Weaknesses of CDCL Solvers

Armin Biere
Johannes Kepler University
Linz, Austria

Workshop on

Theoretical Foundations of SAT Solving

The Fields Institute, Toronto, Canada
Tuesday, August 16, 2016

JOHANNES KEPLER
UNIVERSITY LINZ

Four Parts

- Evaluating CDCL Variable Scoring Schemes [SAT’15 paper with Andreas Fröhlich]
  - continued discussion from our Banff and Dagstuhl workshops
  - shows that VMTF **is** as good as VSIDS (and explains boths)
- Evaluating CDCL Restart Schemes [POS’15 paper with Andreas Fröhlich]
  - simplifies Glucose style restart schemes
  - evaluation shows clear benefits but also weaknesses
- Weaknesses of CDCL for Equivalence Checking (miters)
  - CDCL has a hard time to learn the right clauses
  - fast restarts important for miters
- Arithmetic Reasoning Hard for CDCL

Part I

- Evaluating CDCL Variable Scoring Schemes [SAT'15 paper with Andreas Fröhlich]
  - continued discussion from our Banff and Dagstuhl workshops
  - shows that VMTF **is** as good as VSIDS (and explains boths)
- Evaluating CDCL Restart Schemes [POS'15 paper with Andreas Fröhlich]
  - simplifies Glucose style restart schemes
  - evaluation shows clear benefits but also weaknesses
- Weaknesses of CDCL for Equivalence Checking (miters)
  - CDCL has a hard time to learn the right clauses
  - fast restarts important for miters
- Arithmetic Reasoning Hard for CDCL

Motivation Part I

- (E)VSIDS decision heuristic
  - (Exponential) Variable State Independent Decaying Sum (VSIDS)
  - empirically one of the **most important features** of state-of-the-art solvers
  - no formal argument “why it works”
- reconsider simpler alternatives
  - particularly variable move to front schemes (VMTF)
  - requires careful data structure design
  - formalization of these heuristics
  - empirical evaluation
- so a step towards “Trying to Understand the Power of VSIDS”
  - . . . and thus “**Why** CDCL Works”

original longer slide set at http://fmv.jku.at/biere/talks/Biere-SAT15-talk.pdf

Decision Heuristics

- decision heuristics consist of
  - variable selection: which variable to assign next?
  - phase selection: assign variable to which phase (true or false)?
- phase saving [PipatsrisawatDarwiche'07]
  - select phase to which variable was assigned before
  - initialized by static one-side heuristics [JeroslowWang'90]
  - very effective and thus default in state-of-the-art solvers
- we consider only variable selection as decision heuristic here
  - clause based heuristics less effective (BerkMin, CMTF)
  - same applies to literal based heuristics (using literal scores)
- variable selection and **decision heuristic** boils down to
  - compute and maintain heuristic scores for variables
  - **select variable with highest score**

Variable Scoring Schemes

- how to compute scores
  - static or dynamic
  - **bump** variables: when to increase scores and by how much
  - **rescore** variables: when to decrease scores and by how much
  - state-of-the-art: VSIDS (from Chaff)

    more precisely the exponential variant (EVSIDS) of MiniSAT!

- data structures for finding decision variables
  - eager or lazy update of “order”
  - state-of-the-art: priority queue of variables ordered by score (MiniSAT)

- data structure depends on how scores are computed and vice versa

Variable Scoring Schemes

- zero order scheme = static scores
  - computed for instance once during preprocessing
  - still needs search for “best” unassinged variable
  - only total orders considered so far
- first order schemes = dynamic but state less
  - for instance: score = pos occs × neg occs
  - independent of how search reached current branch / search node
  - might be quite expensive to compute / update (linear in CNF size)
- second order schemes: variable score depends on history of search
  - first order + learning ⇒ second order
  - but can also be used to speed up search for “best” variable
  - goal is logarithmic or even constant algorithm for variable selection

VSIDS

- VSIDS appeared in seminal Chaff paper from Princeton (2001)
- bump variables occurring in learned clauses
  - bumping means incrementing an integer VSIDS score

    current state-of-the-art: bump all variables used to derive learned clause
  - independent of state of clauses (satisfied or not)
- rescoring gives focus on recently used variables
  - scores are "decayed", e.g., originally divided by two every 256th conflict
  - "low pass filter" on "use frequency" of variables

    video http://youtu.be/MOjhFywLre8
- search for next unassigned variable with largest score
  - keeps an array of variables sorted by score
  - only re-sorts it w.r.t. score during rescoring (every 256th conflict)
  - uses right-most unassigned variable, thus original implementation **imprecise**

Variable Move To Front

- Siege SAT solver [Ryan'04] used variable move to front (VMTF)
  - bumped variables moved to head of doubly linked list
  - search for unassigned variable starts at head
  - variable selection is an **online sorting algorithm of scores**
  - classic “move-to-front” strategy achieves good amortized complexity

- original implementation severely restricted
  - only moved a subset of bumped variables
  - details about caching the search not described
    no source code published either
  - not exactly the same as VSIDS

- as consequence VMTF not used in state-of-the-art solvers

MiniSAT's Exponential VSIDS (EVSIDS)

- **floating point scores**
  - allows fine grained rescore at every conflict
  - consider multiplying by $f = 0.9$ every score at each conflict
- actually, instead of updating scores of all variables (at every conflict)
  - only increase score of bumped variables by g^i
  - with i the conflict-index, and $g = 1/f$
  - non-bumped variables not touched
- priority queue of variables ordered by score
  - implemented as **binary heap** with update (bump and bubble up)
  - lazy assigned variable removal
    - remove largest score variable from heap until unassigned one found
    - put unassigned variables not on the heap back (logarithmic complexity)
- normalized VSIDS (NVSIDS) $\in [0,1]$ as (theoretical) model [Biere'08] + video

Summary Variable Scoring Schemes

s old score s' new score

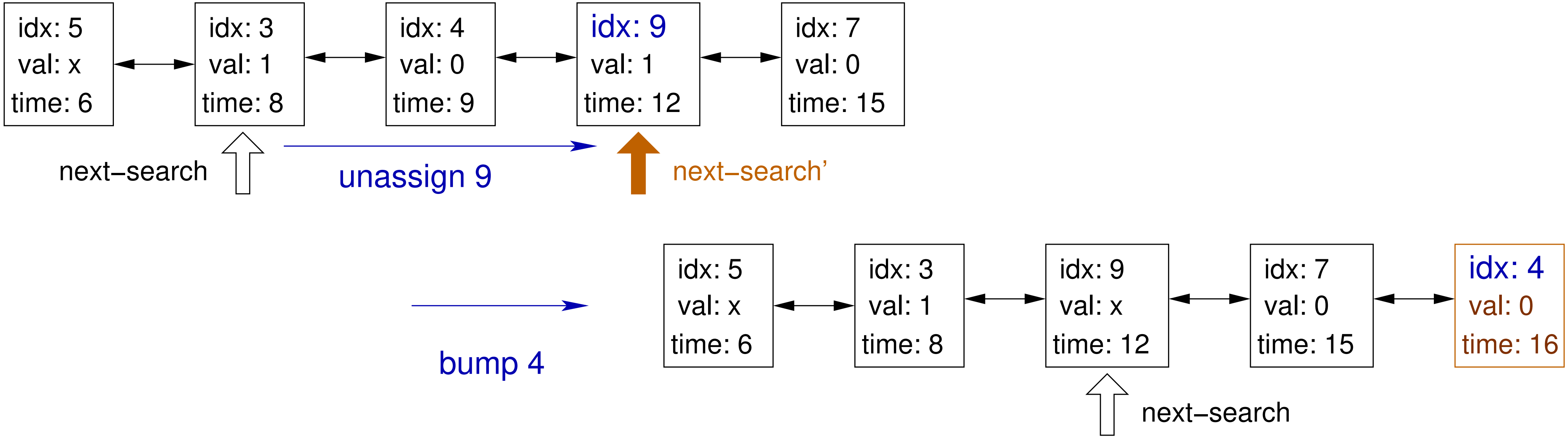
	variable score s' after i conflicts		
	bumped	not-bumped	
STATIC	s	s	static decision order
INC	$s+1$	s	increment scores
SUM	$s+i$	s	sum of conflict-indices
VSIDS	$h_i^{256} \cdot s + 1$	$h_i^{256} \cdot s$	original implementation in Chaff
NVSIDS	$f \cdot s + (1-f)$	$f \cdot s$	normalized variant of VSIDS
EVSIDS	$s + g^i$	s	exponential MiniSAT dual of NVSIDS
ACIDS	$(s+i)/2$	s	average conflict-index decision scheme
VMTF	i	s	variable move-to-front
VMTF’	b	s	variable move-to-front variant

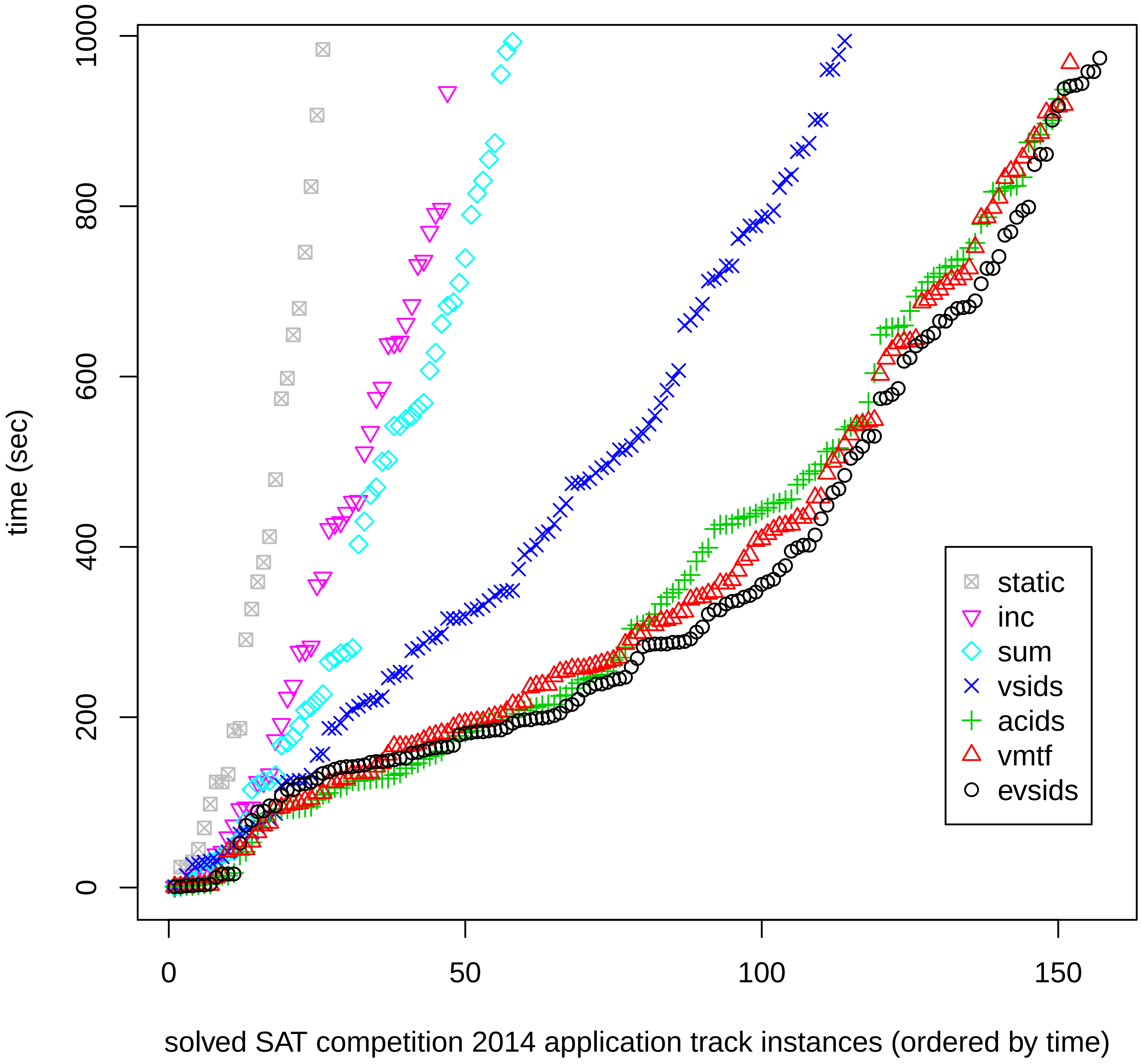
$0 < f < 1$ $g = 1/f$ $h_i^m = 0.5$ if m divides i $h_i^m = 1$ otherwise

