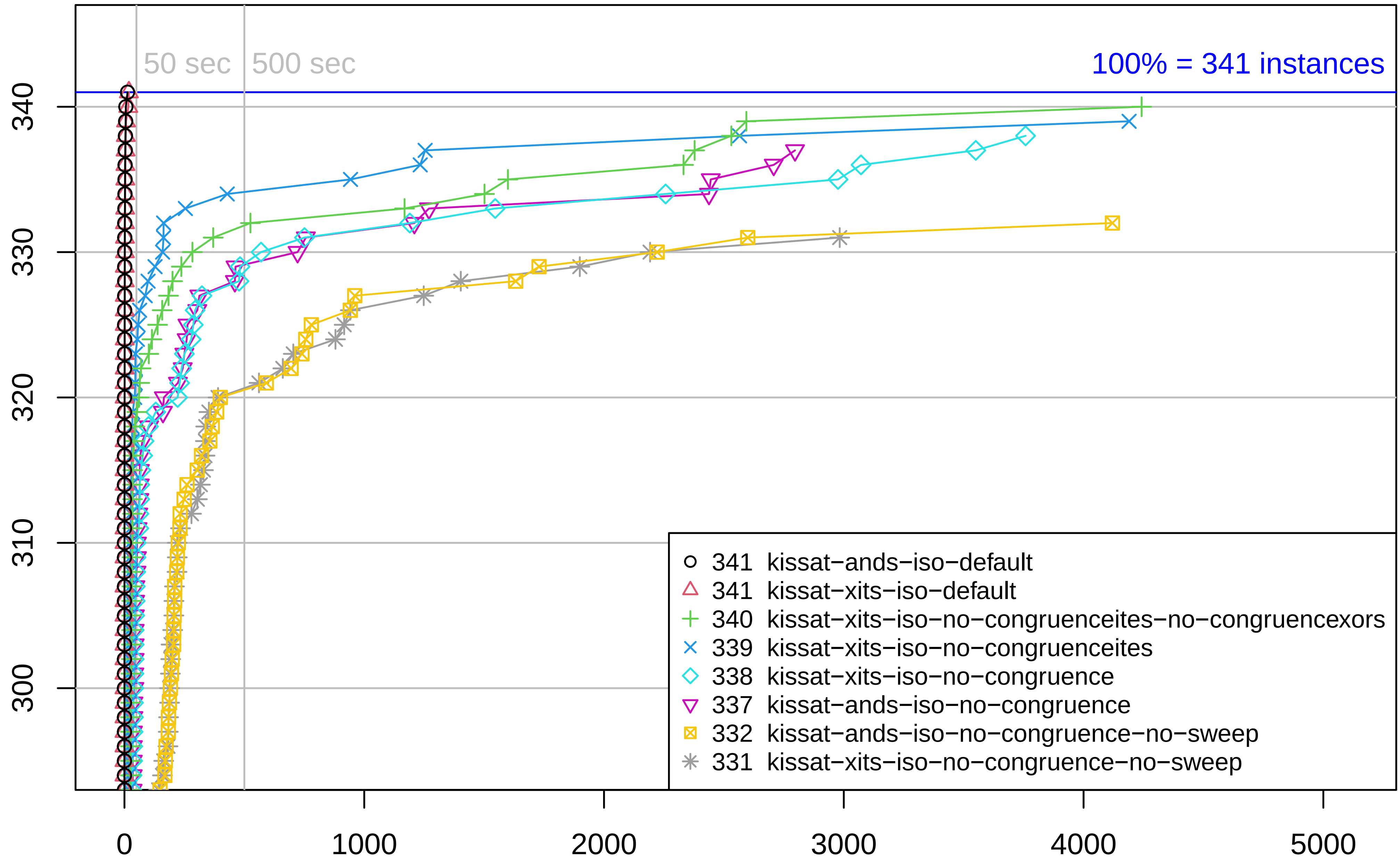
$$(\bar{x}_4 x_1), (\bar{x}_4 x_2), (x_4 \bar{x}_1 \bar{x}_2),$$
$$(\bar{x}_5 \bar{x}_1), (\bar{x}_5 x_3), (x_5 x_1 \bar{x}_3),$$
$$(\bar{x}_6 \bar{x}_4), (\bar{x}_6 \bar{x}_5), (x_6 x_4 x_5),$$
$$(\bar{x}_7 x_1), (\bar{x}_7 \bar{x}_2), (x_7 \bar{x}_1 x_2),$$
$$(\bar{x}_8 \bar{x}_1), (\bar{x}_8 \bar{x}_3), (x_8 x_1 x_3),$$
$$(\bar{x}_9 \bar{x}_7), (\bar{x}_9 \bar{x}_8), (x_9 x_7 x_8),$$
$$(\bar{x}_{10} x_6), (\bar{x}_{10} x_9), (x_{10} \bar{x}_6 \bar{x}_9),$$
$$(\bar{x}_{11} \bar{x}_6), (\bar{x}_{11} \bar{x}_9), (x_{11} x_6 x_9),$$
$$(x_{12} x_{10} x_{11}), (\bar{x}_{12}).$$

