Aalto University
School of Science
and Technology

Concurrent Clause Strengthening

Siert Wieringa and Keijo Heljanko

Department of Information and Computer Science
Aalto University, School of Science and Technology
siert.wieringa@aalto.fi

July 10, 2013

Introduction

- Modern SAT solvers rely on many techniques outside the core CDCL search procedure.
- For example preprocessing and inprocessing, but also conflict clause strengthening.
- The solver must decide when, and to what extent, it should apply such techniques.
- Instead of interleaving additional reasoning with search, both can be executed concurrently.

Using concurrency

- Avoids difficult to design heuristics for deciding when to switch between tasks.
- Exploits the availability of multi-core hardware.
- Provides a true division of work without dividing the search space.
- Concurrent clause strengthening yields surprisingly consistent performance improvements.

Clause strengthening

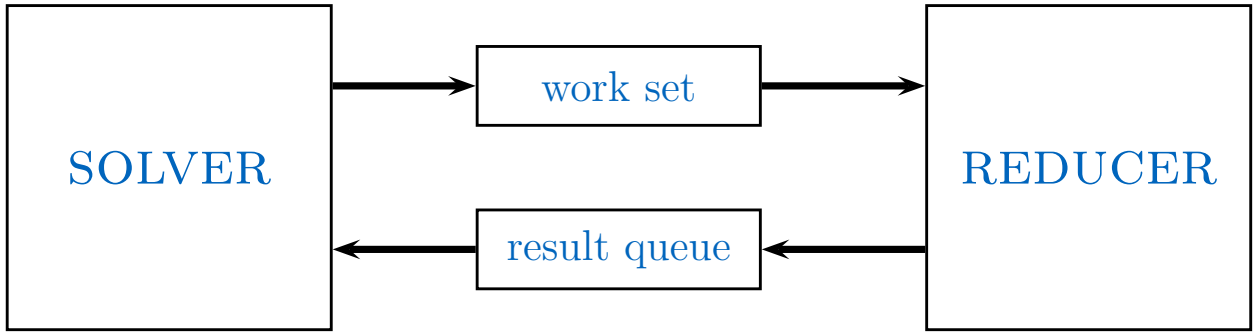
- Strengthening a clause means removing redundant literals.

Given:	A clause	c	such that $\mathcal{F} \models c$
Find:	A subclause	$c' \subseteq c$	such that $\mathcal{F} \models c'$

- Finding c' such that it is of minimal length is an NP-hard problem.

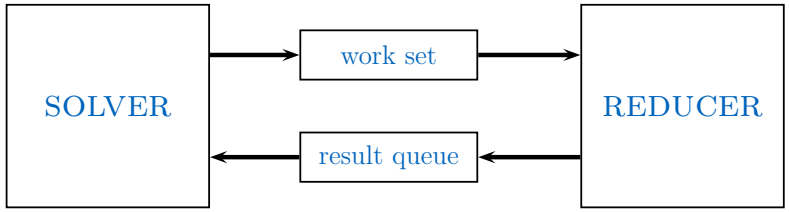
- MiniSAT minimizes all conflict clauses with respect to the clauses used in their derivation.

The solver-reducer architecture



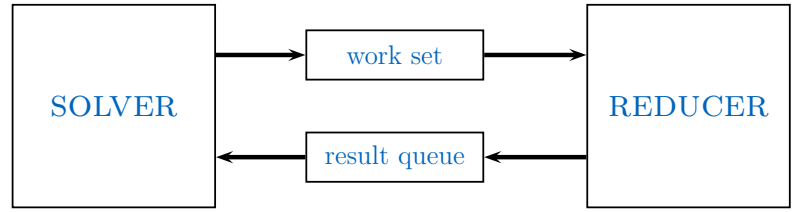
- Two concurrently executing threads.
- The SOLVER is a conventional CDCL solver.
- The REDUCER provides a clause strengthening algorithm.
- Communication solely by passing clauses through the work set and the result queue.

Basic operation



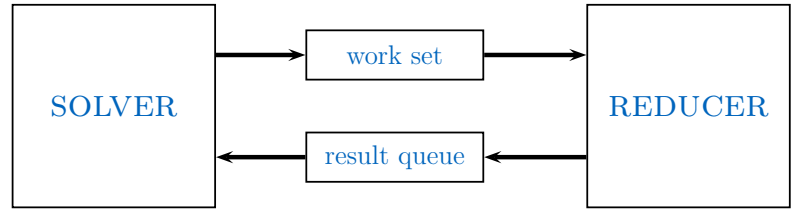
- Whenever the SOLVER learns a clause it writes a copy of that clause to the work set.
- The REDUCER reads its input clauses from the work set, and writes clauses it has strengthened to the result queue.
- The SOLVER frequently introduces clauses from the result queue to its learnt clause database.
- The REDUCER has its own copy of the problem clauses as well as its own learnt clause database.

The REDUCER's algorithm



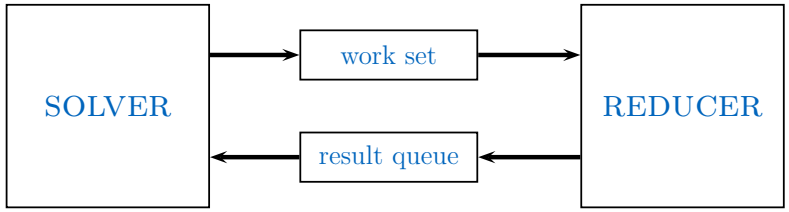
- Assign a literal of input clause c to **false**, then perform unit propagation.
- Remove from c literals that became assigned **false** during unit propagation.
- Repeat until all literals of c are assigned **false**, or a conflict arises.
- If a conflict arises then analyze, learn, and return the subclause $c' \subseteq c$ containing literals "causing" the conflict.
- Otherwise, add c to the learnt clause database, return c.

The work set



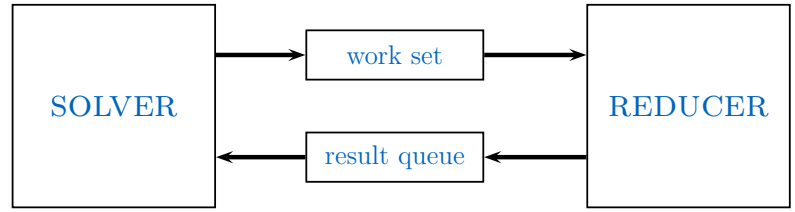
- As the REDUCER learns, it becomes stronger but slower.
- The REDUCER can usually not keep up with the supply of clauses from the SOLVER.
- How to implement the work set?
- FIFO - Tends to deliver clauses to the REDUCER that are old, and often no longer interesting.
- LIFO - Strong clauses may never be delivered as they shift backwards in the queue quickly.

Sorting the work set



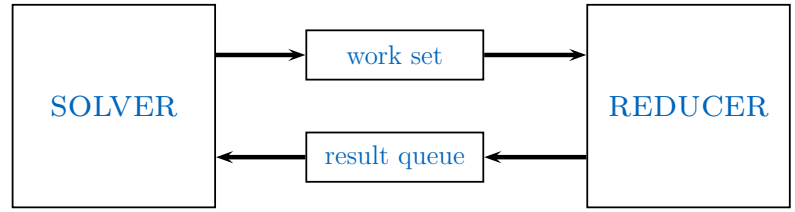
- We can use clause length or LBD as an approximation for clause quality.
- As the average length changes clauses that were relatively long when learnt may seem short when they are old.
- Solution: Limit the capacity.
- If the SOLVER adds a clause to a full work set then this clause replaces the oldest clause.
- If the REDUCER requests a clause from a non-empty work set it receives the best clause.

Keeping it simple



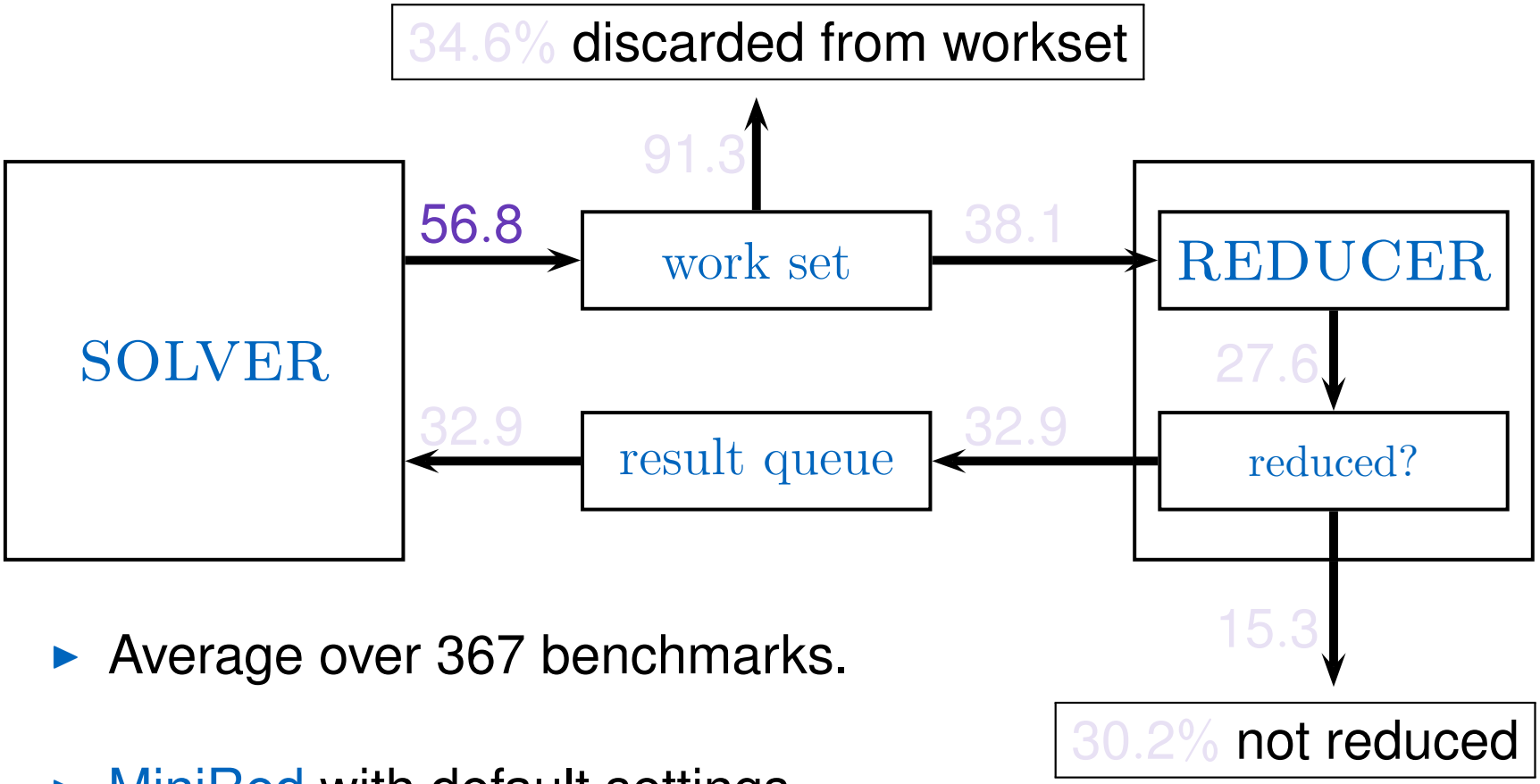
- The REDUCER only returns clauses that are strict subclauses of its inputs.
- The REDUCER does not share its learnt clauses.
- The REDUCER assigns literals in the order they appear in the input clause.
- The SOLVER does not have a mechanism for deleting clauses for which a subclause is found in the result queue.
- The result queue is a simple unbounded FIFO queue.

Implementation



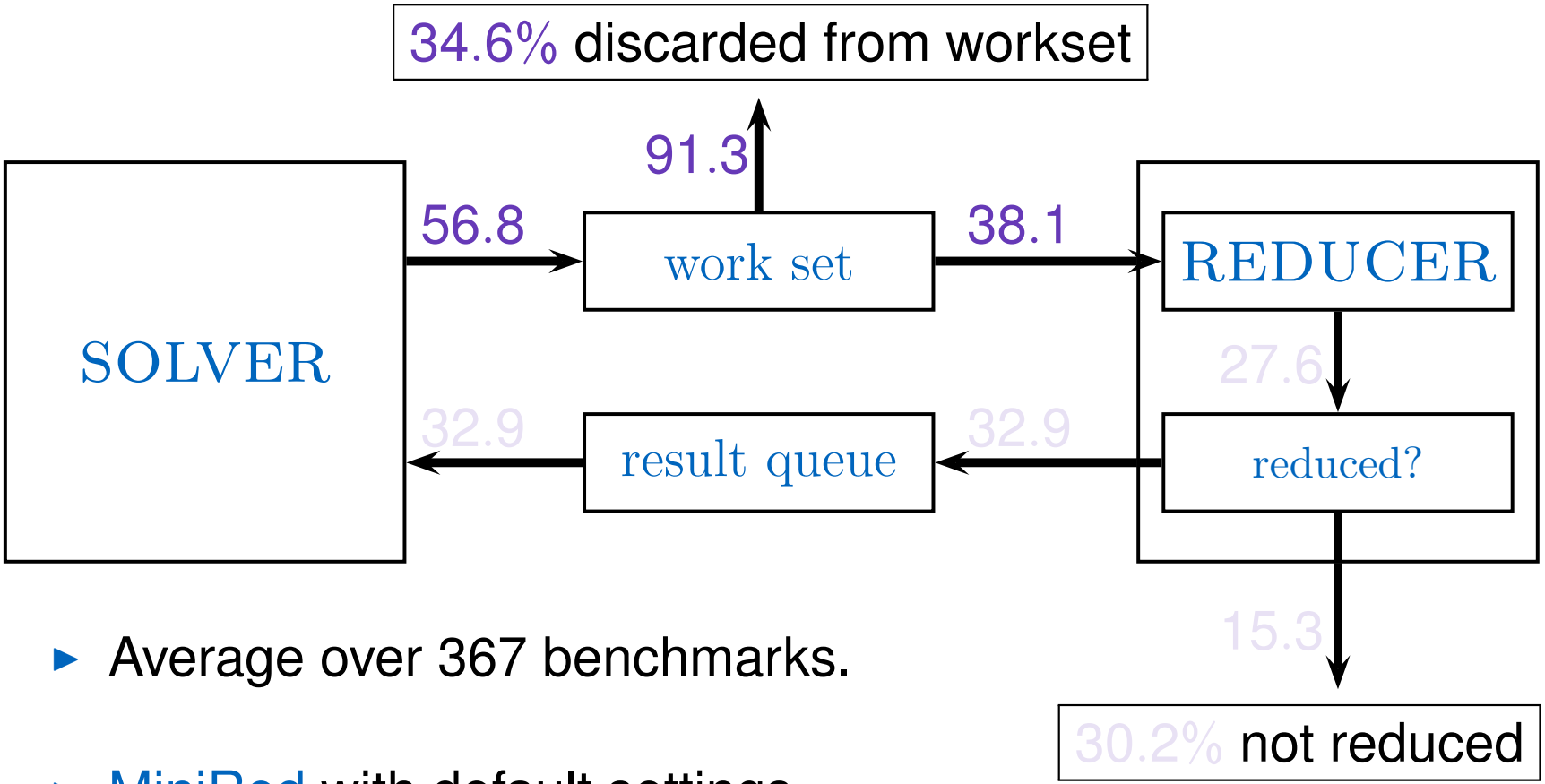
- MiniRed based on MiniSAT 2.2.0.
- GlucoRed based on Glucose 2.1 / 2.2.
- Base solvers modified as little as possible.
- The code added to both solvers is identical, except:
- MiniRed sorts its work set by clause length.
- GlucoRed sorts its work set by LBD.

