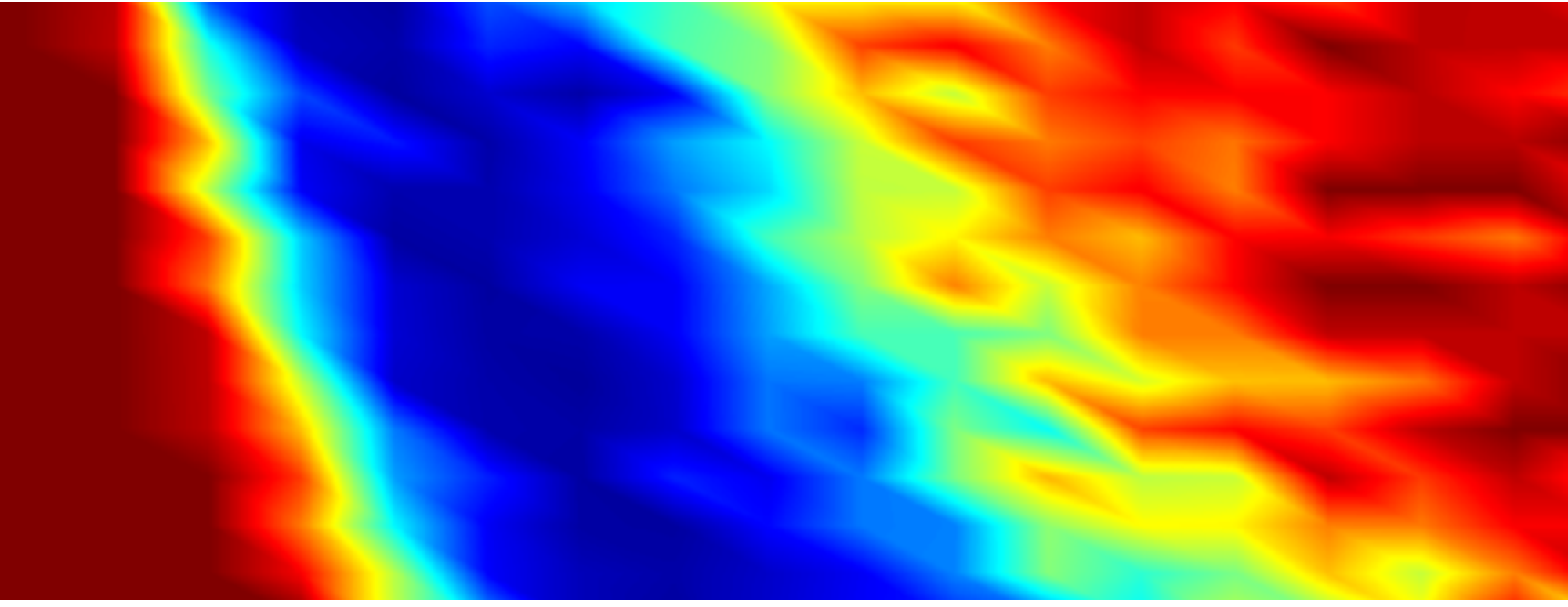

Adrian Balint[1] Armin Biere[2]
Andreas Fröhlich[2] Uwe Schöning[1]
July 16, 2014

Improving implementation of SLS solvers for SAT and new heuristics for *k*-SAT with long clauses

Contribution of this work

1. new implementation techniques in SLS solvers
   - inspired from the CDCL solver NanoSAT
   - **XOR scheme**
2. new decision heuristic for the *probSAT* SLS SAT solver
   - use of the **multilevel break value**
3. analysis of **selection policies** of unsatisfied clauses in SLS solvers

Introduction - SLS Solvers

SLS Solvers

- work on complete assignments as opposed to partial assignments
- start with a randomly generated assignment
- selects a variable according to some heuristic (*pickVar*) and then changes its value (*flip(var)*)
- process is repeated until a solution has been found

SLS Solver Complexity

- dominated by the complexity of the *pickVar* and *flip*(*var*) procedures
- complexity of *pickVar* depends on the heuristic used by the solver
- complexity of *flip*(*var*) depends on the information computed

SLS Implementations

- SLS solvers track the satisfaction level of clauses
  - a t-satisfied clause has t true literals
- most SLS heuristics use the *make* and *break* value of variables
- the **break value** counts the transitions of clauses
  - **from 1-satisfied $\rightarrow$ 0-satisfied**
- the **make value** counts the transitions
  - **from 0-satisfied $\rightarrow$ 1-satisfied**
- consequently SLS solvers have to monitor
  - 0-satisfied and 1-satisfied clauses
  - their transitions within the *flipVar*(*var*) method