i conflict index b bumped counter

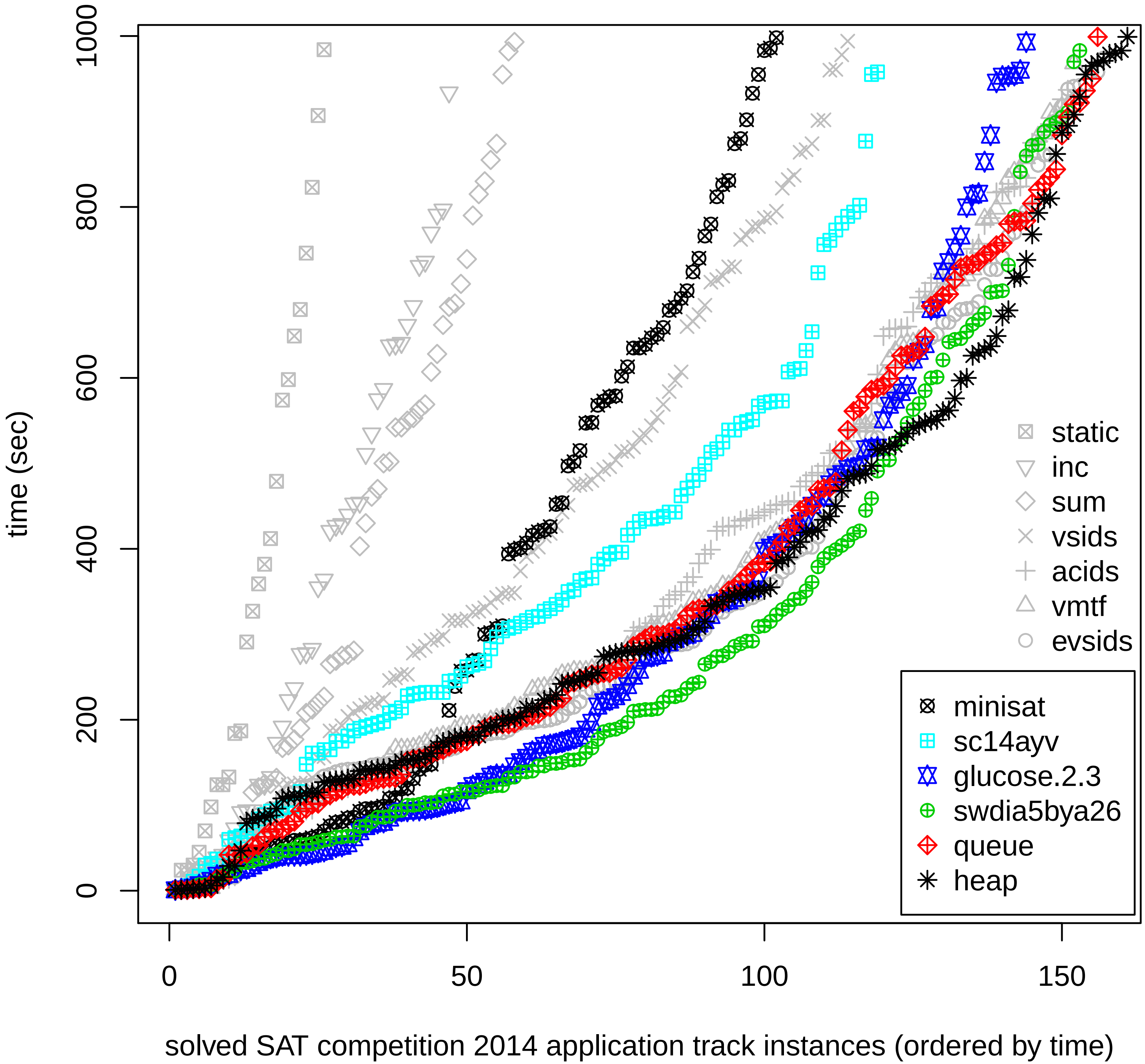
Fast VMTF Implementation

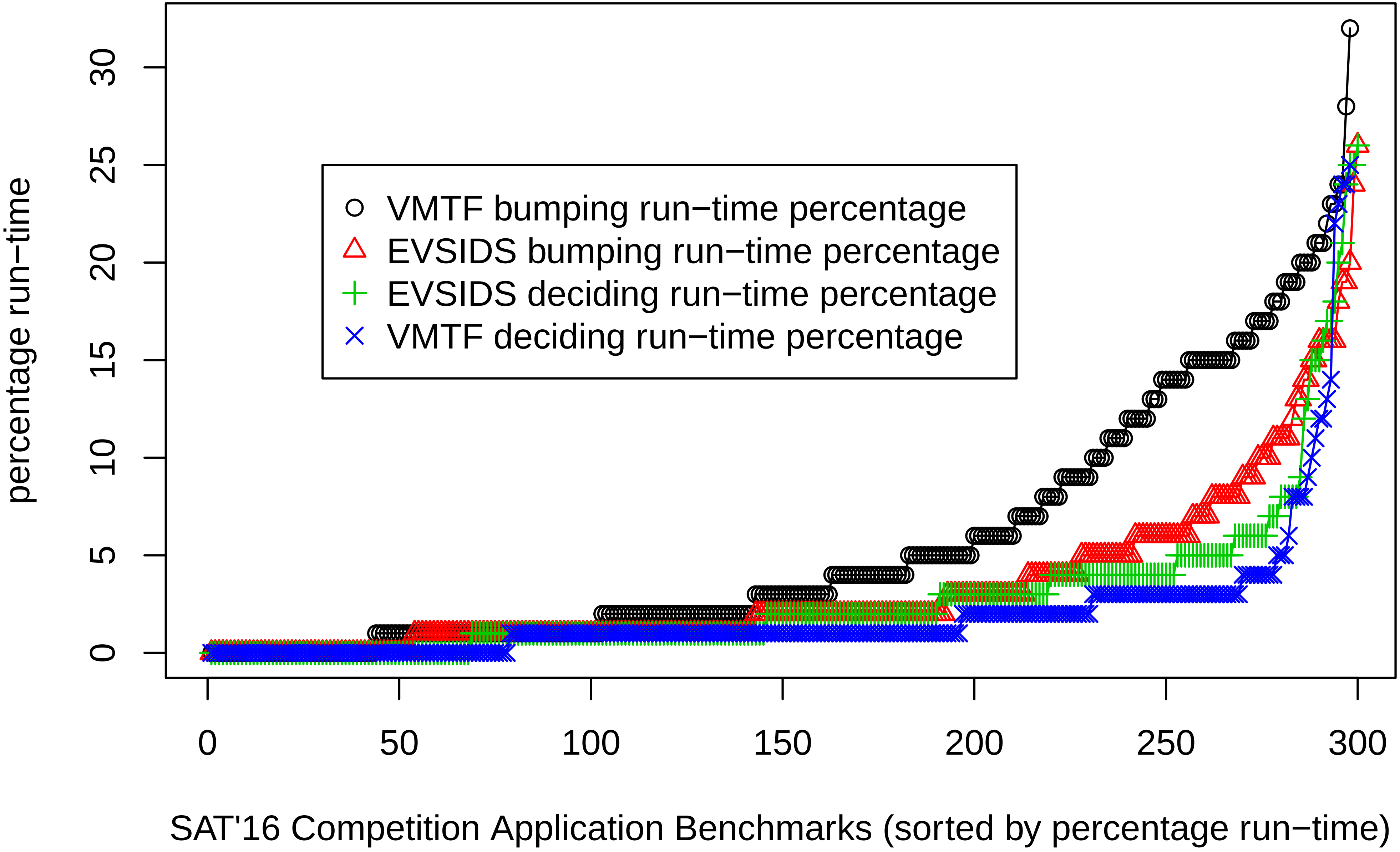
- fast simple implementation for caching searches in VMTF [Biere'15]
  - doubly linked list does not have positions as an ordered array
  - bump = move-to-front = dequeue then insertion at the head
- time-stamp list entries with insertion time
  - maintained invariant: all variables right of next-search are assigned
  - requires update to next-search while unassigning variables





time (sec)
0
200
400
600
800
1000
0
50
100
150
solved SAT competition 2014 application track instances (ordered by time)
static
inc
sum
vsids
acids
vmtf
evsids


time (sec)
0
200
400
600
800
1000
0
50
100
150
solved SAT competition 2014 application track instances (ordered by time)
static
inc
sum
vsids
acids
vmtf
evsids
minisat
sc14ayv
glucose.2.3
swdia5bya26
queue
heap


percentage run−time
0
5
10
15
20
25
30
○ VMTF bumping run−time percentage
△ EVSIDS bumping run−time percentage
+ EVSIDS deciding run−time percentage
× VMTF deciding run−time percentage
0
50
100
150
200
250
300
SAT'16 Competition Application Benchmarks (sorted by percentage run−time)

Conclusion Part I

- surveyed and classified variable selection / scoring schemes
  - came up with a new one ACIDS (as well as SUM)
  - **EVSIDS, VMTF, ACIDS comparable in performance**
  - with a generic fast queue implementation
- VMTF was considered to be obsolete
  - can be made effective (with less code than EVSIDS)
  - needs proper optimized implementation: time-stamping with insertion-time
  - VMTF might be easier to reason about in proof complexity
- threads to validity
  - unclear whether VMTF **only** works in combination with Glucose restarts

    see also our POS'15 paper and talk in Part II of this talk
  - benchmark selection in recent SAT competitions highly controversial
- Splatz: SAT solver only based on VMTF

  http://fmv.jku.at/splatz

- Evaluating CDCL Variable Scoring Schemes [SAT’15 paper with Andreas Fröhlich]
  - continued discussion from our Banff and Dagstuhl workshops
  - shows that VMTF **is** as good as VSIDS (and explains boths)
- Evaluating CDCL Restart Schemes [POS’15 paper with Andreas Fröhlich]
  - simplifies Glucose style restart schemes
  - evaluation shows clear benefits but also weaknesses
- Weaknesses of CDCL for Equivalence Checking (miters)
  - CDCL has a hard time to learn the right clauses
  - fast restarts important for miters
- Arithmetic Reasoning Hard for CDCL

Post SAT Competition 2014 Analysis

- Lingeling actually barely won
  - only for long time limit of 5000 seconds
  - for 900 seconds: no chance
- two main reasons
  - selected benchmark biased towards decendants of Glucose / MiniSAT
  - but Glucose restarts are important for many (selected) benchmarks
- the POS'15 paper is about lessons learned while
  - porting the Glucose restart scheme to Lingeling
  - and **simplifying** by

    using exponential moving averages (*EMA*)

original longer slide set at http://fmv.jku.at/biere/talks/Biere-POS15-talk.pdf

Buckets

application track instances clustered in **buckets** (by the organizers):

2d-strip-packing (4), argumentation (20), bio (11),

crypto-aes (8), crypto-des (7), crypto-gos (9),
crypto-md5 (21), crypto-sha (29), crypto-vpmc (4),

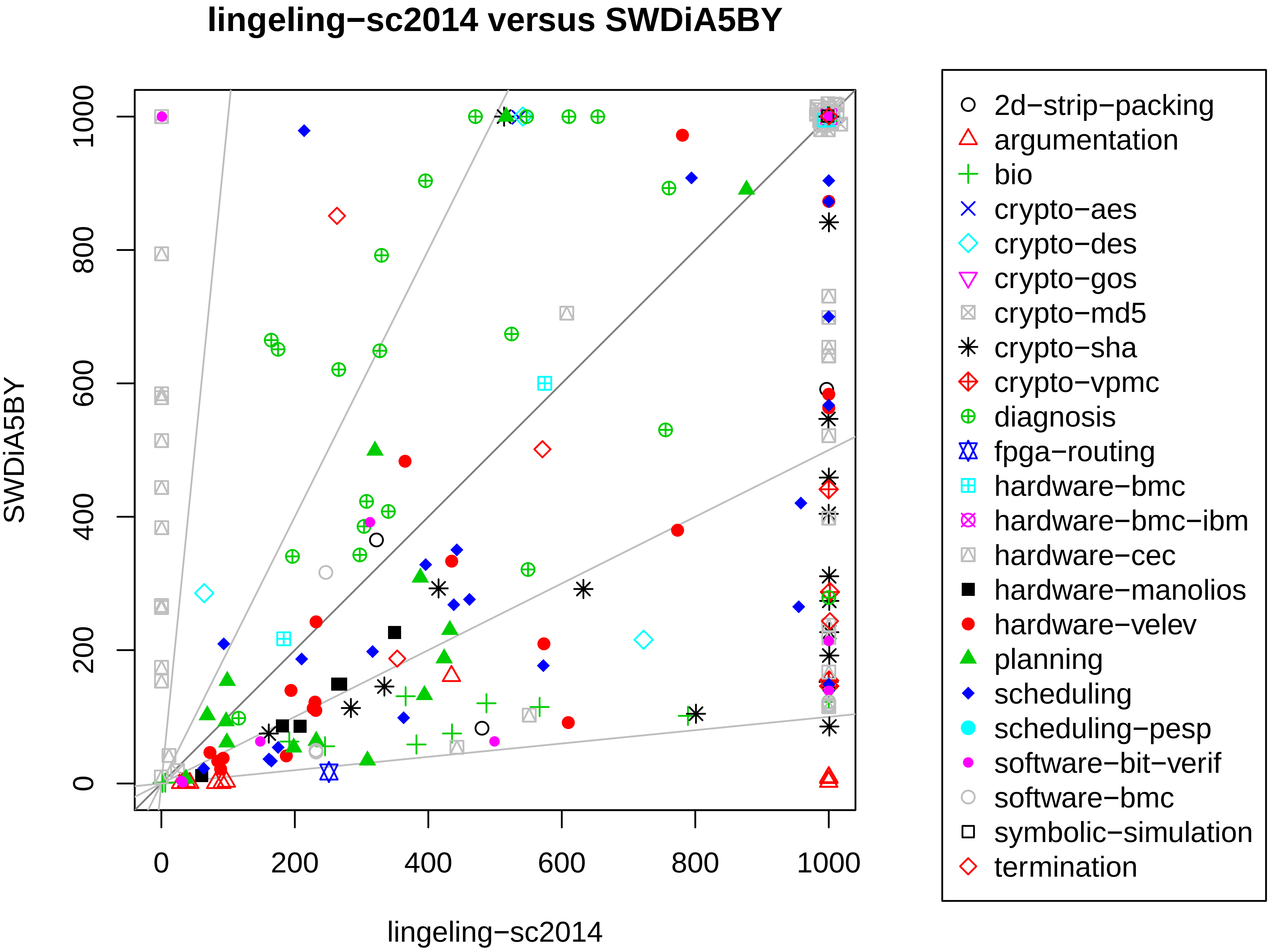
diagnosis (28), fpga-routing (1),

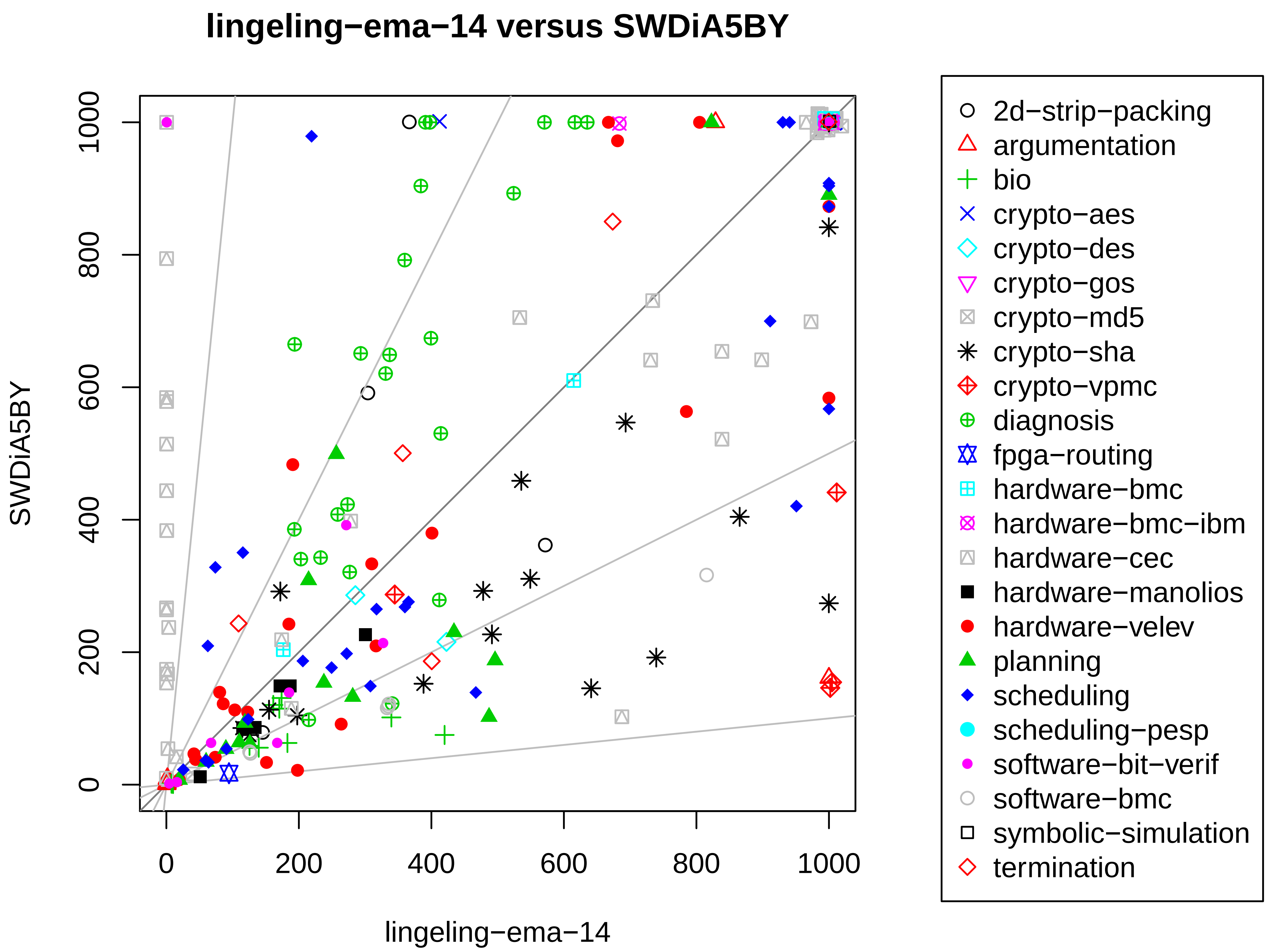
hardware-bmc (4), hardware-bmc-ibm (18), hardware-cec (30),
hardware-manolios (6), hardware-velev (27),

planning (19), scheduling (30), scheduling-pesp (3),

software-bit-verif (9), software-bmc (6), symbolic-simulation (1), termination (5)

in total **300** instances clustered in **23** buckets


lingeling−sc2014 versus SWDiA5BY
SWDiA5BY
lingeling−sc2014
0
200
400
600
800
1000
0
200
400
600
800
1000
○ 2d−strip−packing
△ argumentation
+ bio
× crypto−aes
◇ crypto−des
▽ crypto−gos
⊠ crypto−md5
✳ crypto−sha
◇ crypto−vpmc
⊕ diagnosis
✡ fpga−routing
⊞ hardware−bmc
⊗ hardware−bmc−ibm
△ hardware−cec
■ hardware−manolios
● hardware−velev
▲ planning
◆ scheduling
● scheduling−pesp
● software−bit−verif
○ software−bmc
□ symbolic−simulation
◇ termination


lingeling−ema−14 versus SWDiA5BY
SWDiA5BY
lingeling−ema−14
0
200
400
600
800
1000
0
200
400
600
800
1000
2d−strip−packing
argumentation
bio
crypto−aes
crypto−des
crypto−gos
crypto−md5
crypto−sha
crypto−vpmc
diagnosis
fpga−routing
hardware−bmc
hardware−bmc−ibm
hardware−cec
hardware−manolios
hardware−velev
planning
scheduling
scheduling−pesp
software−bit−verif
software−bmc
symbolic−simulation
termination

Restarts in CDCL

```
Status run_CDCL_loop_with_restarts () {
  for (;;) {
    if (bcp ()) {
           if (restarting ()) restart ();
      else if (!decide ()) return SATISFIABLE;
    } else {
      conflicts++;
      if (!analyze ()) return UNSATISFIABLE;
    }
  }
}
```

- run BCP and conflict analysis (including learning) until completion
- restart if restart policy implemented in `restarting` says so
- usually based on a global `conflicts` counter
- otherwise pick next decision (unless all are assigned)

Restart Scheme Examples

```
bool restarting () {
  return conflicts >= limit;
}

void static_uniform_restart () {
  restarts++;
  limit = conflicts + interval;
  backtrack (0);
}

void static_geometric_restart () {
  limit = conflicts + interval * pow (1.5, ++restarts);
  backtrack (0);
}

void luby_restart () {
  limit = conflicts + interval * luby (++restarts);
  backtrack (0);
}
```

Restart Scheme Classification

- **static schemes**
  - fixed schedule of restarts only based on conflicts counter
    - **uniform intervals**: wait a fixed number of conflicts after each restart
    - **non-uniform restart intervals**
      - number of performed restarts determines next interval (in terms of conflicts)
      - arithmetically or geometrically increasing actual interval
      - Luby scheme (also known as reluctant doubling)
      - inner-outer scheme
- **dynamic schemes**
  - agility based restart blocking
  - local restarts (not discussed in the paper nor the talk)
  - reusing the trail implicitly also blocks restarts (even partially)
  - Glucose restart scheme (focus here)

