Splatz SAT Solver

Armin Biere
Johannes Kepler Universität Linz

Pragmatics of SAT 2016

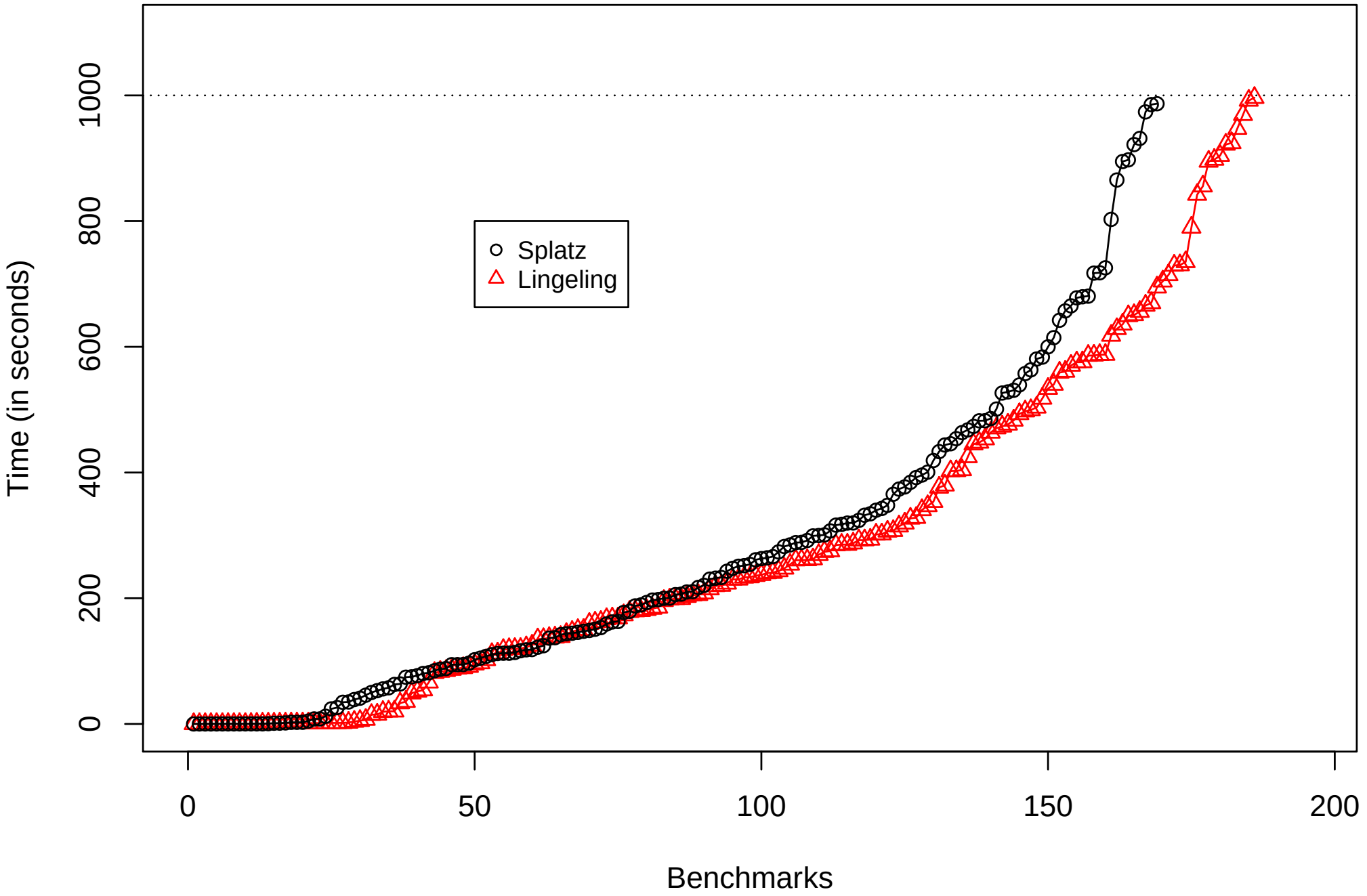
POS'16

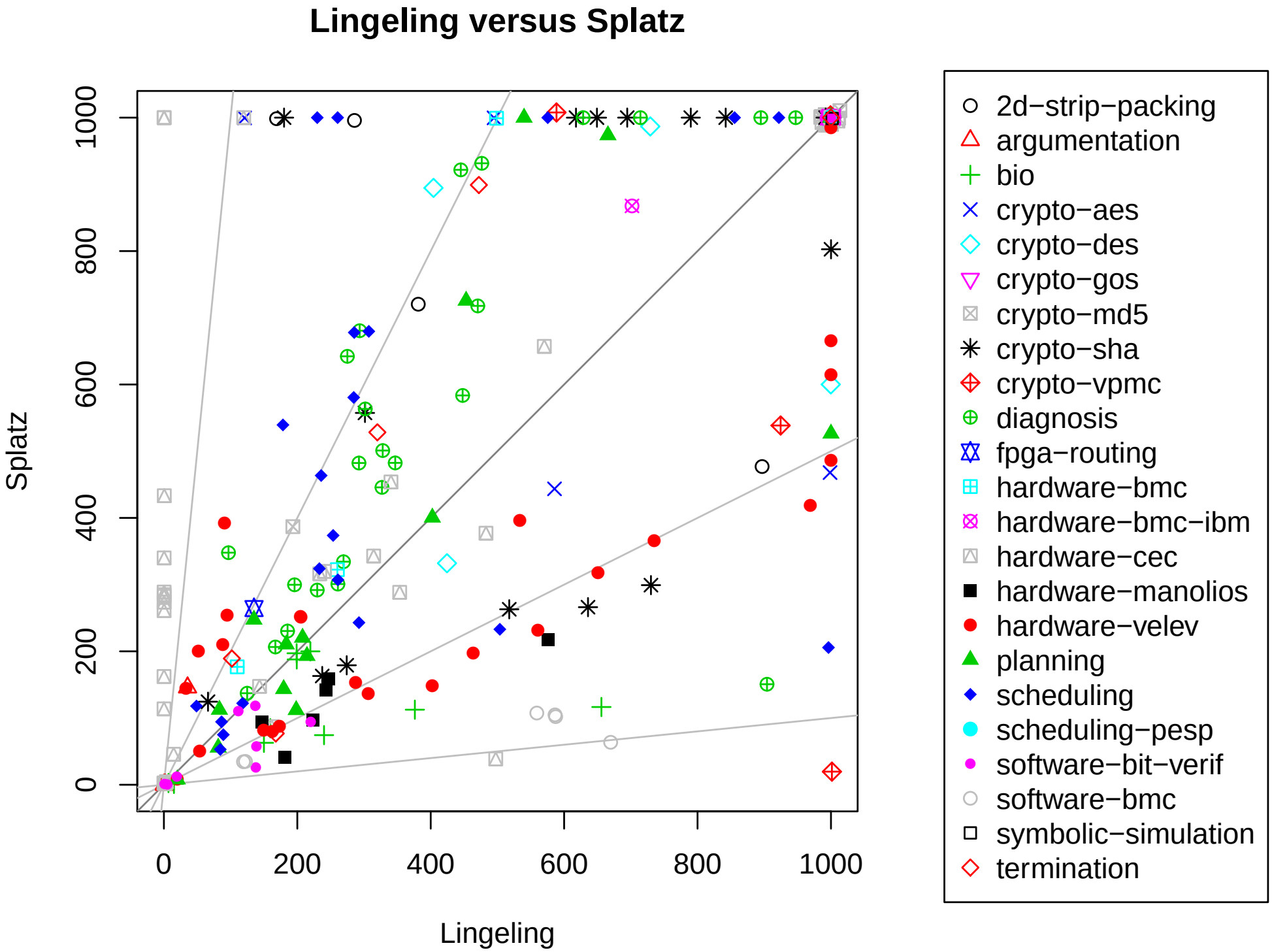
Université de Bordeaux
Bordeaux, France

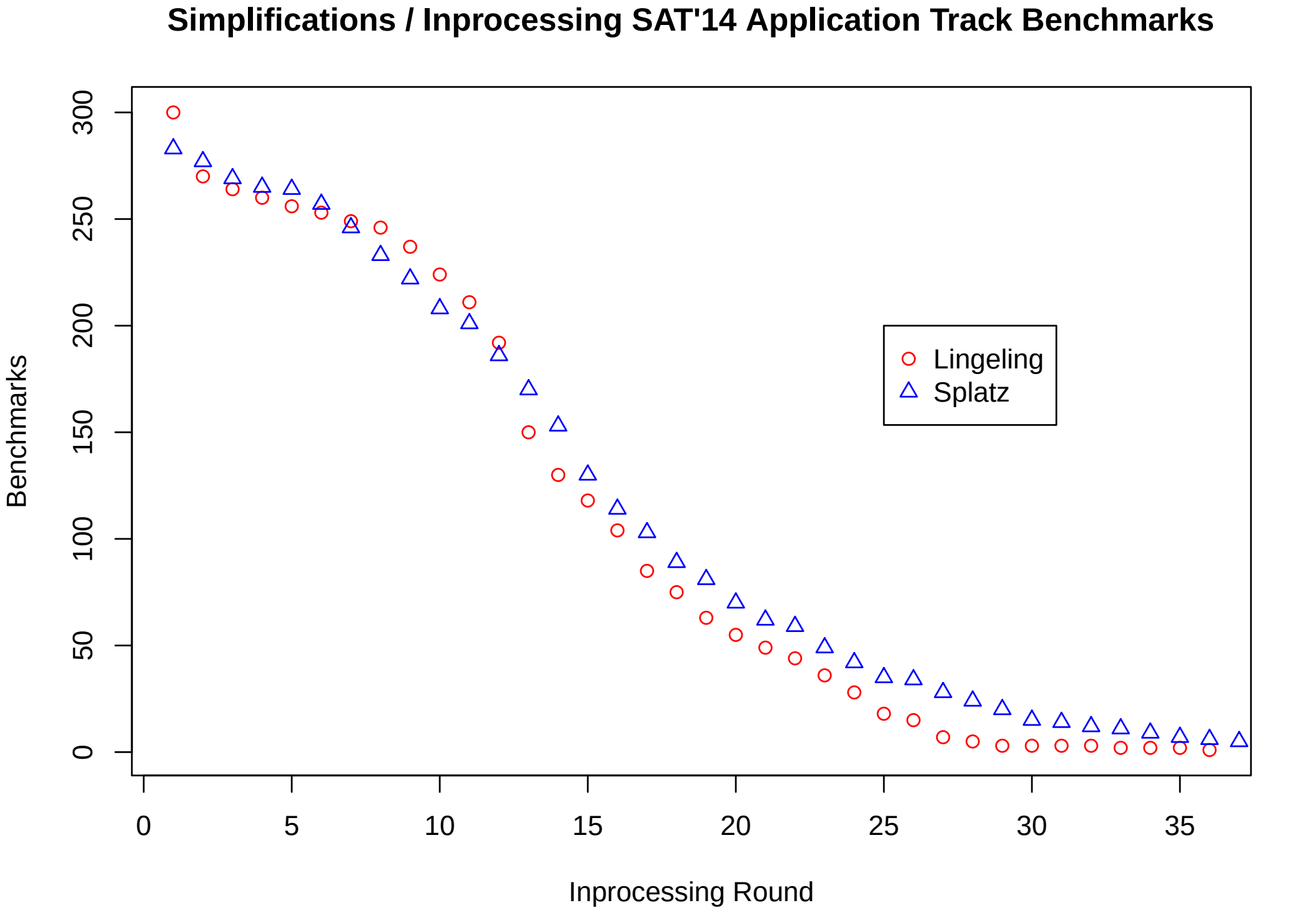
Monday, 4th July, 2016

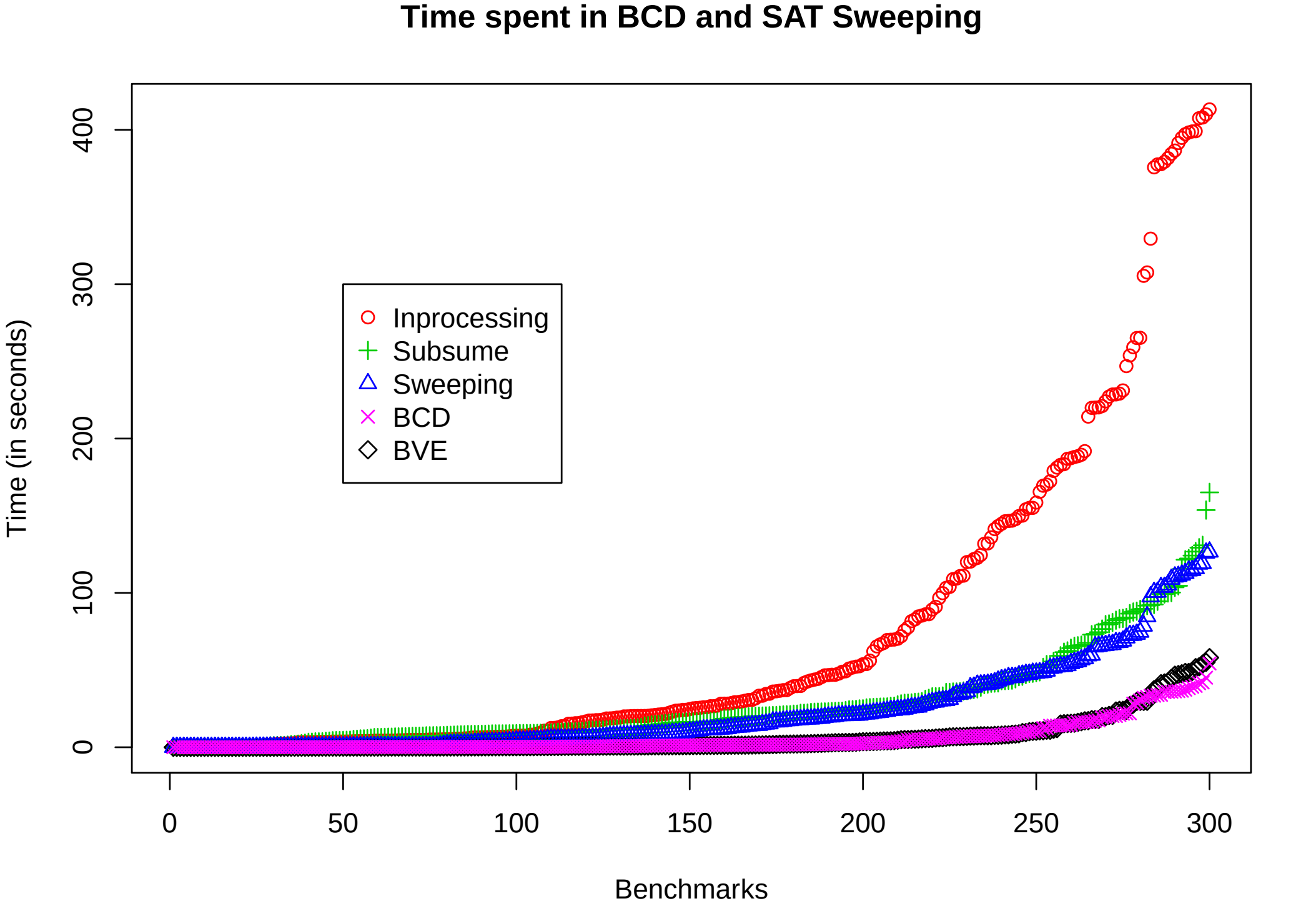
Motivation

- What do you really want to have in a SAT solver? Niklas asked in Austin ...
  - common impression: Lingeling has too many stuff implemented
  - tuned to existing benchmarks, reached **local minimum**