1. SLS Implementation Improvement

- for each 1-satisfied clause the critical variable has to be stored
- the transition $2 \rightarrow 1$ is the only one with complexity $O(len(C))$
- the remaining transitions have complexity $O(1)$

Key idea

instead of storing the critical variable of a clause store the *XOR* concatenation of all satisfied literals

Advantage

the critical variable can be obtained in $O(1)$

Example:

- two satisfying variables in a clause C, i.e. $trueVarX[C] = x_i \oplus x_j$
- variable x_i is being flipped
- the critical variable can be obtained with one single operation: $x_j = x_i \oplus trueVarX[C]$

Algorithm 1: Variable flip with XOR caching

Input: Variable to flip v

```
1  α[v] = ¬α[v];                                          /* change variable value */
2  satisfyingLiteral = α[v] ? v : ¬v;
3  falsifyingLiteral = α[v] ? ¬v : v;
4  for clause in occurrenceList[satisfyingLiteral] do
5      if numTrueLit[clause] == 0 then                    /* transition 0 → 1 */
6          remove clause from falseClause ;
7          break[v]++;
8          trueVarX[clause] = 0
9      else
10         if numTrueLit[clause] == 1 then                /* transition 1 → 2 */
11             break[trueVarX[clause]]--;
12     numTrueLit[clause]++;
13     trueVarX[clause] ⊕= v;
14 for clause in occurrenceList[falsifyingLiteral] do
15     trueVarX[clause] ⊕= v;
16     if numTrueLit[clause] == 1 then                    /* transition 0 ← 1 */
17         add clause to falseClause ;
18         break[v]--;
19     else
20         if numTrueLit[clause] == 2 then                /* transition 1 ← 2 */
21             break[trueVarX[clause]]++;
22     numTrueLit[clause]--;
```