The ANDS encoding of the AIG.

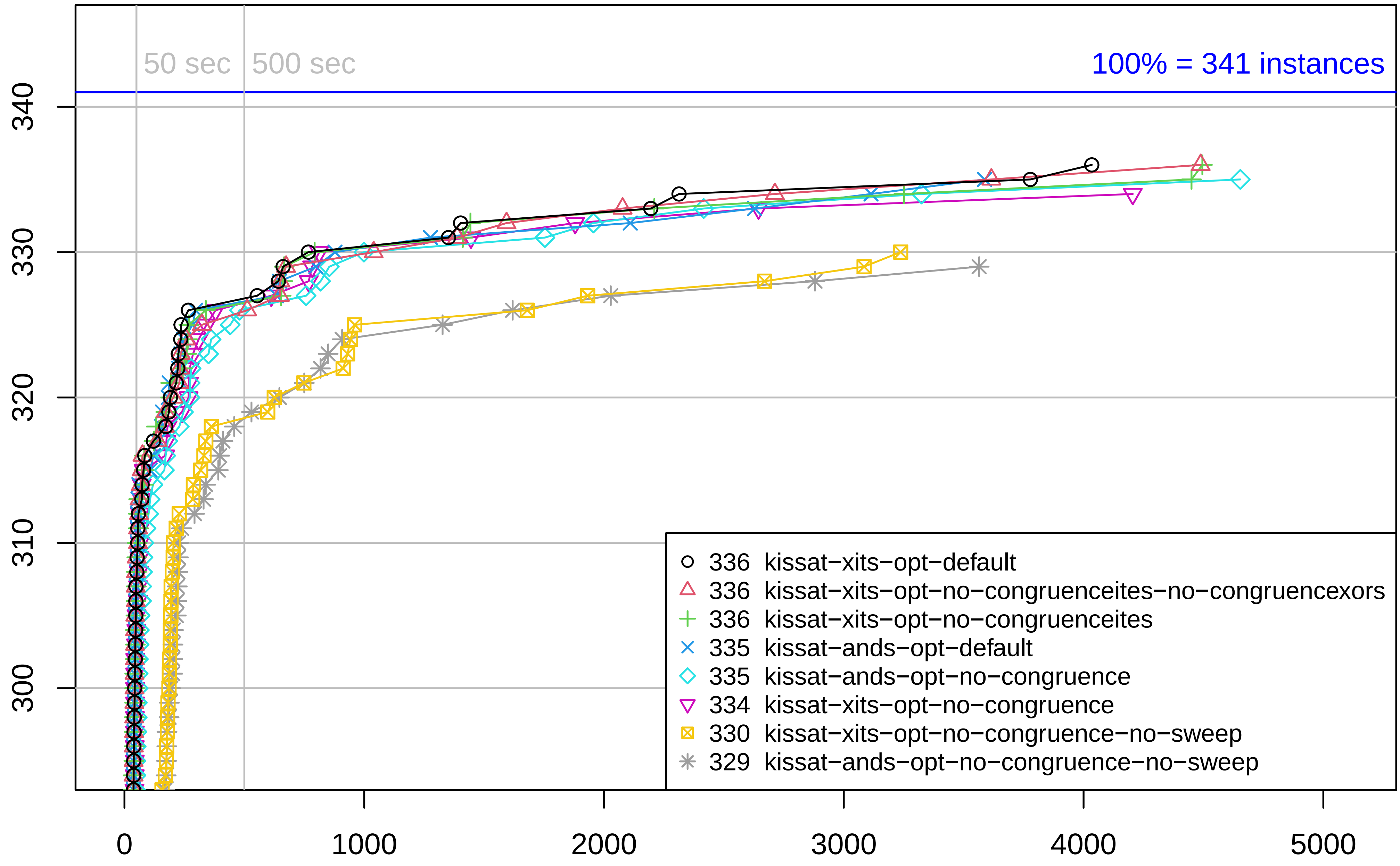
$$(\bar{x}_4 \bar{x}_1 x_3), (\bar{x}_4 x_1 x_2),$$
$$(x_4 \bar{x}_1 \bar{x}_3), (x_4 x_1 \bar{x}_2),$$
$$(\bar{x}_5 \bar{x}_1 \bar{x}_3), (\bar{x}_5 x_1 \bar{x}_2),$$
$$(x_5 \bar{x}_1 x_3), (x_5 x_1 x_2),$$
$$(x_6 \bar{x}_5 x_4), (x_6 x_5 \bar{x}_4), (\bar{x}_6).$$