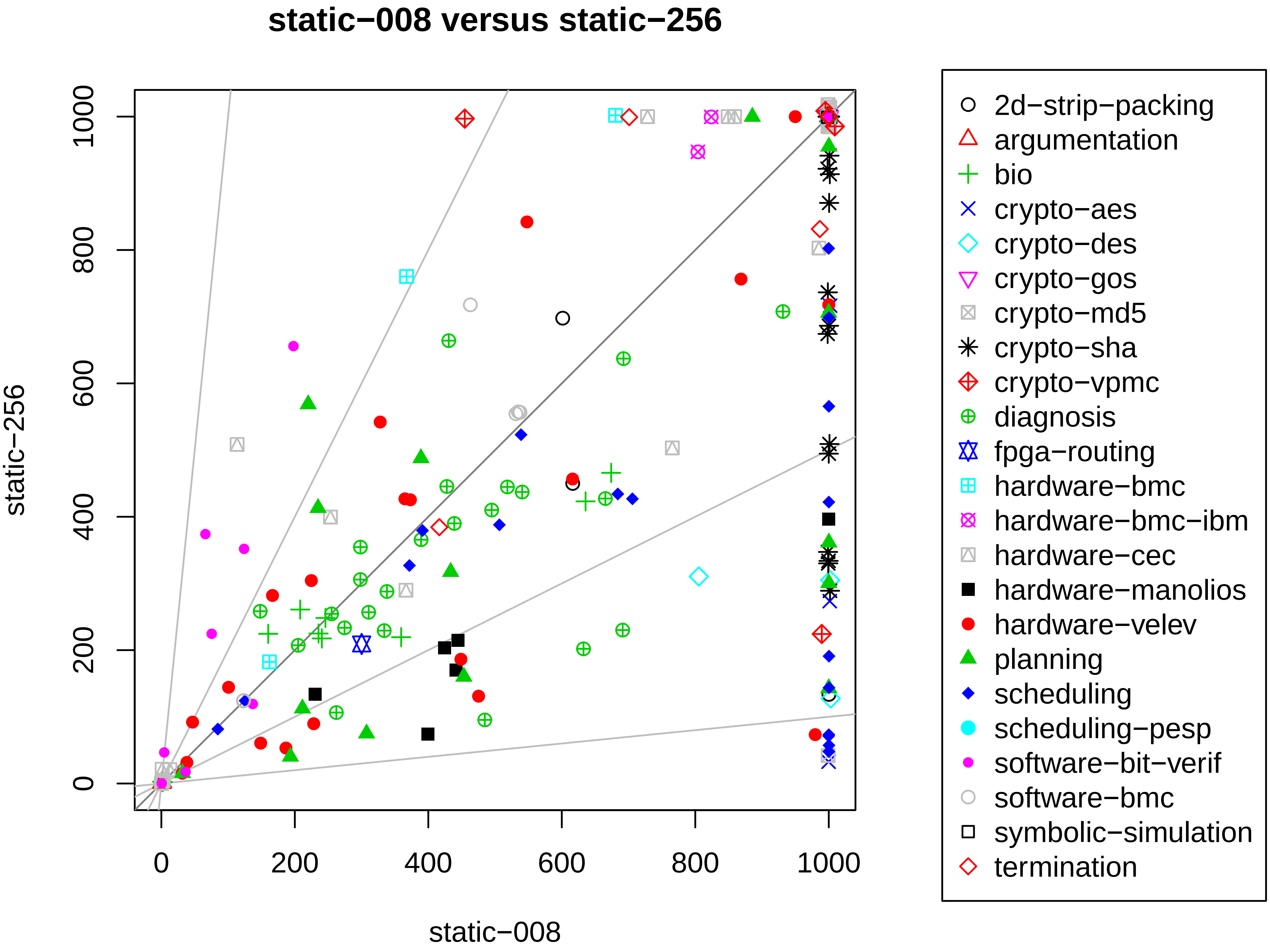
Comparing Static Uniform Restart Schemes

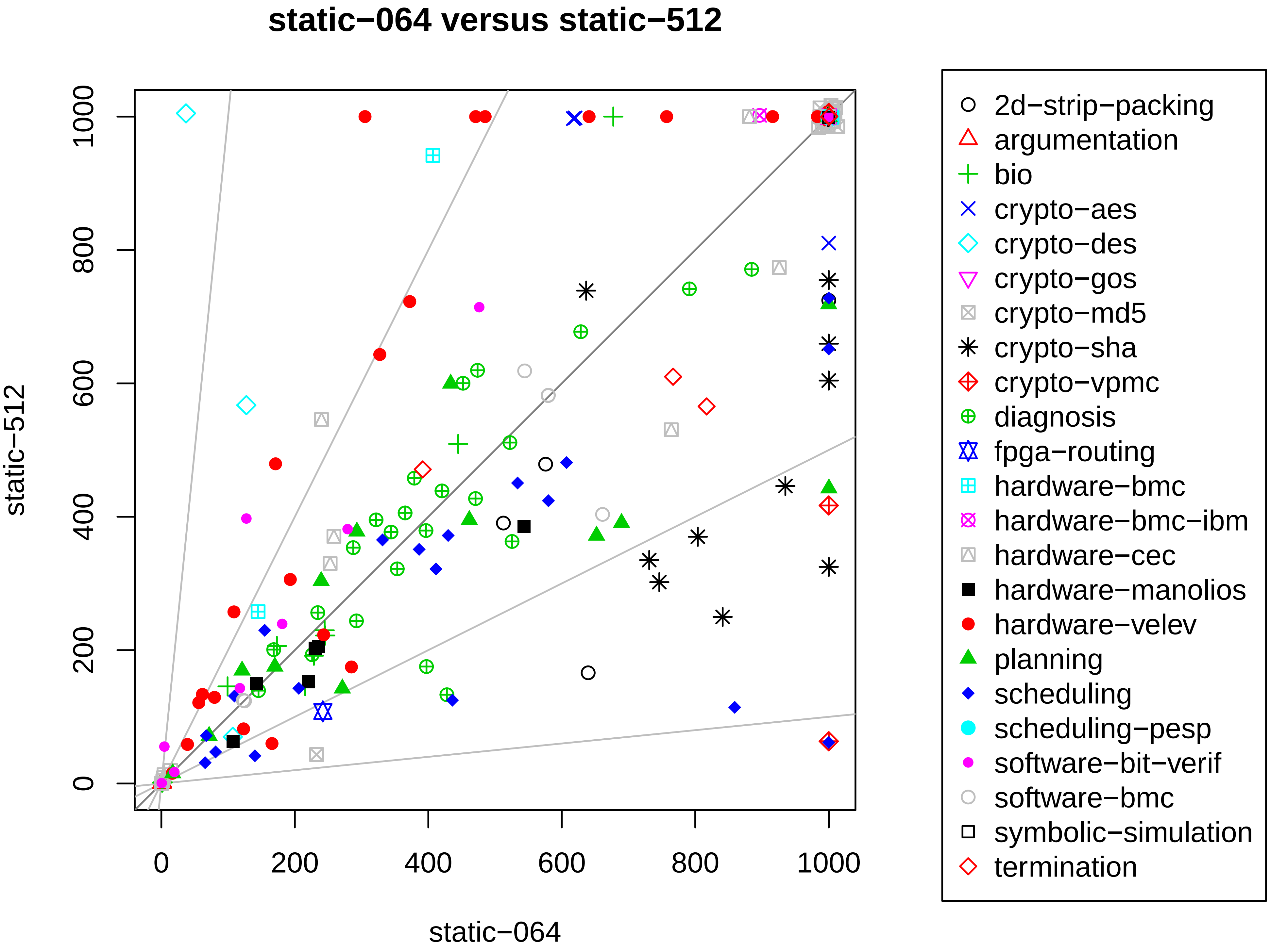
r	002	004	008	016	032	064	128	256	512
tot	122	127	139	144	144	161	163	**168**	158
sat	40	41	49	56	56	73	76	**83**	79
uns	82	86	**90**	88	88	88	87	85	79

r	002	004	008	016	032	064	128	256	512
2d-strip-packing	0/2	0/2	0/2	0/2	0/2	1/2	1/2	1/2	**2/2**
crypto-sha	0/0	0/0	0/0	0/0	1/0	7/0	11/0	**13/0**	10/0
hardware-cec	0/22	0/23	**0/24**	0/22	0/22	0/23	0/22	0/21	0/21
hardware-manolios	0/4	0/5	0/5	0/5	**0/6**	**0/6**	**0/6**	**0/6**	**0/6**
hardware-velev	5/9	6/10	8/11	8/12	8/12	**8/13**	8/12	8/11	8/6
planning	6/3	6/5	7/4	7/4	8/4	9/3	9/4	**11/4**	10/4
scheduling	1/7	0/7	1/7	4/7	6/7	9/7	9/7	11/7	**12/7**

SAT / UNSAT

underlined best


static−008 versus static−256
static−256
static−008
0
200
400
600
800
1000
0
200
400
600
800
1000
2d−strip−packing
argumentation
bio
crypto−aes
crypto−des
crypto−gos
crypto−md5
crypto−sha
crypto−vpmc
diagnosis
fpga−routing
hardware−bmc
hardware−bmc−ibm
hardware−cec
hardware−manolios
hardware−velev
planning
scheduling
scheduling−pesp
software−bit−verif
software−bmc
symbolic−simulation
termination


static−064 versus static−512
static−512
static−064
0
200
400
600
800
1000
0
200
400
600
800
1000
2d−strip−packing
argumentation
bio
crypto−aes
crypto−des
crypto−gos
crypto−md5
crypto−sha
crypto−vpmc
diagnosis
fpga−routing
hardware−bmc
hardware−bmc−ibm
hardware−cec
hardware−manolios
hardware−velev
planning
scheduling
scheduling−pesp
software−bit−verif
software−bmc
symbolic−simulation
termination