Empirical Evaluation - probSAT - SC12 Random

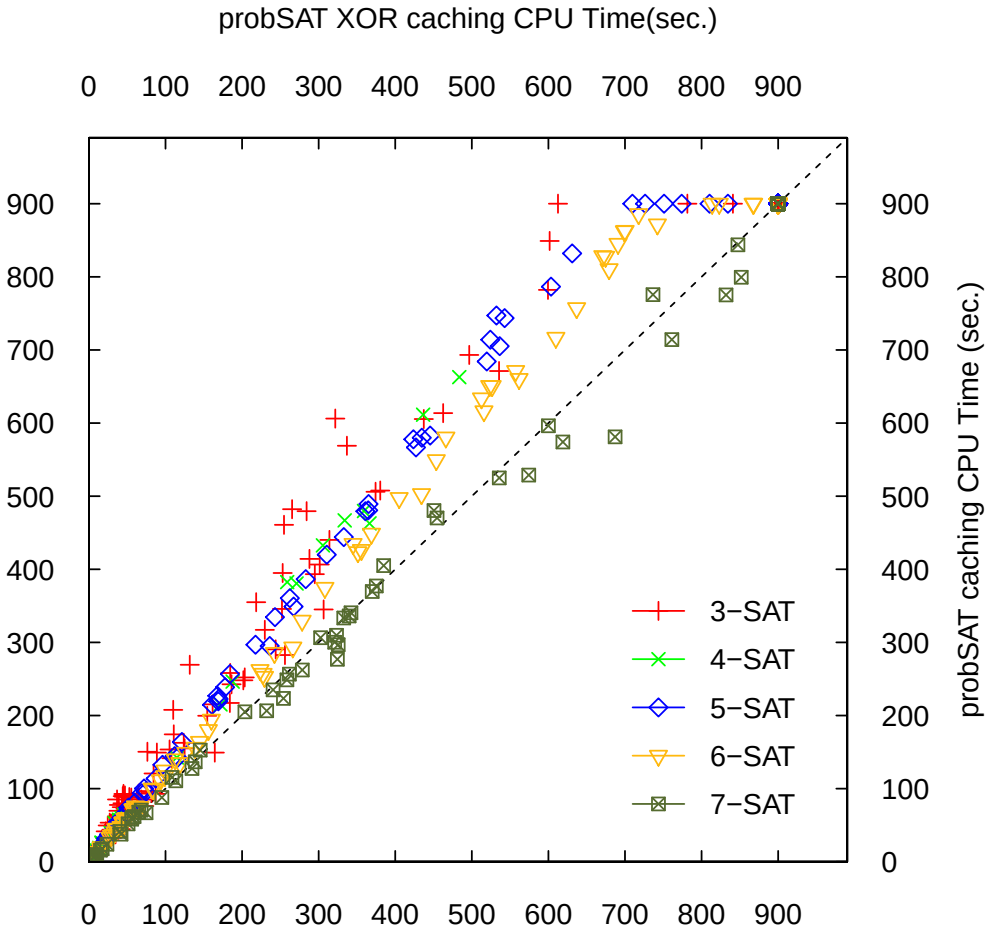

Figure: probSAT solver with XOR scheme versus default implementation on the SAT Challenge 2012 random instances with a cutoff of 900 seconds.

Empirical Evaluation - Sparrow - SC12 Random

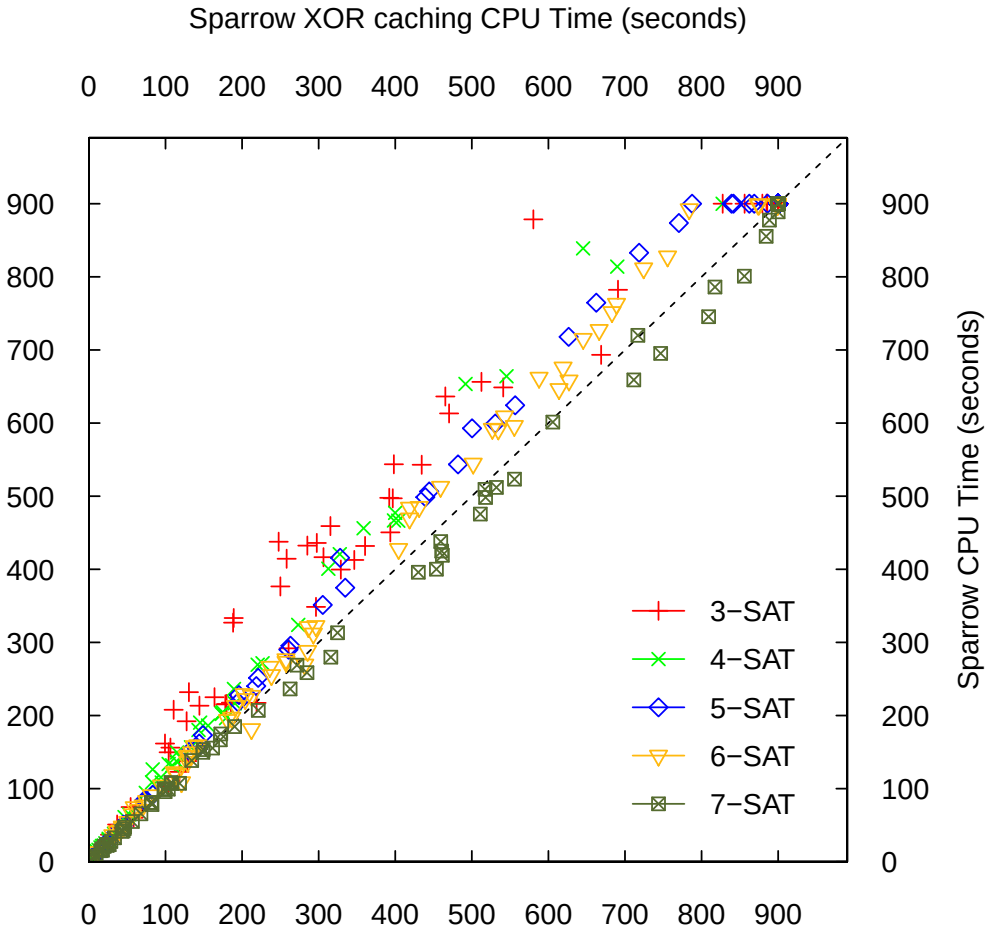

Figure: Sparrow solver with XOR scheme versus default implementation on the SAT Challenge 2012 random instances with a cutoff of 900 seconds.