Average clause length experiment



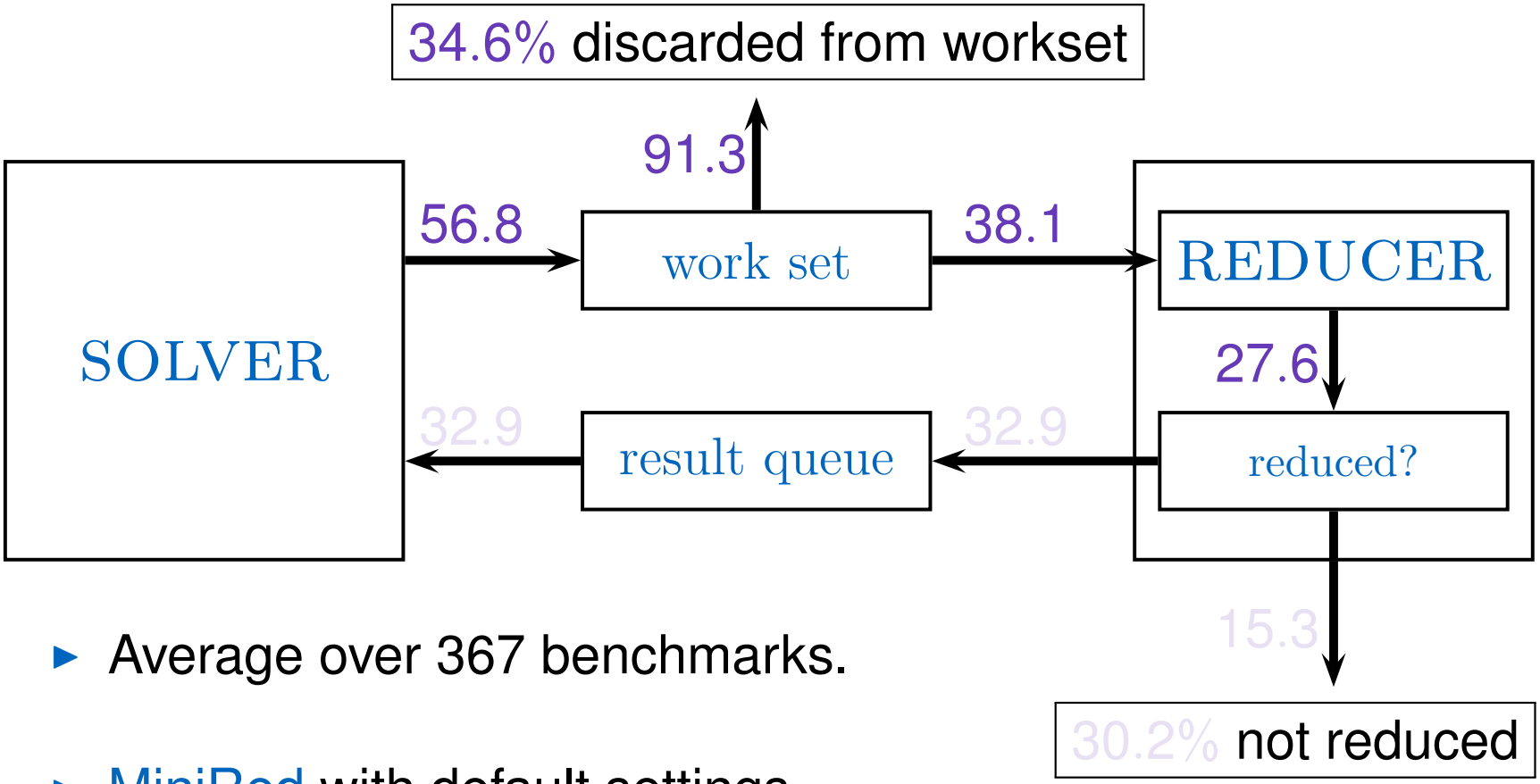
- Average over 367 benchmarks.
- MiniRed with default settings.
- work set capacity 1000 clauses.

Average clause length experiment



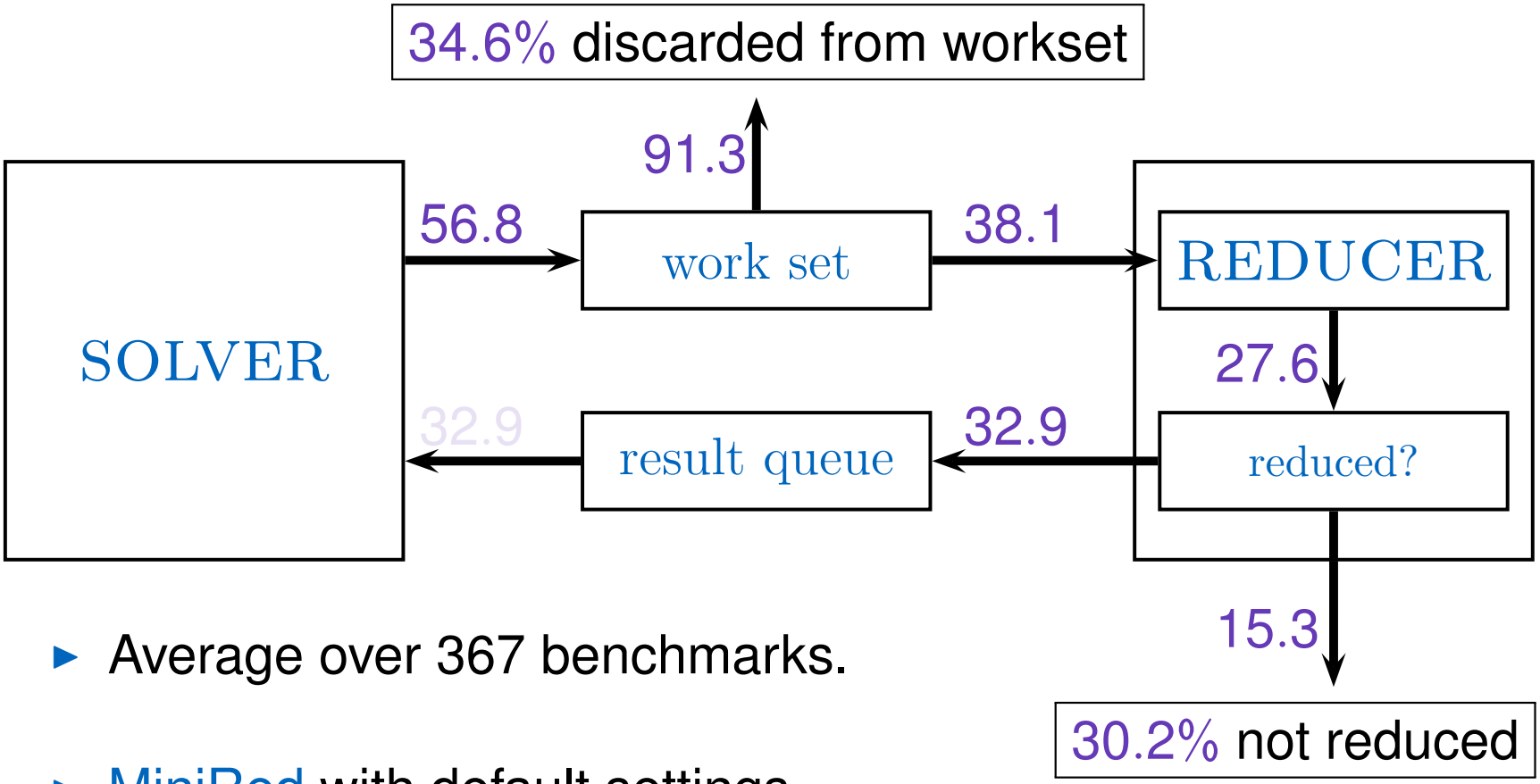
- Average over 367 benchmarks.
- MiniRed with default settings.
- work set capacity 1000 clauses.

Average clause length experiment



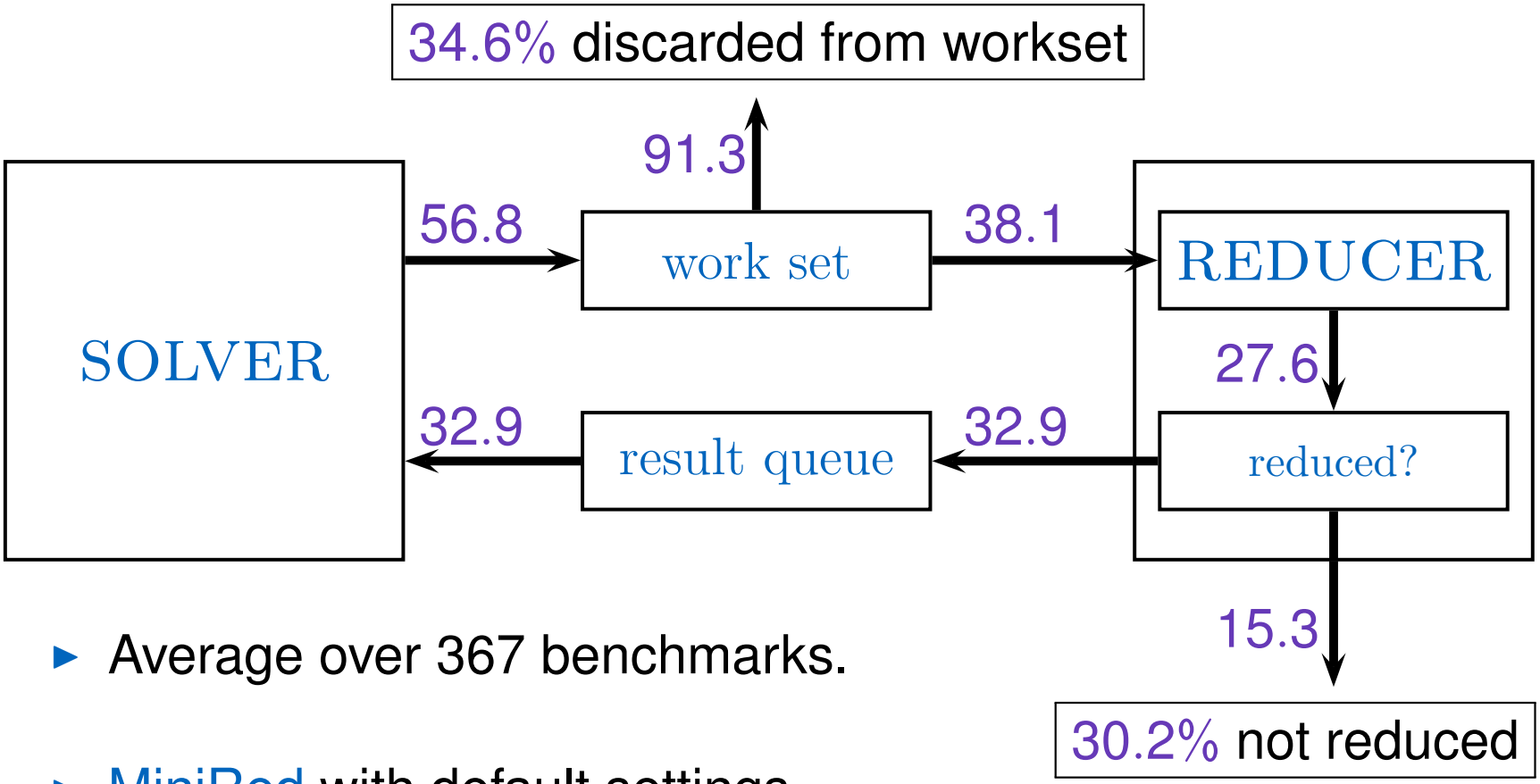
- Average over 367 benchmarks.
- MiniRed with default settings.
- work set capacity 1000 clauses.

Average clause length experiment



- Average over 367 benchmarks.
- MiniRed with default settings.
- work set capacity 1000 clauses.

Average clause length experiment

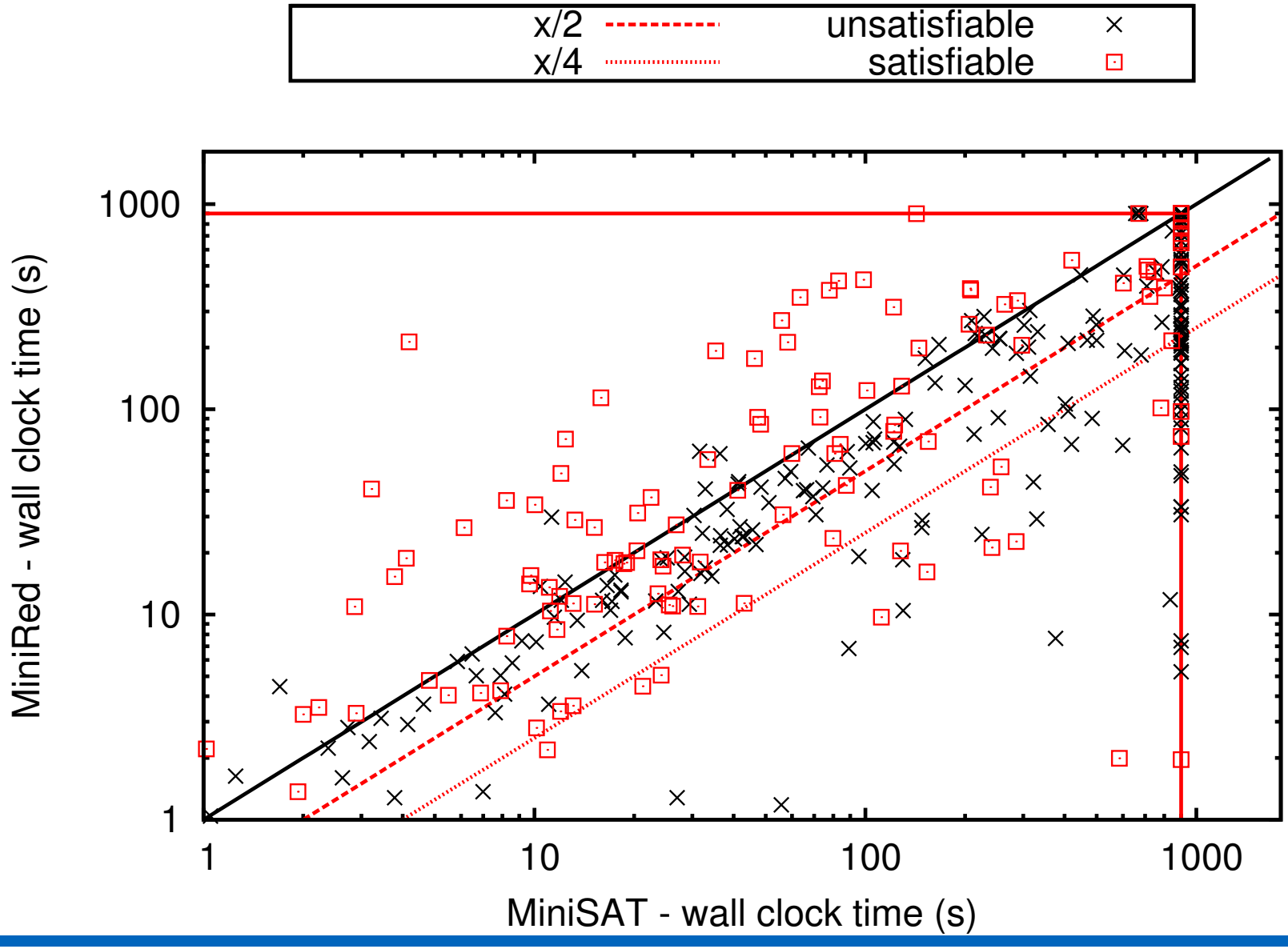


- Average over 367 benchmarks.
- MiniRed with default settings.
- work set capacity 1000 clauses.

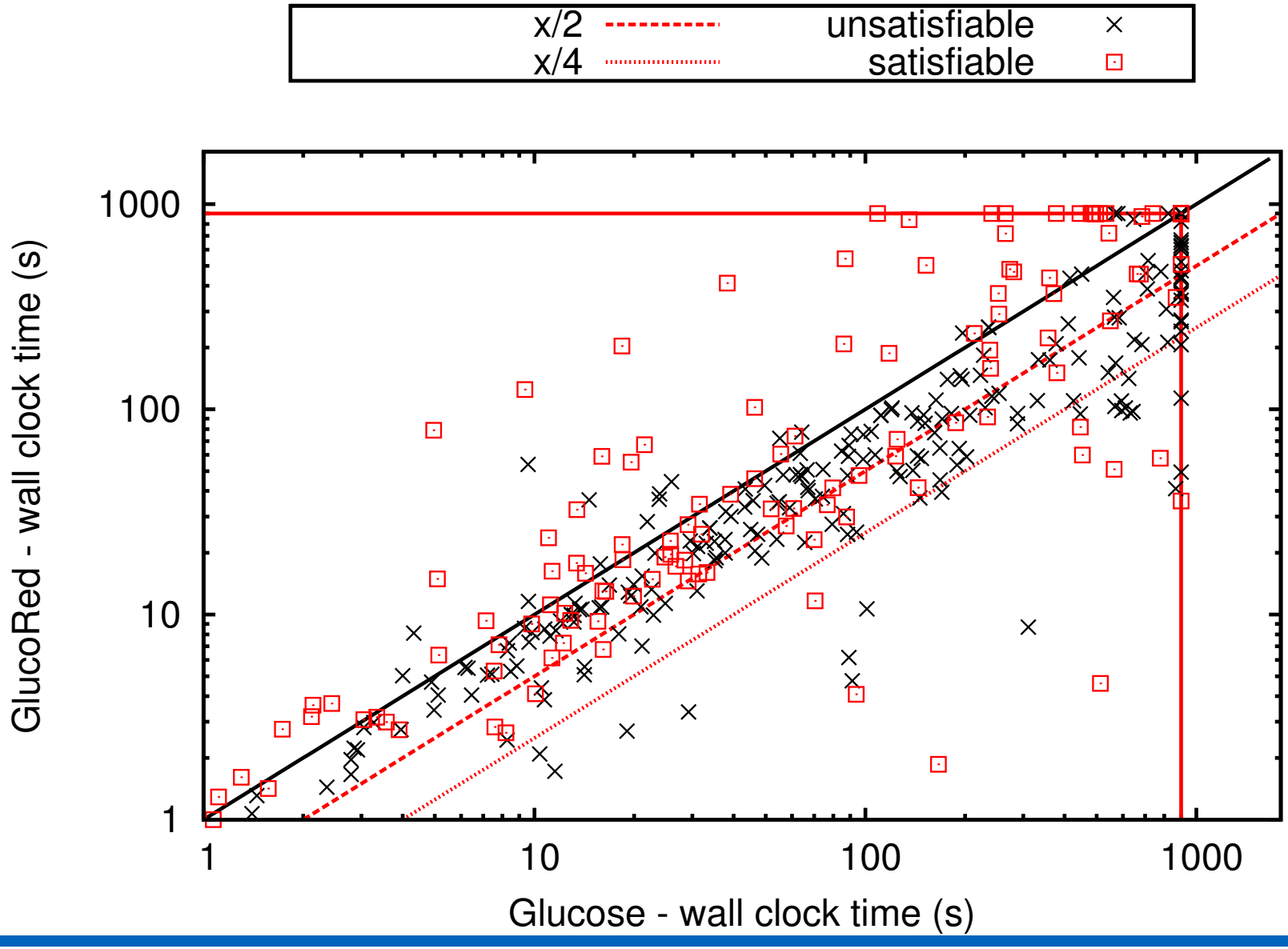
Performance testing

- The set Competition contains 547 application track benchmarks (Competition 2011/Challenge 2012).
- The set Simplified contains 501 benchmarks resulting from running SatElite on the Competition set.
- In these slides we will only present results for the Simplified set.
- 900 second wall clock time limit.
- 1800 second CPU time limit.

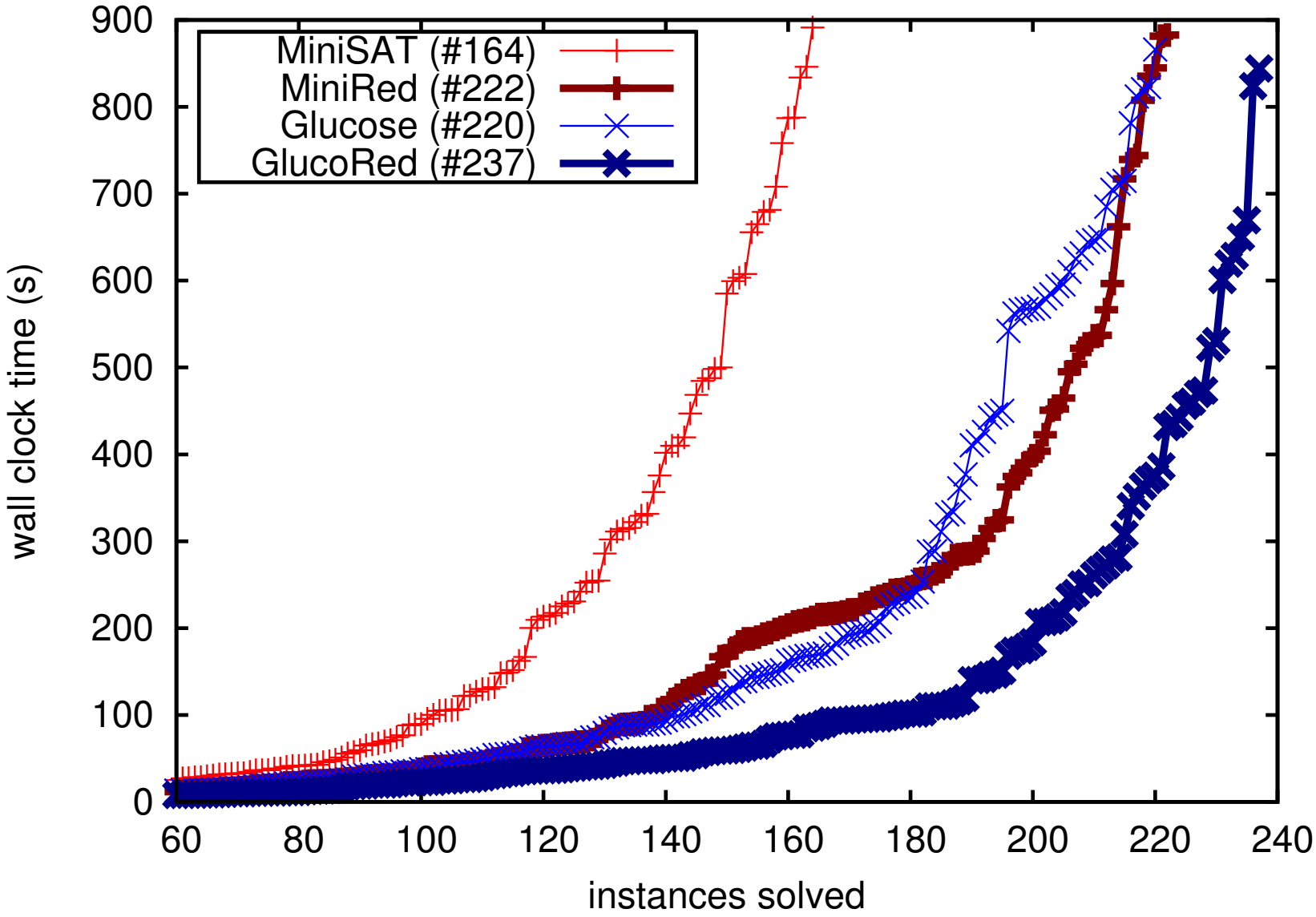
MiniRed scatter plot



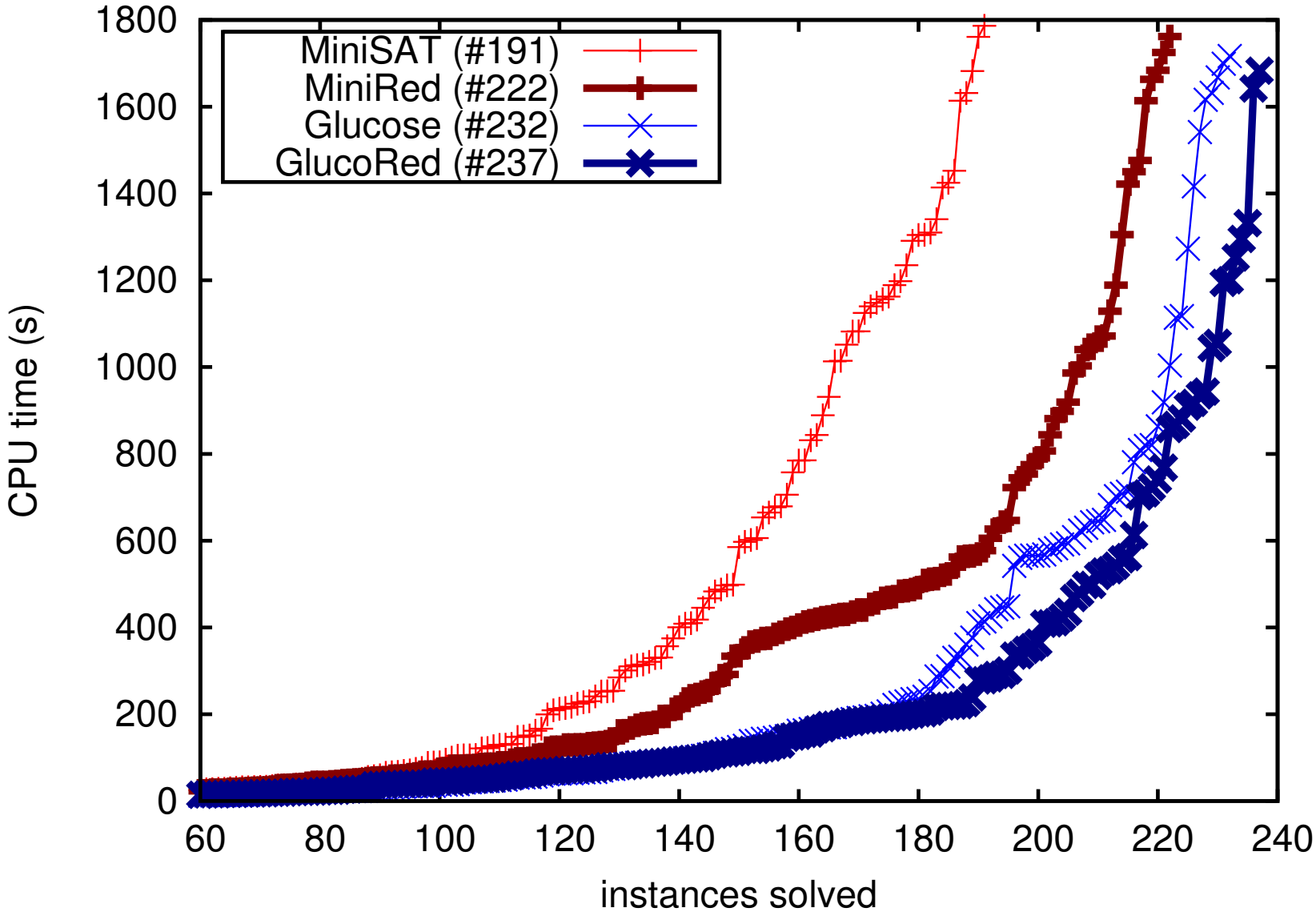
GlucoRed scatter plot



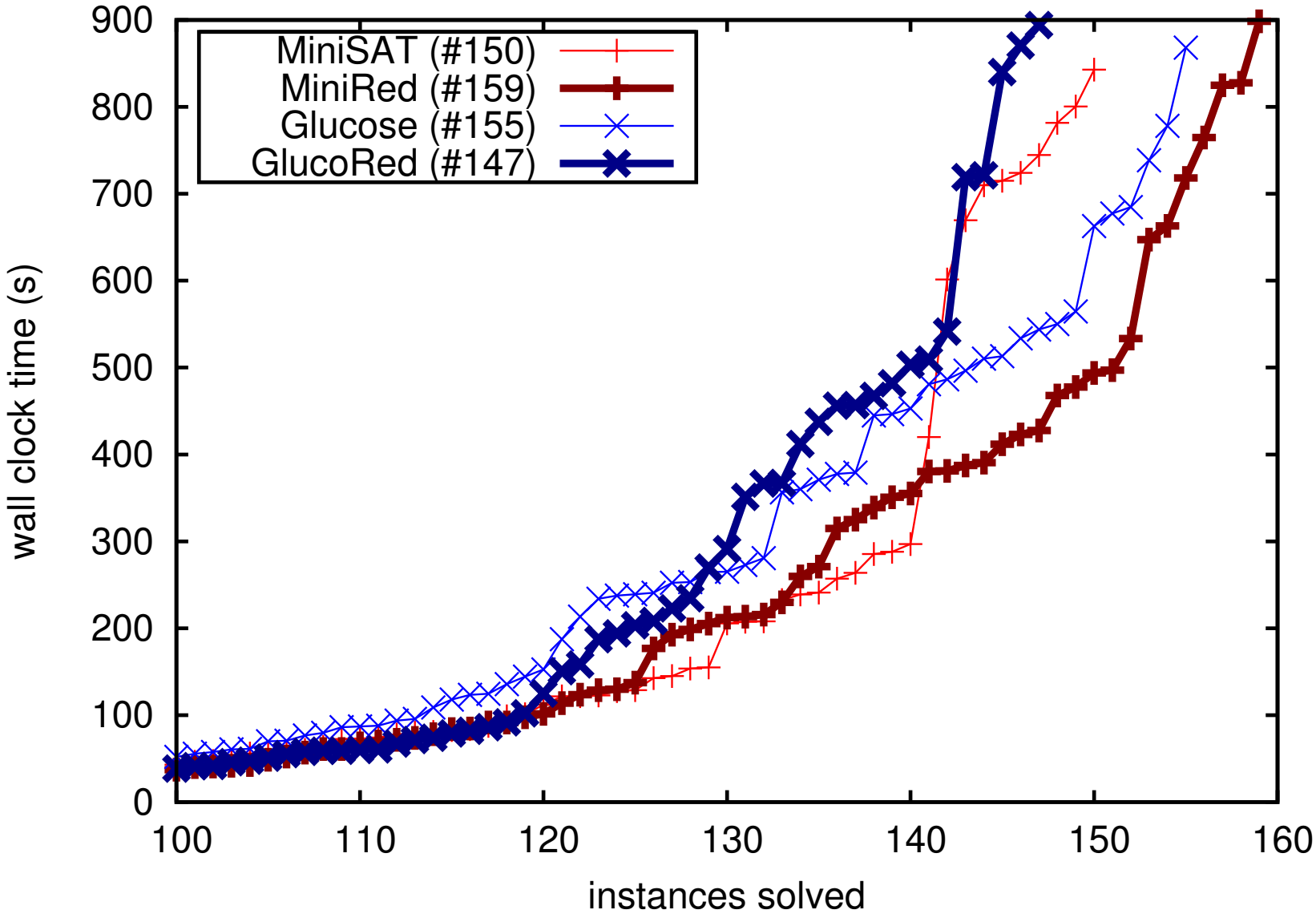
UNSAT benchmarks - wall clock time cactus



UNSAT benchmarks - CPU time cactus



SAT benchmarks - wall clock time cactus



Results discussion

- Concurrent clause strengthening is strong on unsatisfiable benchmarks.

		GlucoRed	PeneLoPe		
		2-core	2-core	4-core	8-core
UNSAT	Wall clock	237	227	231	247
	CPU	237	227	221	217
SAT	Wall clock	147	142	160	164
	CPU	149	142	154	149

- Portfolio solvers expose orthogonal behavior.
- The two approaches can be combined!

Conclusions

- Concurrent clause strengthening is a simple technique, providing significant performance improvements.
- Particularly strong on unsatisfiable benchmarks.
- Using concurrency to aid CDCL search, rather than to parallelize it.
- The basic idea can be exploited in many ways, e.g. concurrent inprocessing.

Availability

- Source code for MiniRed and GlucoRed is available from:

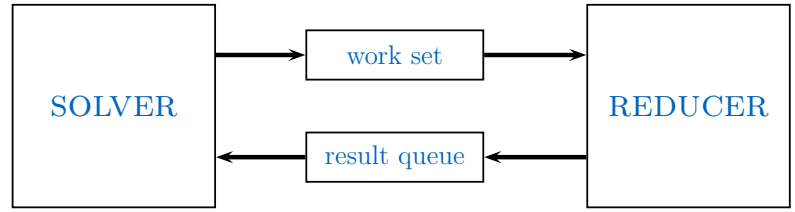
  `http://bitbucket.org/siert`

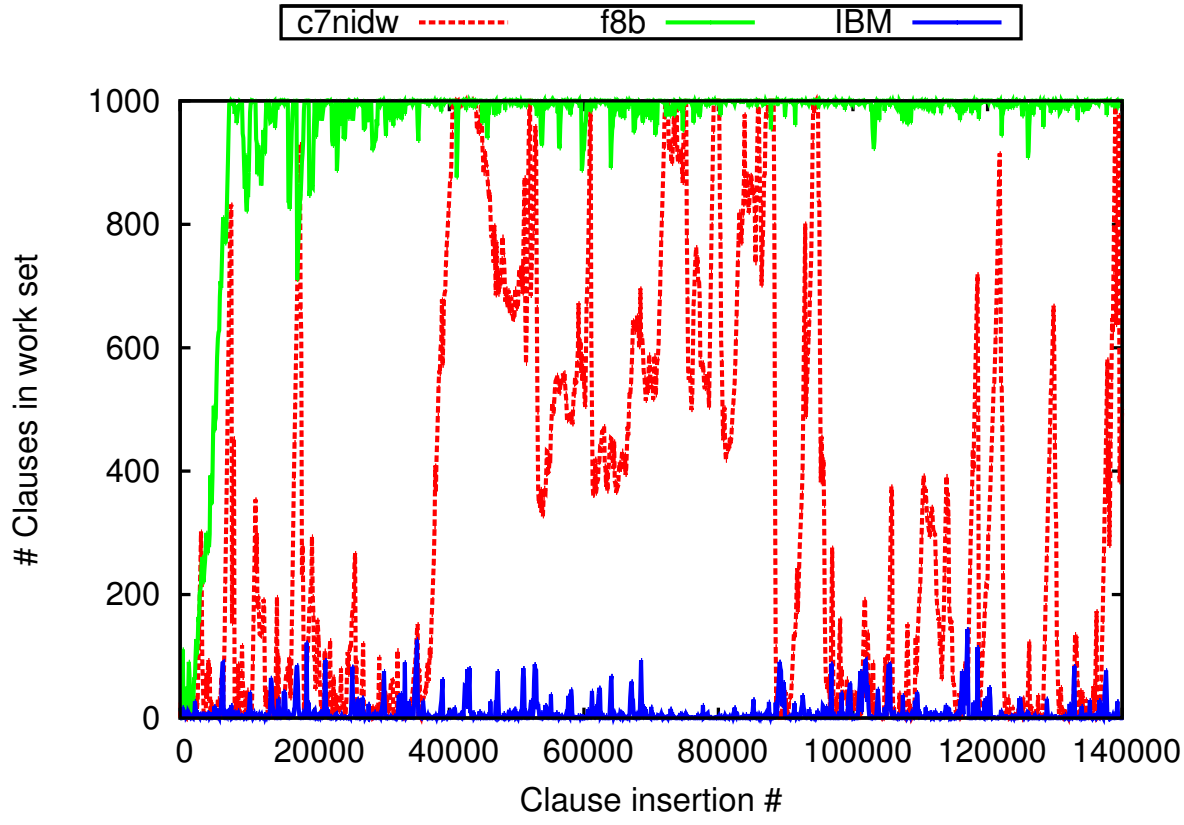
- MiniRed and GlucoRed have been integrated in ZZ:

  `http://bitbucket.org/niklaseen`

- The ZZ-framework by Niklas Eén provides the Bip model checker, including e.g. PDR and BMC algorithms.

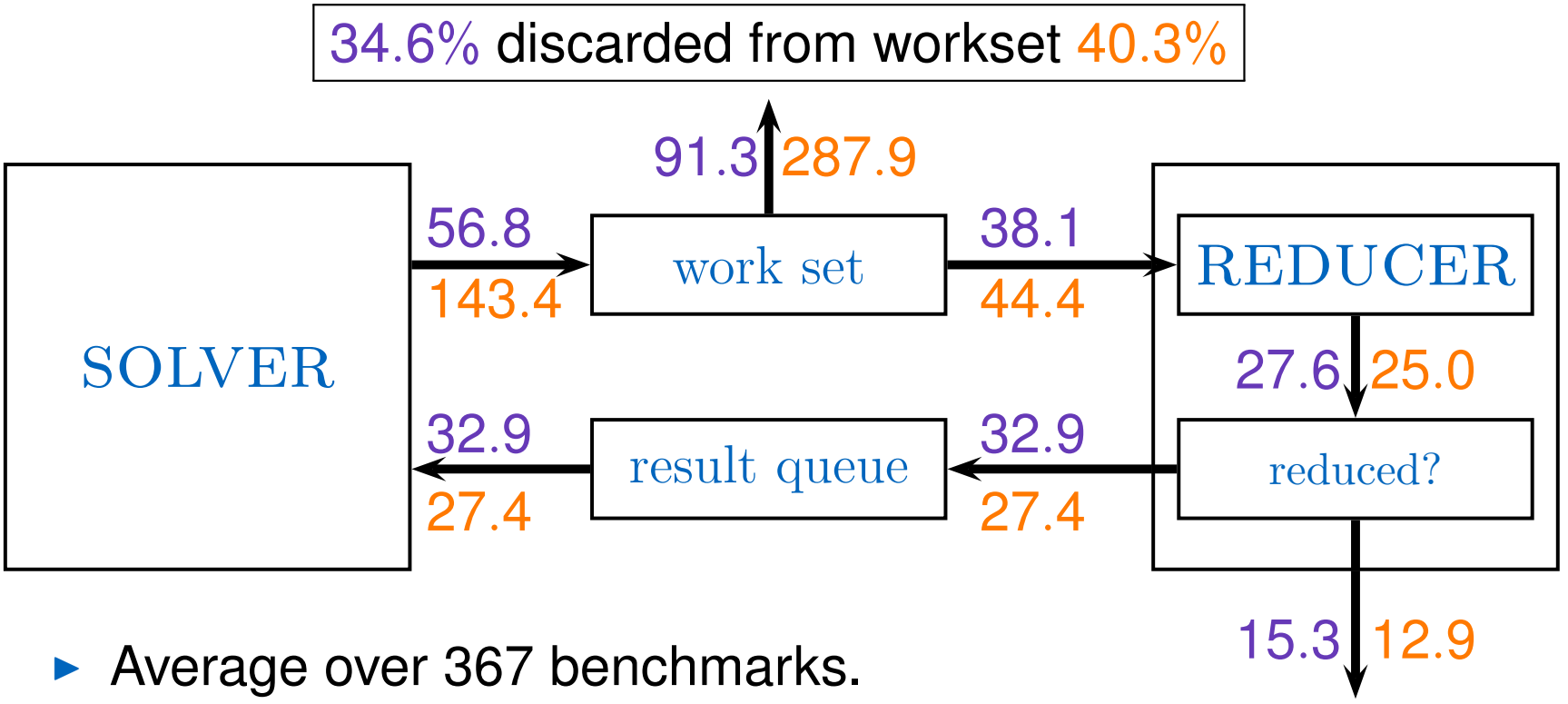
Capacity of the work set





- The default work set capacity is 1000 clauses.

Average clause length experiment (2)



- Average over 367 benchmarks.
- Clause min. disabled in SOLVER.
- Total number of clauses generated increased by 17%.