The XITS encoding of the AIG.

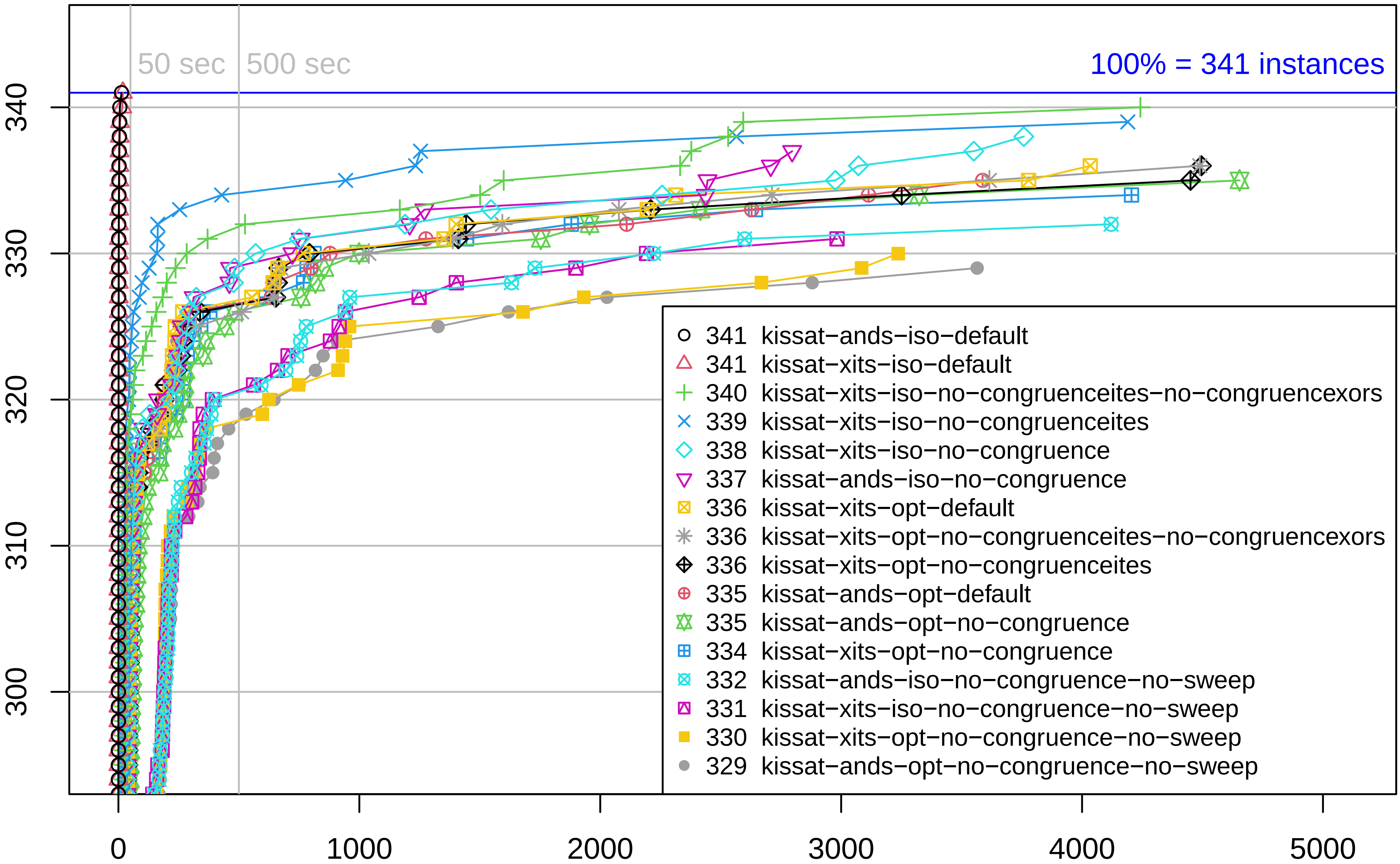
Kissat on ANDs and XITS encoded Isomorphic HWMCC'12 Miters



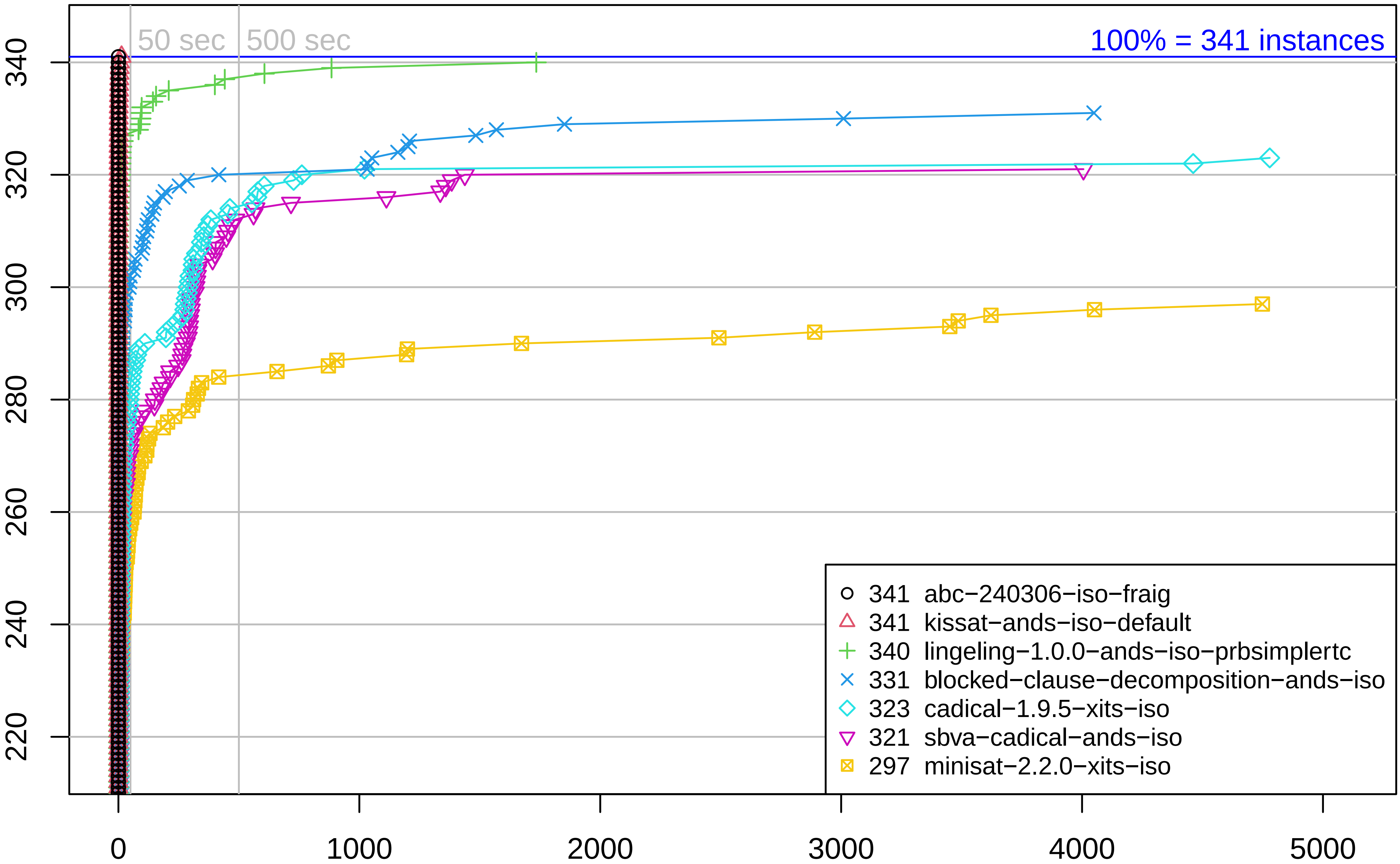
Kissat on ANDs and XITS encoded Optimized HWMCC'12 Miters



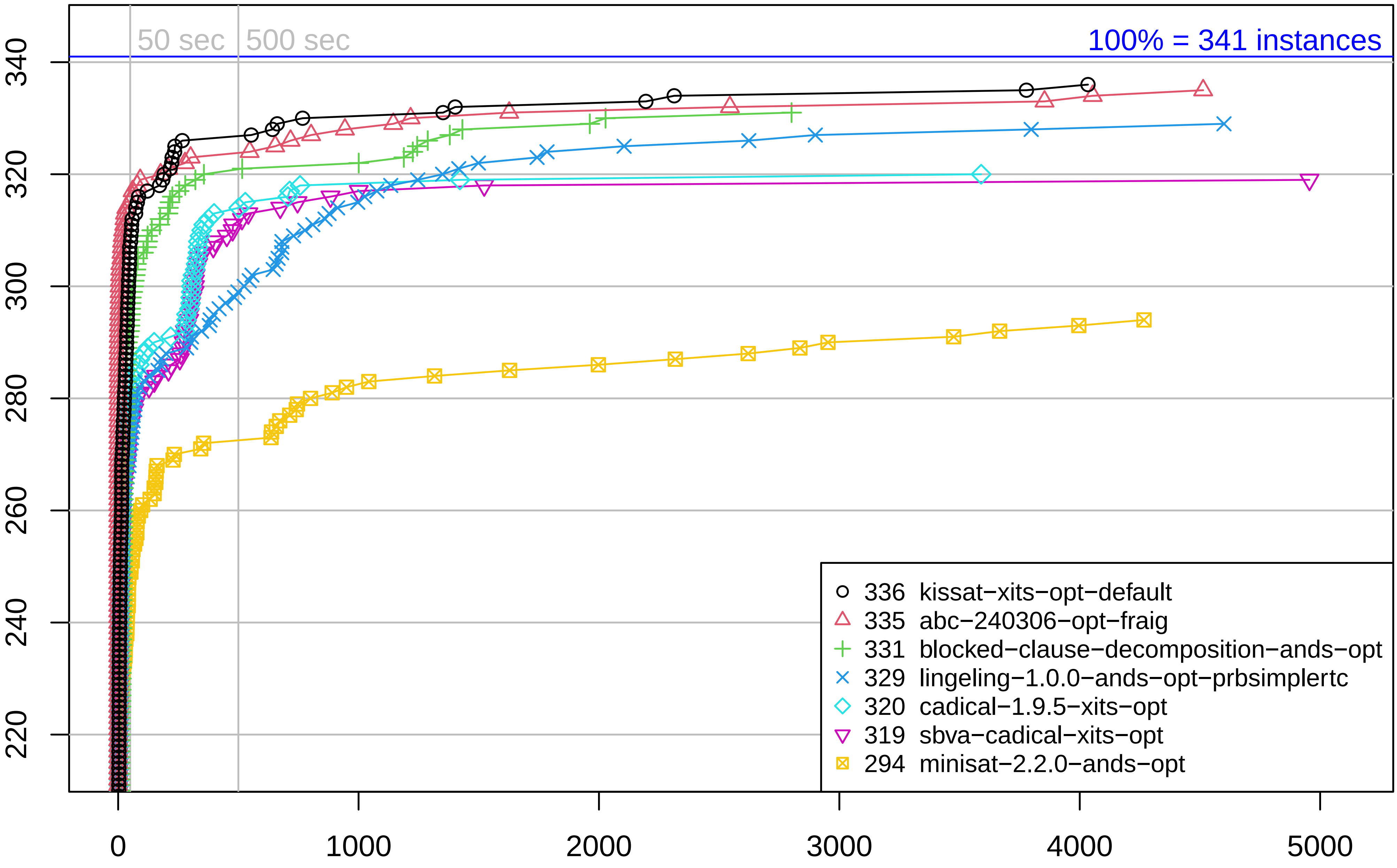
All Configurations of Kissat on all HWMCC'12 Miters



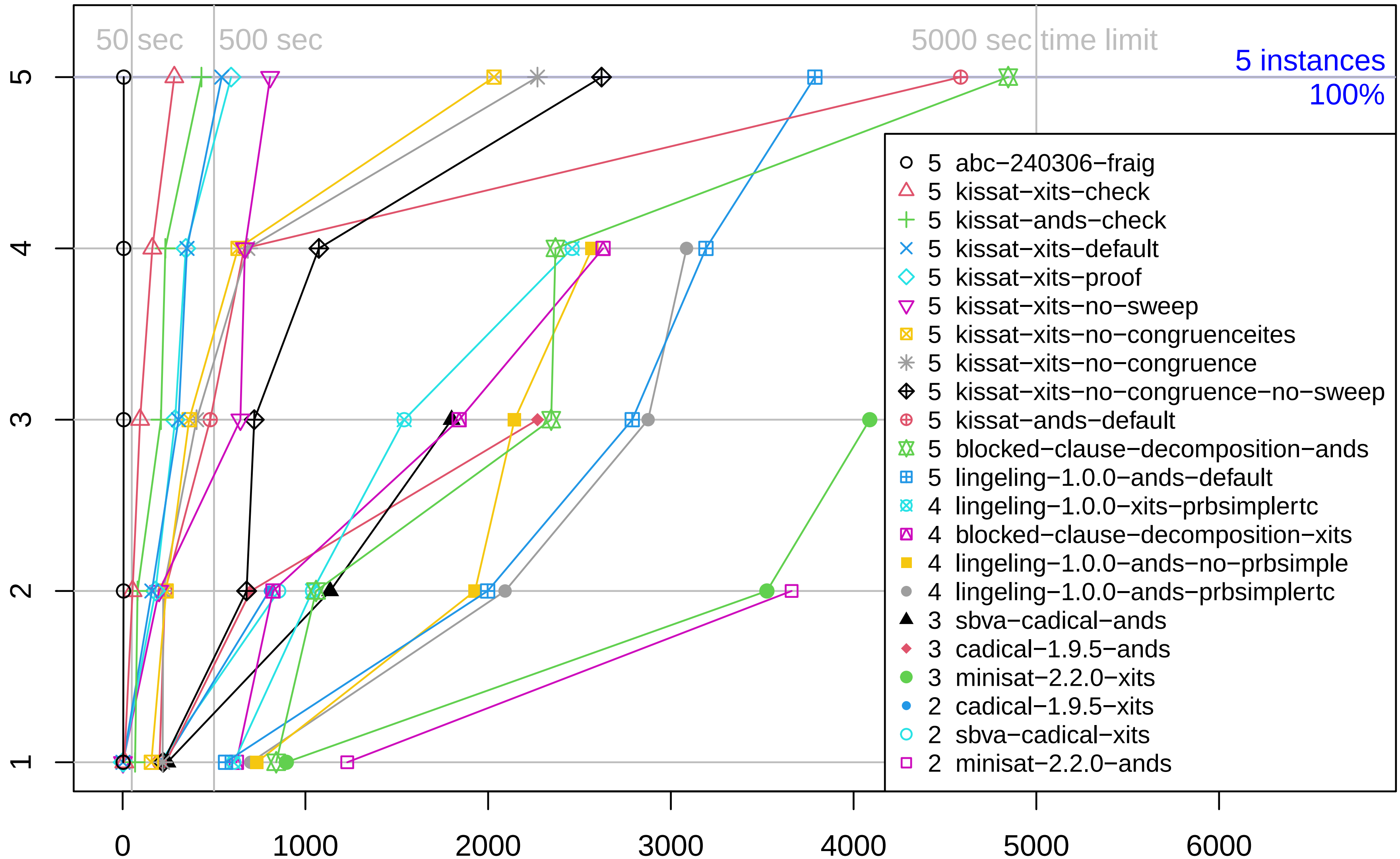
Best Solver Configuration on Isomorphic HWMCC'12 Miters