Empirical Evaluation - probSAT - SC12 Hard Comb

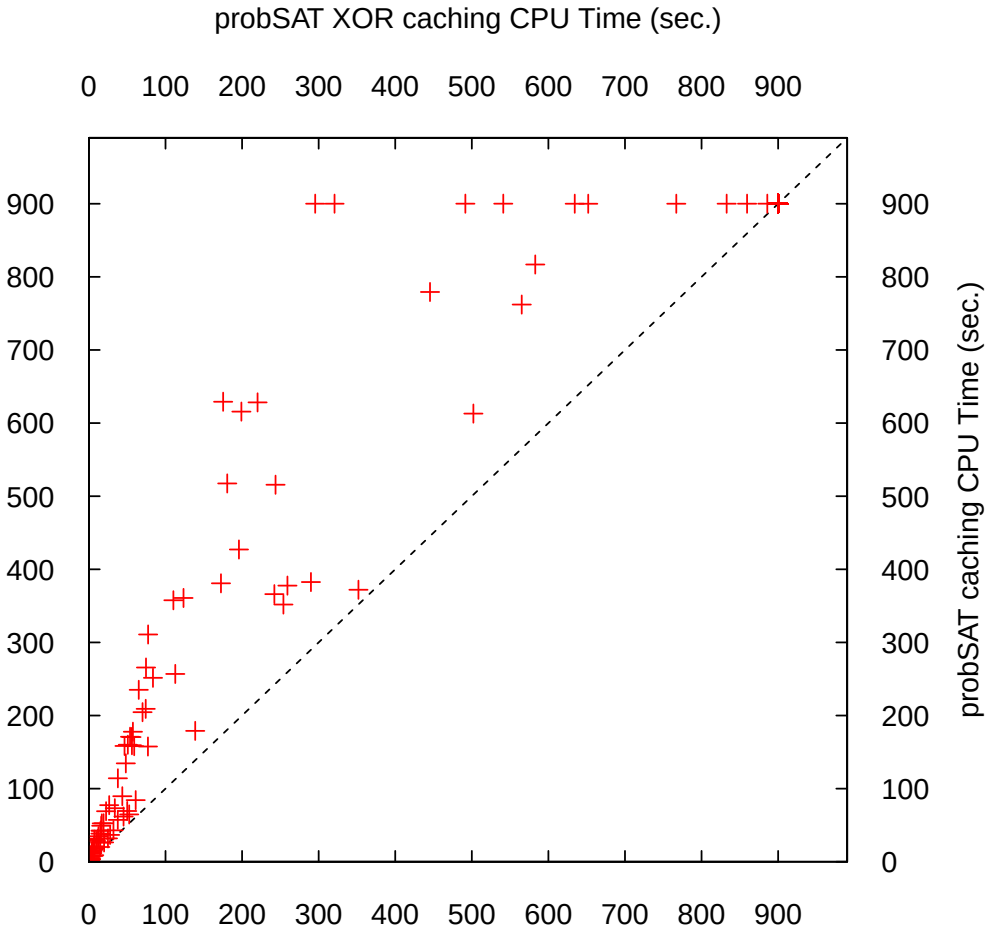

Figure: probSAT solver with XOR scheme versus default implementation on the SAT Challenge 2012 HC instances with a cutoff of 900 seconds.

Empirical Evaluation - Sparrow - SC12 Hard Comb

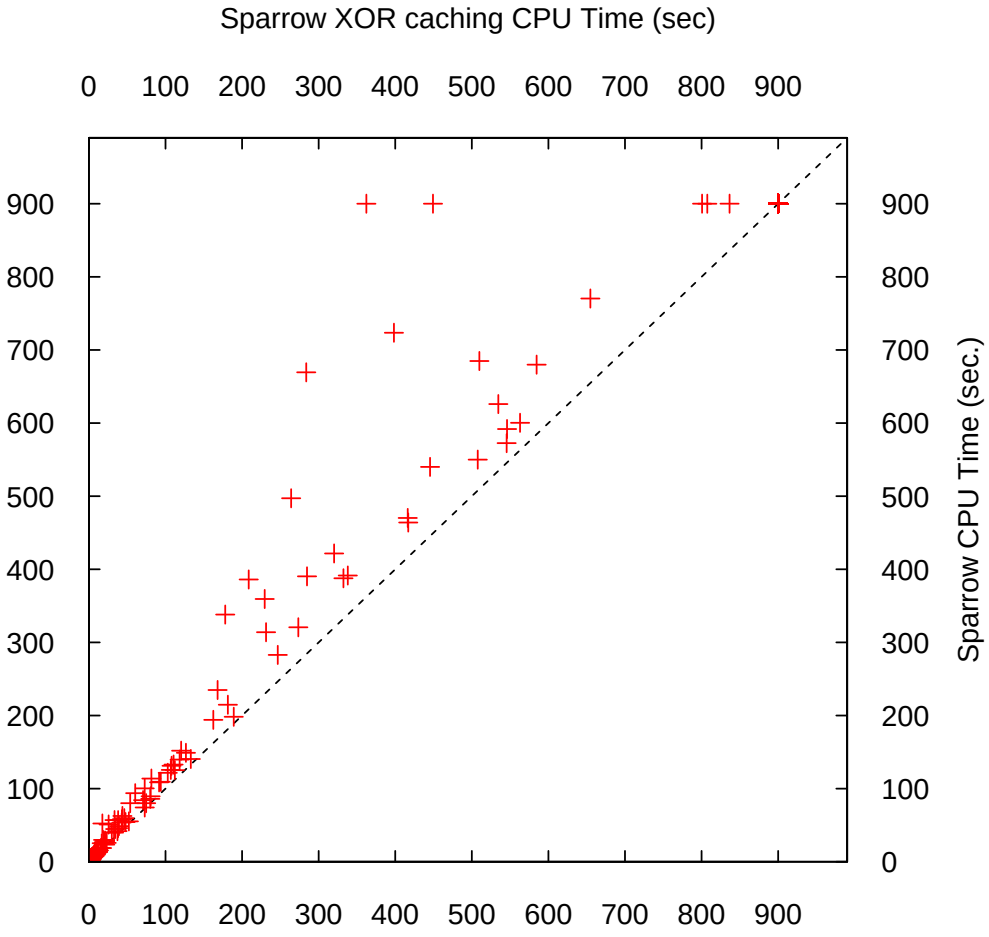

Figure: Sparrow solver with XOR scheme versus default implementation on the SAT Challenge 2012 HC instances with a cutoff of 900 seconds.

2. Multilevel properties

Defintion

- $break_1(x) =$ number of clauses $1 \to 0$-satisfied if x is flipped
- $break_l(x) =$ number of clauses $l \to (l-1)$-satisfied if x is flipped
- $make_l$ and $score_l$ are defined accordingly

- first mentioned in [CAI2013]
- used in the decision heuristics of WalkSATlm and CScoreSAT
- $break_l$ can be cheaply computed in *non-caching* implementations of SLS solvers (compute the *break* value in each iteration from scratch)
- can be easily integrated in the probSAT probability distribution

$$p(x) = cb^{-break(x)} \quad \longrightarrow \quad p(x) = \prod_l cb_l{}^{-break_l(x)}$$

$$p(x) = (1 + break(x))^{-cb} \quad \longrightarrow \quad p(x) = \prod_l (1 + break_l(x))^{-cb_l}$$

Experimental Setup - Multilevel break in probSAT

- use an automated algorithm configurator the determine appropriate values for cb_l (EDACC Parallel Automated Algorithm Configurator)
  - check the Configurable SAT Solver Challenge (CSSC 2014)
- two scenarios:
  1. configuration of cb_1 and cb_2 (version $probSAT_2$)
  2. configuration of all cb_l (version $probSAT_l$)
- on 3-SAT the configurator sets $cb_l = 1$, i.e. multilevel break useless
- compare with best performing solvers WalkSATlm and CScoreSAT
- randomly generated instances: 5-SAT and 7-SAT problems from SC11, SC12 and SC13

Results - Multilevel break in probSAT

	probSAT$_x$		probSAT$_2$		probSAT$_l$		WalkSATlm		CScoreSAT	
	#sol.	time	#sol.	time	#sol.	time	#sol.	time	#sol.	time
SC11-5-SAT	32	333s	40	34s	**40**	**19s**	39	108s	39	118s
SC12-5-SAT	67	578s	103	246s	**107**	**207s**	82	427s	77	465s
SC13-5-SAT	7	809s	5	836s	**7**	**808s**	6	817s	5	824s
5sat4000	0	900s	36	470s	**41**	**382s**	8	827s	0	900s
SC11-7-SAT	8	721s	10	521s	12	495s	11	571s	**14**	**437s**
SC12-7-SAT	49	633s	67	548s	69	488s	66	520s	**73**	**462s**
SC13-7-SAT	19	666s	17	671s	**20**	**652s**	16	673s	18	652s

- the number of solved instances (#sol.)
- the average run time (time) (counts also unsuccessful runs)
- bold values $\rightarrow$ the best achieved results for particular instance class

3. Unsatisfied Clause Selection Strategies

Selection of unsatisfied clauses

- WalkSAT type solvers select a variable from an unsatisfied clauses
- **Problem: Which unsatisfied clause to pick?**

Unsatisfied Clause Selection Strategies

- Random Selection (RS)
- Depth first search (DFS)
- Breadth first search (BFS)
- Pseudo breadth first search make first (PBFS mf)
- Pseudo breadth first search break first (PBFS bf)
- Unfair breadth first search (UBFS)

Evaluation - Unsatisfied Clause Selection Strategies

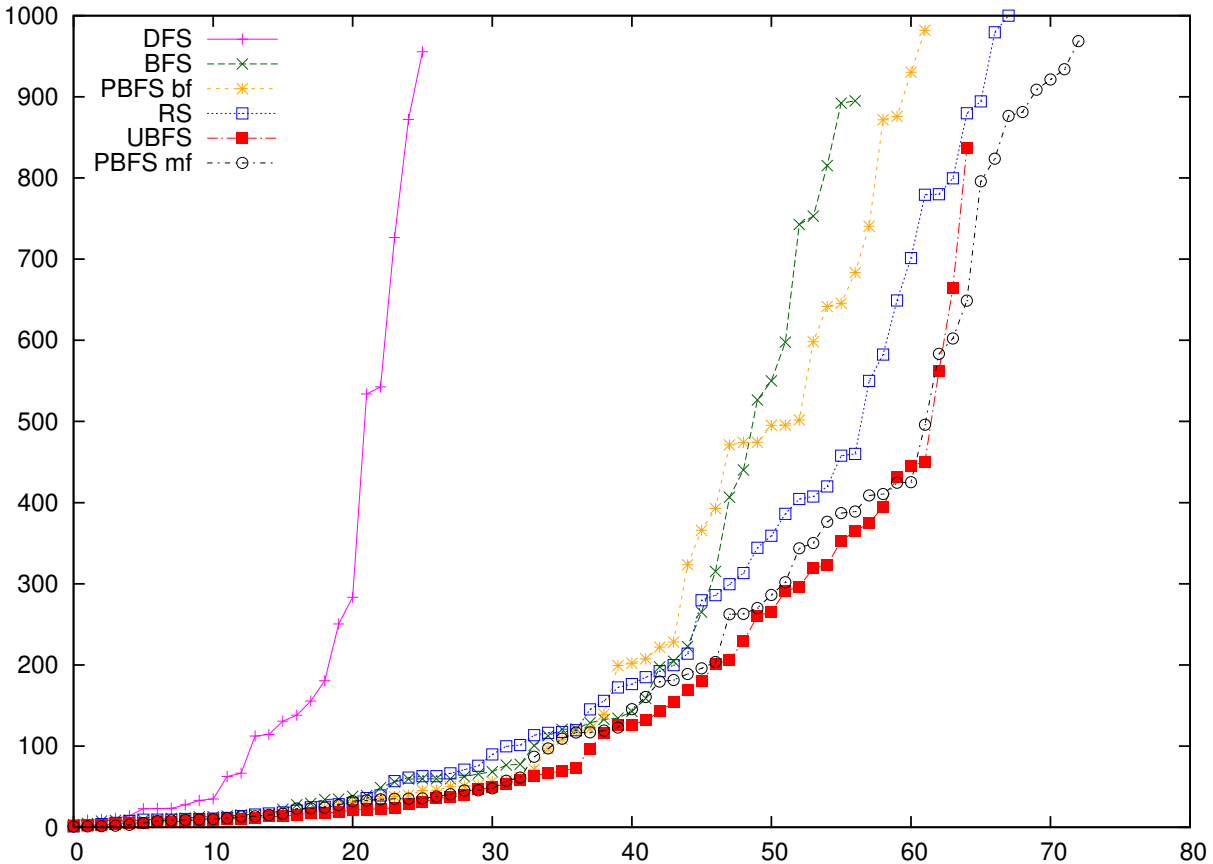

Figure: Cactus plot showing the performance of a faithful reimplementation of probSAT (yalssat) with various different clause selection heuristics on the satisfiable instances from the SAT Competition 2013 hard combinatorial track (cutoff time is 1000 seconds). The X-Axis and Y-Axis represent the number of solved instances and the runtime (in seconds), respectively.

Conclusions

- **XOR scheme** improves performance of SLS solvers implementations on a wide range random and structured instances
- **mutilevel break** heuristic of probSAT yields new state-of-the-art results on 5-SAT problems
- **clause selection heuristics** have a high impact on the performance of SLS solvers and should be rethought

Thank you for your attention!