Glucose Restarts in a Nutshell

```
bool restarting () {
  return conflicts >= limit &&
     average_RECENT_lbds () > 1.25 * average_ALL_lbds ();
}

void glucose_restart () {  // same as static_uniform_restart
  restarts++;
  limit = conflicts + 50;
  backtrack (0);
}
```

- glucose level (LBD) of learned clause:
  - number of different decision levels in a learned clauses
  - calculated at the point the clause is learned during conflict analysis
- last 50 LBDs are stored and considered recent (explicit LBD queue)
- total average of all LBDs is simply `sum_lbd / conflicts`
- for discussion of blocking restarts since Glucose 2.1 see the paper

Simple and Cumulative Moving Averages

Glucose uses *simple moving average* (*SMA*) for the average of recent LBDs and *cumulative moving average* (*CMA*) for the the average of all LBDs and

simple $\mathit{SMA}(n,w) = \frac{1}{w} \cdot (t_n + t_{n-1} + \ldots + t_{n-w+1})$ with $n \geq w \geq 1$

cumulative $\mathit{CMA}(n) = \mathit{SMA}(n,n)$

$$\mathit{CMA}(n) = \mathit{CMA}(n-1) + \frac{t_n - \mathit{CMA}(n-1)}{n}$$

$$\mathit{SMA}(n,w) = \mathit{SMA}(n-1,w) + \frac{t_n}{w} - \frac{t_{n-w}}{w}$$

requires $\mathit{SMA}(n,50) > 1.25 \cdot \mathit{CMA}(n)$ to restart

and 50 conflicts have passed

Exponential Moving Average

we suggest to use *EMA*s instead of the “fast” *SMA* and/or “slow” *CMA*

exponential $\mathit{EMA}(n,\alpha) = \alpha \cdot t_n + (1-\alpha)\cdot \mathit{EMA}(n-1,\alpha)$ with $0 < \alpha < 1$ $\quad a \approx \frac{2}{1+w}$

current estimate

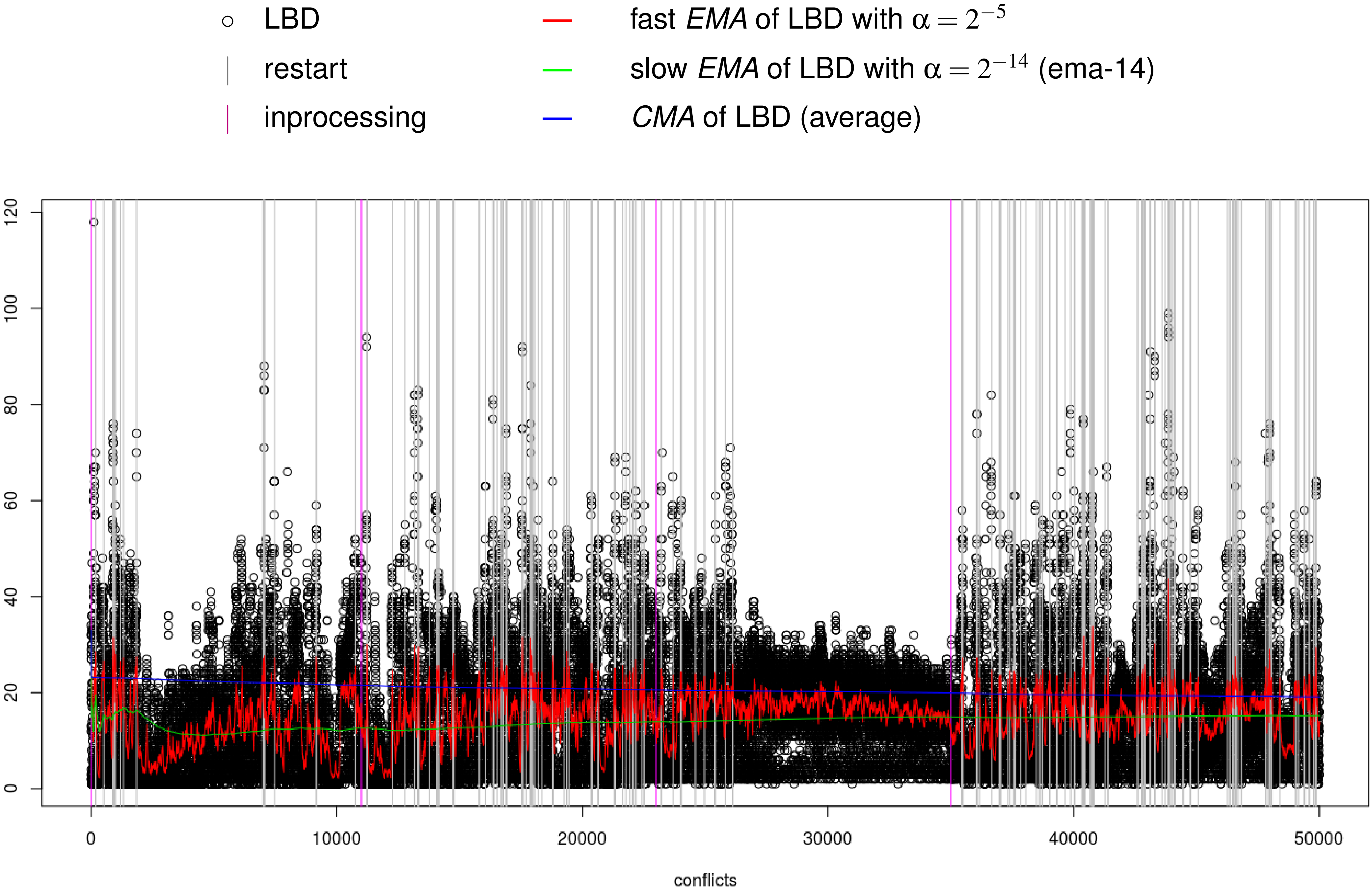
alternative $\underbrace{\mathit{EMA}(n,\alpha)}_{\text{next estimate}} = \mathit{EMA}(n-1,\alpha) + \alpha \cdot (t_n - \mathit{EMA}(n-1,\alpha))$

difference/error

to restart version average requires $\mathit{EMA}(n,2^{-5}) > 1.25 \cdot \mathit{CMA}(n)$

to restart version ema-14 requires $\mathit{EMA}(n,2^{-5}) > 1.25 \cdot \mathit{EMA}(n,2^{-14})$

and again in both cases that a certain number of conflicts say 50 have passed

○ LBD
restart
inprocessing
fast *EMA* of LBD with $\alpha = 2^{-5}$
slow *EMA* of LBD with $\alpha = 2^{-14}$ (ema-14)
CMA of LBD (average)
0
20
40
60
80
100
120
0
10000
20000
30000
40000
50000
conflicts

Comparing EMAs with SMA and CMA

solver	Glucose 4.0			Lingeling ba2							
restarts	*ss*	*es*	*ee*	*avg*	*e8*	*e10*	*e12*	*e14*	*e16*	*e18*	*e20*
tot	163	163	165	178	167	170	180	**181**	180	177	171
sat	72	73	76	83	80	78	**86**	**86**	**86**	82	77
uns	91	90	89	**95**	87	92	94	**95**	94	**95**	94
avgc	192	166	167	145	230	204	195	186	172	147	108

Glucose 4.0 column *ss* correspond to the original Glucose version

column *es* to adding EMAs for only forcing restarts

column *ee* includes using EMA for blocking restarts too

column avg is Lingeling version *average* of Glucose version *ee*

columns eX correspond to Lingeling versions *ema*-X
using a slow EMA with $\alpha = 2^{-X}$ instead of CMA