Best Solver Configuration on Optimized HWMCC'12 Miters



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



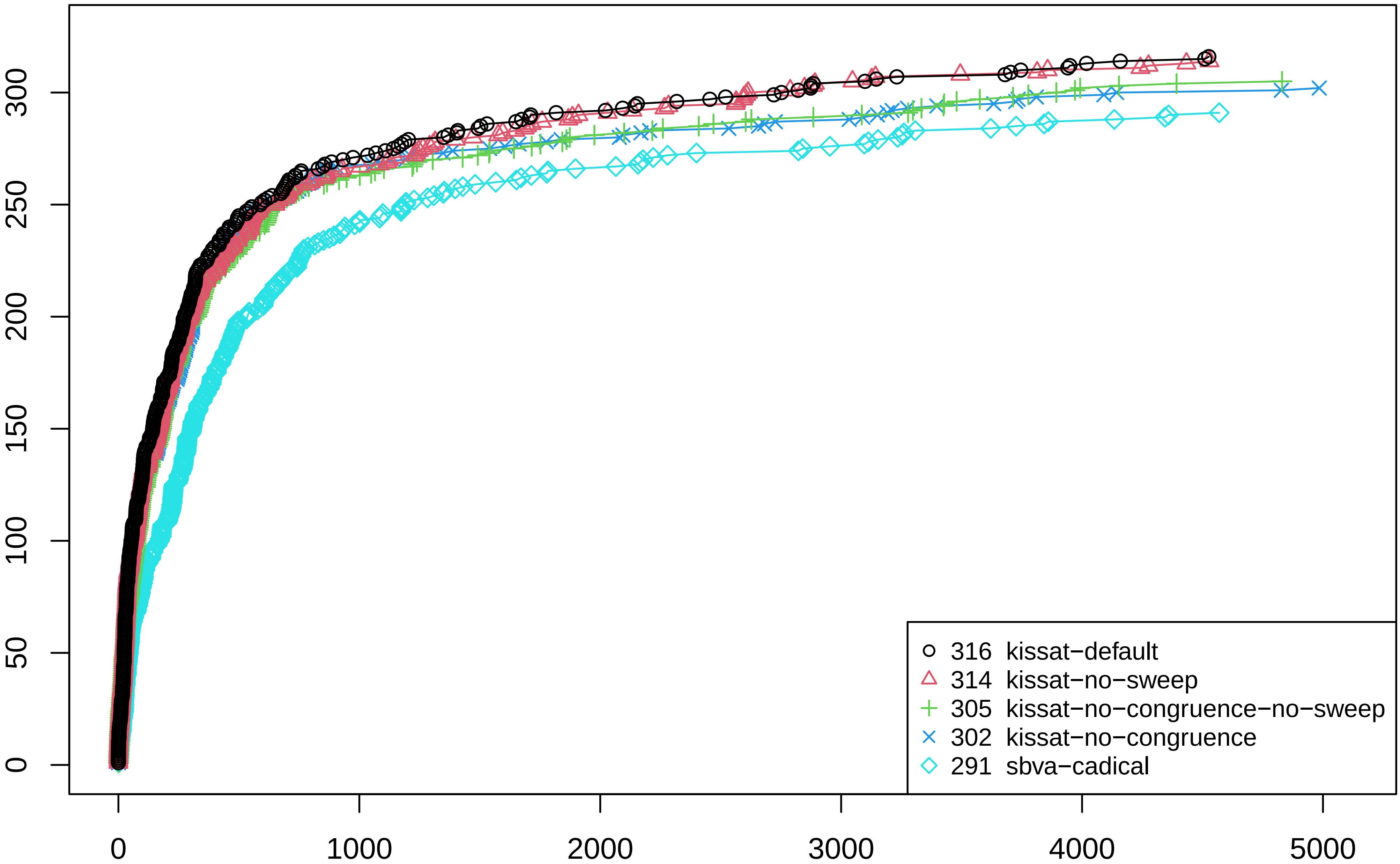
[IWLS'22] *He-Teng Zhang, Jie-Hong R. Jiang, Alan Mishchenko, and Luca Amarù*. "Improved large-scale SAT sweeping"