  - implementing / tuning / debugging takes time and is error prone
  - hard to figure what is really important and **hard to evaluate new ideas**
- Restart to figure out ...
  - painful, since Lingeling is good on current benchmarks
  - taking away features (moving away from local minimum) solves less instances
  - but chance for **simplifying design** based on **new insights**
    - Glucose style restarts with exponential smoothing averages [POS'15]
    - using variable move to front (VMTF) instead of VSIDS [SAT'15]
  - experimenting with certain ideas is very hard to implement within Lingeling
    - **inprocessing of SAT sweeping** + blocked clause decomposition (BCD) [LPAR'13]
    - new **subsumption** algorithm (on **learned clauses too**)


SAT'14 Competition Application Track Instances
Time (in seconds)
0
200
400
600
800
1000
0
50
100
150
200
Benchmarks
Splatz
Lingeling


Lingeling versus Splatz
Splatz
Lingeling
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


Simplifications / Inprocessing SAT'14 Application Track Benchmarks
Benchmarks
0
50
100
150
200
250
300
0
5
10
15
20
25
30
35
Inprocessing Round
Lingeling
Splatz


Time spent in BCD and SAT Sweeping
Time (in seconds)
0
100
200
300
400
0
50
100
150
200
250
300
Benchmarks
Inprocessing
Subsume
Sweeping
BCD
BVE

SAT Sweeping and Blocked Clause Decomposition

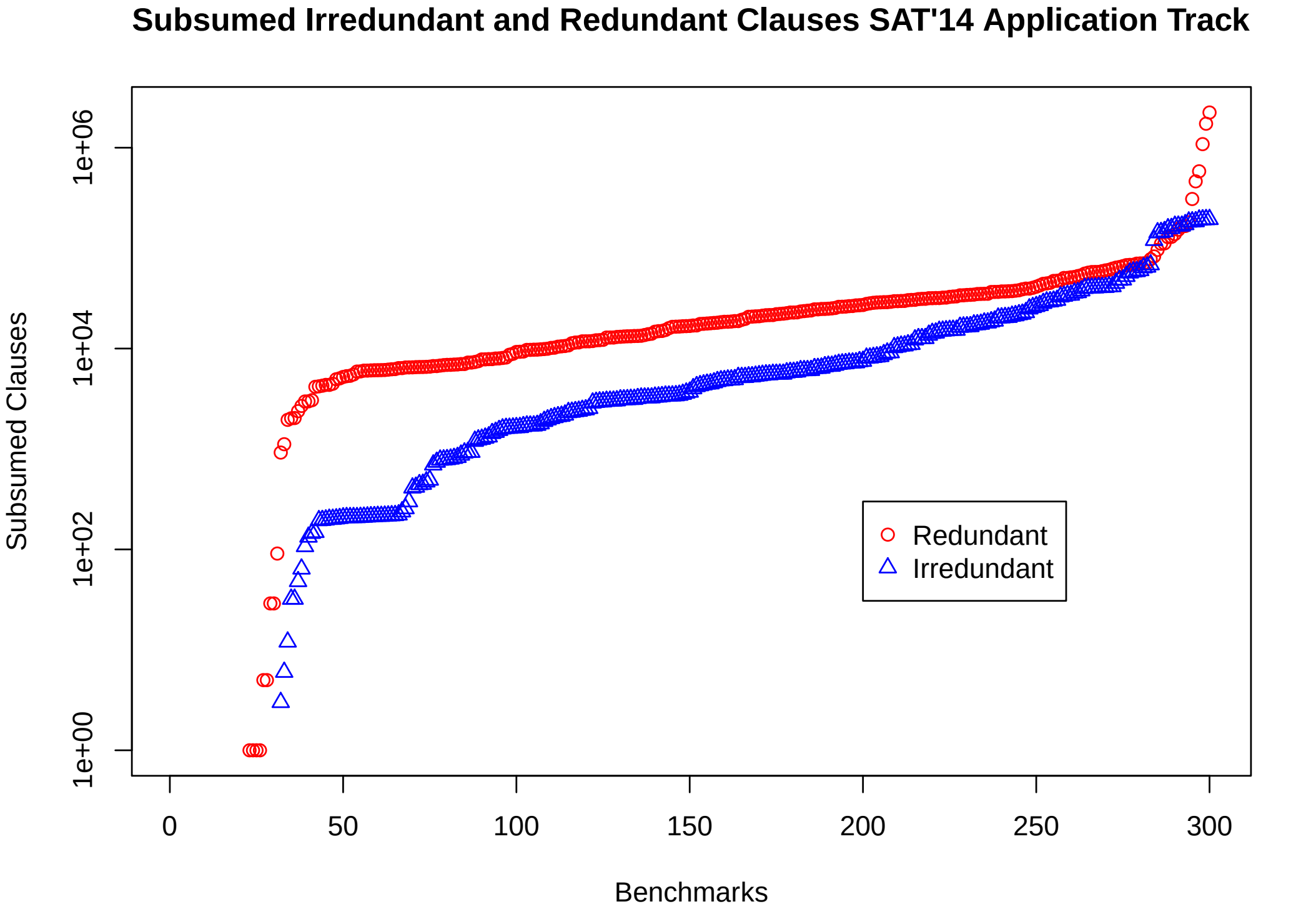
- Simulate structural SAT sweeping in CNF [LPAR'13]
  - uses blocked clause decomposition (BCD) instead of structural knowledge
  - blocked part of BCD acts as circuit (e.g., can be simulated)
  - goal is to find **backbone** variables and **equivalences**
  - relies on effectiveness of BCD (goal is highly unbalanced BCD)
- Inprocessing version interleaved with CDCL search
  - original experiments in preprocessing mode
  - inprocessing can take learned facts into account
- Inprocessing Results mixed
  - does not allow to *effectively* simulate "simple probing" in Lingeling
  - 17,339 backbones and 39,696 equivalences through sweeping
    - while 3,897,113 ELS and 425,098 Failed Literals
  - difficult to find and tune good decomposition algorithms:
    - linear (70% / 54%), pure-linear (72% / 69%), pure-inverse (71% / 70%)
  - circuit structure for effective BCD is partially lost (in inprocessing)