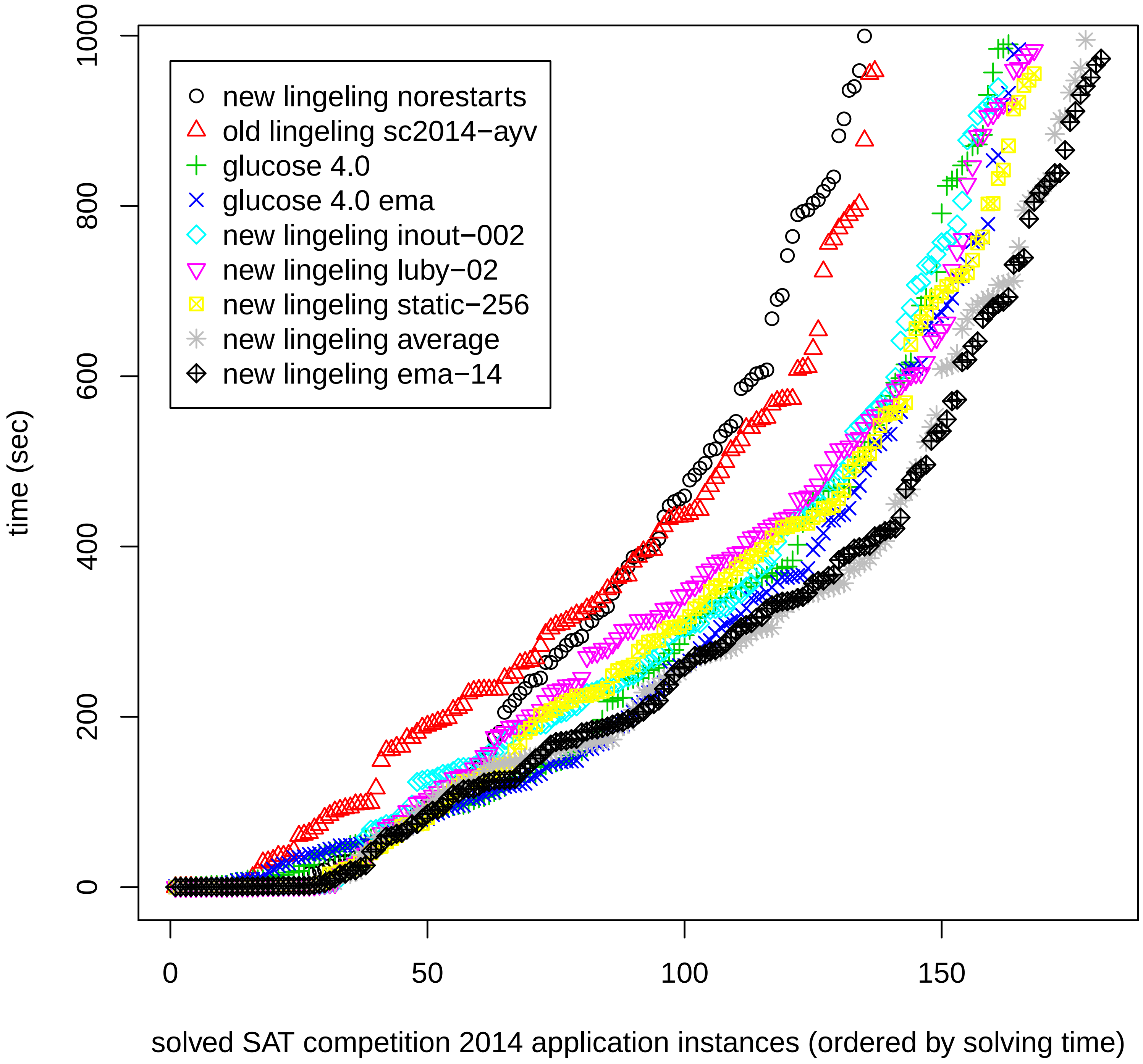
Floating Point Implementation

```
double fast, slow;
...

bool analyze () {
  int lbd;
  ...
  slow += (lbd - slow)/(double)(1<<14);
  fast += (lbd - fast)/(double)(1<<5);
  ...
}

bool restarting () {
  return conflicts > limit && fast > 1.25 * slow;
}
```

Conclusion Part II

- data and source: http://fmv.jku.at/evalrestart/evalrestart.7z
- optimal restart interval varies with benchmark bucket
  - for miters fast restarts essential
  - for crypto benchmarks longer intervals necessary
  - disabling restarts completely is bad
  - Glucose restarts superior to Luby style
- presented an EMA variant of the Glucose restart scheme
  - simpler model, simpler to implement
  - similar performance (slightly faster)
- future work
  - how to improve blocking of restarts
  - restart intervals still not optimal: really need machine learning?
  - combined SAT and Stock Market Analysis or SAT and Reinforcement Learning

originally proposed title for the POS'15 paper

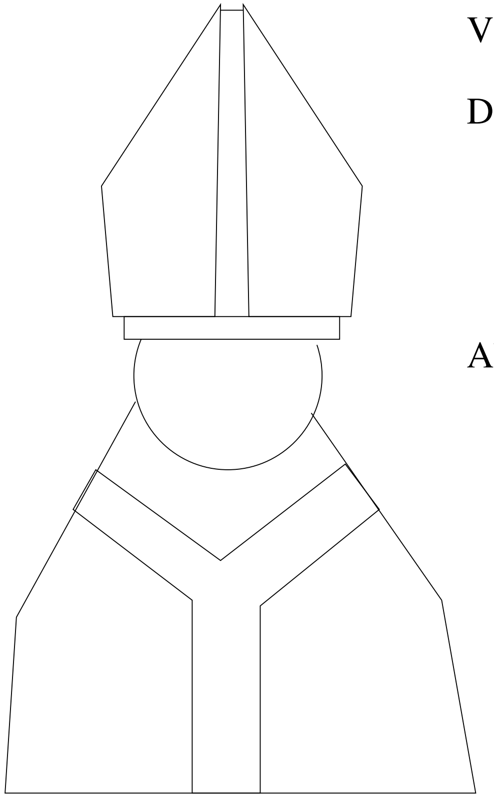


○ new lingeling norestarts
△ old lingeling sc2014−ayv
+ glucose 4.0
× glucose 4.0 ema
◇ new lingeling inout−002
▽ new lingeling luby−02
⊠ new lingeling static−256
✳ new lingeling average
◈ new lingeling ema−14
time (sec)
0
200
400
600
800
1000
0
50
100
150
solved SAT competition 2014 application instances (ordered by solving time)

- Evaluating CDCL Variable Scoring Schemes [SAT’15 paper with Andreas Fröhlich]
  - continued discussion from our Banff and Dagstuhl workshops
  - shows that VMTF **is** as good as VSIDS (and explains boths)
- Evaluating CDCL Restart Schemes [POS’15 paper with Andreas Fröhlich]
  - simplifies Glucose style restart schemes
  - evaluation shows clear benefits but also weaknesses
- Weaknesses of CDCL for Equivalence Checking (miters)
  - CDCL has a hard time to learn the right clauses
  - fast restarts important for miters
- Arithmetic Reasoning Hard for CDCL

Miter

ICCAD’93

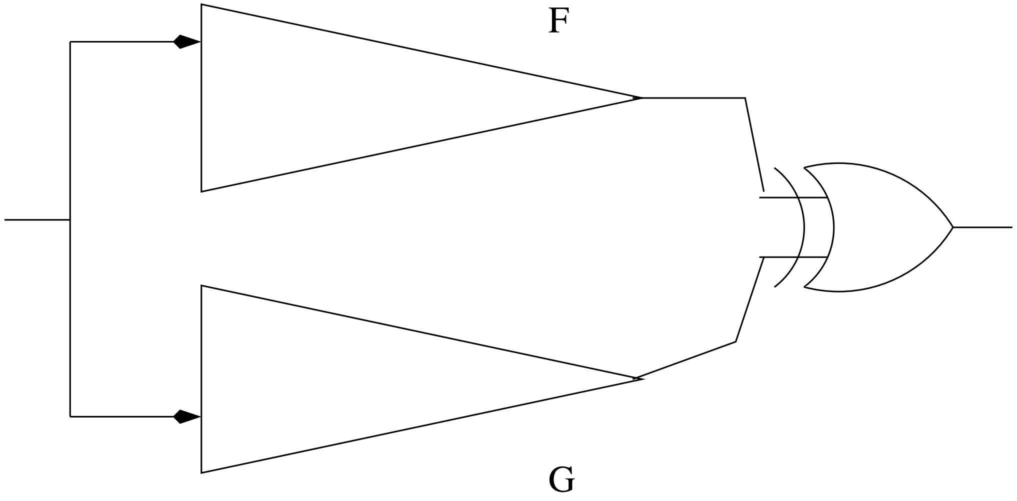


ICCAD’93

Verification of Large Synthesized Designs

Daniel Brand

Abstract ....



Structural Hashing and Hyper-Binary Resolution

[HeuleJärvisaloBiere-CPAIOR'13]

$$\frac{a \leftrightarrow x \land y \qquad b \leftrightarrow x \land y}{a \leftrightarrow b}$$

$$(\bar{a} \lor x)(\bar{a} \lor y)(a \lor \bar{x} \lor \bar{y})(\bar{b} \lor x)(\bar{b} \lor y)(b \lor \bar{x} \lor \bar{y})$$

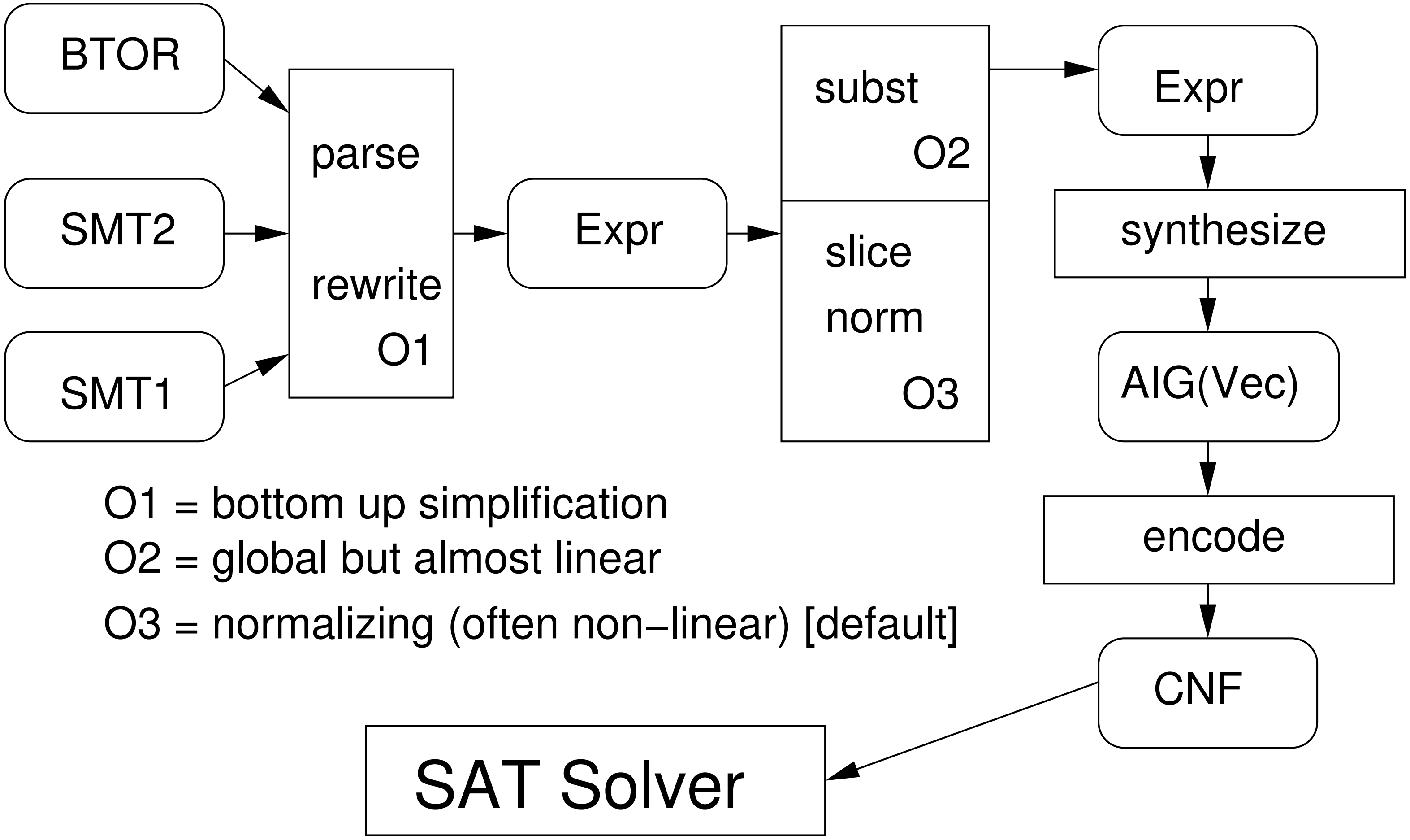
hyper-binary resolve multiple binary clauses in "parallel":

$$\frac{\bar{a} \lor x \quad \bar{a} \lor y \quad b \lor \bar{x} \lor \bar{y}}{\bar{a} \lor b} \qquad \frac{\bar{b} \lor x \quad \bar{b} \lor y \quad a \lor \bar{x} \lor \bar{y}}{a \lor \bar{b}}$$

thus "in principle" hyper-binary resolution can simulate structural hashing

but we do not know how to implement it fast
(without having the circuit and including equivalence literal substitution)

Software Architecture of our Boolector SMT Solver



O1 = bottom up simplification
O2 = global but almost linear
O3 = normalizing (often non−linear) [default]

Lingeling / PicoSAT / MiniSAT

Solving Miters with CDCL Requires Restarts

- trivial miters of identical circuits
  - equivalence = bimplication (two implications aka binary clauses)
  - equivalences are reused recursively and thus order of learning important
  - best solved by “finding” and applying equivalences bottom-up
- point of Yakau Novikov at our predecessor Dagstuhl meeting in 2015
  - assume first implication $a \rightarrow b$ is learned
  - w.l.o.g. a assigned before b
  - then a was assigned to 1 and b to 0
  - after learning first implication, b is flipped and both a and b are assigned to 1
  - in order to learn $b \rightarrow a$ we have to assign b to 1 and a to 0
  - thus without unassigning a we can not learn the second implication
- so frequent restarts are useful here
  - triggered by Glucose restart style schedules (as discused in Part II)
  - actually phase saving is also counter productive here (tried to fix it without success)
  - still not perfect, dedicated preprocessing faster

Challenges Part III

- how to do equivalence checking on the CNF level
  - (even) more efficient implementation of hyper-binary resolution
  - partial solution [HeuleBiere-LPAR'13]
    - blocked clause decomposition
    - SAT sweeping
  - another (unpublished) partial solution:
    - simple-probing in Lingeling
    - simulates structural hashing on the CNF level
    - eager equivalent literal substitution
    - fails after preprocessing (bounded variable elimination)
- recover / use circuit structure
  - locality
  - direction (inputs)
  - functional dependencies (might get lost during preprocessing)

Part IV

- Evaluating CDCL Variable Scoring Schemes [SAT'15 paper with Andreas Fröhlich]
  - continued discussion from our Banff and Dagstuhl workshops
  - shows that VMTF **is** as good as VSIDS (and explains boths)
- Evaluating CDCL Restart Schemes [POS'15 paper with Andreas Fröhlich]
  - simplifies Glucose style restart schemes
  - evaluation shows clear benefits but also weaknesses
- Weaknesses of CDCL for Equivalence Checking (miters)
  - CDCL has a hard time to learn the right clauses
  - fast restarts important for miters
- Arithmetic Reasoning Hard for CDCL

Commutativity of Bit-Vector Multiplication

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 12))
(declare-fun y () (_ BitVec 12))
(assert (distinct (bvmul x y) (bvmul y x)))
(check-sat)
```

			12 core	
	1 core	1 core	cube-and-conquer	12 core
bits	Glucose	Lingeling	March|iLingeling	Treengeling
01	0.00	0.00	0.00	0.01
02	0.00	0.00	0.00	0.01
03	0.00	0.00	0.00	0.01
04	0.00	0.00	0.02	0.03
05	0.00	0.01	0.05	0.13
06	0.02	0.03	0.36	0.31
07	0.14	0.27	0.63	0.72
08	1.18	1.98	1.38	2.47
09	7.85	10.98	2.63	4.65
10	37.16	41.49	5.02	10.86
11	147.62	214.98	15.72	21.96
12	833.62	649.49	56.57	61.48
13	--	--	238.10	263.44

limit of 900 seconds wall clock time

Arithmetic Circuit Equivalence Checking

- secret of the success of (combinational) equivalence checking
  - assumption: many internal equivalence points
  - makes BDD and SAT sweeping effective
- problems with arithmetic circuits
  - almost no equivalent internal signals (except for outputs)
  - proof complexity conjectured to be beyond resolution
  - often no ”clean” implementation circuit available
- challenges
  - prove conjectured complexity
  - use world-level (bit-vector) information
  - arithmetic reasoning on the bit-level
  - robust integration in SAT and/or SMT solver
- started to collect a large number of such benchmarks: http://fmv.jku.at/datapath

Challenge Part IV

Problem

Tseitin encoding of the miter of a multiplier with itself but with inputs swapped.

Conjecture

Refuting the resulting CNF requires exponential resolution proofs (and thus CDCL too).

Research Question

Determine proof systems with polynomial proofs for this problem.

candidate proof systems:

polynomial ring reasoning in $\mathbb{B}[X]/\langle F\rangle$ or $\mathbb{Z}[X]/\langle F\rangle$ with $F = \{x^2 - x \mid x \in X\}$

see our benchmark description in SAT'16 Competition proceedings for more references

Overall Conclusion

- decision heuristics
  - new empirical insights: VMTF simpler and as good as VSIDS
  - can we prove why these work?
- restarts
  - simplified Glucose restart scheme, showed that it somehow works
  - clearly not where we want it to be (machine learning necessary?)
- miters
  - CDCL implementations do not learn the right clauses
  - fast restarts partially fix it, can we prove that?
- arithmetic reasoning
  - need stronger proof systems?
  - can we prove that?