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

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

	n01	n04	n06	test01	test02
abc-240306-fraig	5.96	5.38	4.86	2.89	5.75
kissat-xits-check	95.45	162.38	282.61	54.28	9.06
kissat-ands-check	81.57	209.54	233.75	67.95	431.21
kissat-xits-default	305.60	160.01	542.18	352.15	1.79
kissat-xits-proof	287.21	179.72	593.54	345.60	2.38
kissat-xits-no-sweep	199.17	807.07	644.22	669.11	1.79
kissat-xits-no-congruenceites	238.32	157.06	631.46	363.93	2032.41
kissat-xits-no-congruence	222.25	218.73	684.94	404.48	2270.00
kissat-xits-no-congruence-no-sweep	221.25	678.17	720.29	1073.75	2620.65
kissat-ands-default	231.87	201.45	664.81	479.28	4585.76
blocked-clause-decomposition-ands	840.19	1058.28	2345.20	2368.54	4846.14
lingeling-1.0.0-ands-default	563.03	3192.09	1997.28	2788.51	3788.10
lingeling-1.0.0-xits-prbsimplertc	607.82	1039.04	1540.55	2459.75	—
blocked-clause-decomposition-xits	622.46	822.68	1841.48	2628.96	—
lingeling-1.0.0-ands-no-prbsimple	733.61	1928.03	2144.69	2568.83	—
lingeling-1.0.0-ands-prbsimplertc	700.58	3085.86	2092.79	2875.45	—
sbva-cadical-ands	244.94	1800.21	1135.28	—	—
cadical-1.9.5-ands	236.14	2270.17	701.13	—	—
minisat-2.2.0-xits	895.77	4088.40	3525.61	—	—
cadical-1.9.5-xits	227.21	—	801.69	—	—
sbva-cadical-xits	205.70	—	853.77	—	—
minisat-2.2.0-ands	1229.07	—	3660.71	—	—

Kissat and SBVA-CaDiCaL on 400 SAT Competition 2022 Benchmarks



Conclusion and Future Work

- First approach which *instantly* solves large isomorphic CNF-encoded miters!
- Complements semantic SAT-sweeping with our embedded SAT solver Kitten
  - which by itself is too slow *but*
  - see our upcoming FMCAD'24 paper on “Clausal Equivalence Sweeping”
- Beneficial for other benchmarks too
  - optimized miters (industrial use-case)
  - paper has comparison with hard miters from state-of-the-art circuit approach
  - SAT competition benchmarks have many congruences too
- Ongoing work is to extend paper with description of all optimizations
- Port clausal congruence closure to CaDiCaL
- How to cheaply achieve (even) more semantic rewriting?
- How to produce linear proofs (LRAT)?