Issues with Subsumption

- SATeLite style subsumption:
  - interleave bounded variable style elimination (BVE) ...
  - ... with backward subsumption:
    - go over all clauses C
    - try to find clause D with $C \subseteq D$
    - full occurrences, walk occurrence list of literal in with smallest entries
    - also tries to strengthen clauses
  - quite expensive if not bounded (number of occurrences checked)
  - particularly full occurrence lists prohibit use for learned clauses
- Glucose keeps low glue learned clauses forever (even if subsumed)
  - MiniSAT just automatically discards them due to low activity
  - small learned clauses might subsume or strengthen even irredundant clauses
  - would be good to include subsumption checking on and with learned clauses too
- new subsumption algorithm inspired by [BayardoPanda'11]

New Subsumption Algorithm

- as in SATeLite, MiniSAT, Glucose with BVE in phases
- considers learned clauses as subsumed and subsuming clauses too
- forward subsumption checking only needs one watch per clause
- smallest clauses are checked for being subsumed first
- literals in clauses sorted by number of occurrences
- go over all other clauses in watch lists of literals in candidate subsumed clause
  - mark literals in candidate subsumed clause
  - first case: other clause *same size*
  - second case: other clause *smaller* then check all literals in it marked
  - third case: other clause *larger* then use merge sort style check
- still can become costly and has to be limited
  - comparable in speed to the actual BVE phase
  - fast enough to be called once in a geometric schedule on learned clauses


Subsumed Irredundant and Redundant Clauses SAT'14 Application Track
Subsumed Clauses
1e+00
1e+02
1e+04
1e+06
0
50
100
150
200
250
300
Benchmarks
Redundant
Irredundant

Features (I)

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Thank you, Norbert & Mate!

Features (II)

- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- inprocessing blocked clause decomposition (BCD) enabling ...
- ... inprocessing SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- **dynamic sticky clause reduction**
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse