Effective Preprocessing in SAT through Variable and Clause Elimination

Niklas Eén
Cadence Berkeley Laboratories
Berkeley, USA

Armin Biere
Johannes Kepler University
Linz, Austria

for ***SAT 2005***

The Eighth International Conference on
Theory and Applications of Satisfiability Testing

June 20, 2005

Problem Statement: Clausification of Netlists

- Typically represented using And-Inverter graphs.
- We can write a good clausifier... [JS04,Vel05]
- Can we “recover” after we translated to clauses?
- Yes, do preprocessing in the SAT solver!
  - Global view $\Rightarrow$ Better results
  - More general
  - Saves the *user* of the SAT solver the trouble of writing a good clausifier (to some extent).

Purpose of Preprocessing

- Slim down the CNF to a core without "obvious" redundancies, such as:
  - "Meaningless" internal variables.
    - Big conjunctions: a & (b & (c & d))
    - MUXes: (s & x) | (~s & y)
  - Equivalent literals.
    - Binary clauses: (x → y), (y → x)
  - Unpropagated shallow facts (generalize BCP).
    Example. *Hyper-unary-resolution:*
    - Clauses: (a | b | c), (a → x), (b → x), (c → x)

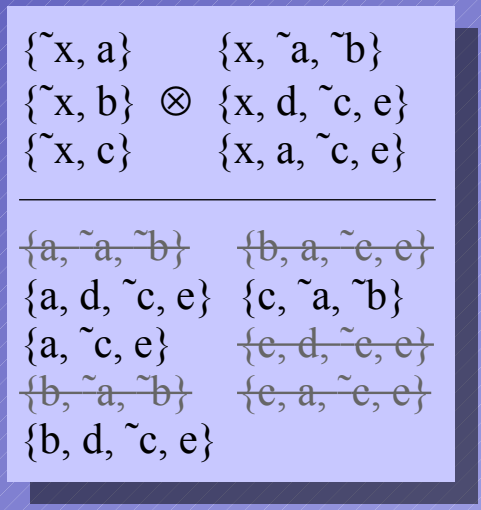
The Preprocessor SATELITE

- We implement three basic techniques:
  - Variable elimination by resolution.

    Takes care of wasteful internal variables (and equivalent literals).*
  - Fast subsumption. $C \subseteq C'$ (backward: $C_{new} \subseteq C_{old}$)

    Removes "trash" produced by the variable elimination (important!).
  - Self-subsumption. $(x \mid A) \otimes_x (\sim x \mid A \mid B) = (A \mid B)$

    Gives a generalization of BCP (Niklas Sörensson)



Outline of Algorithm

Repeat until no more changes:

- *selfSubsume()*
- *subsume()*
- *eliminateVars()*

A clause may become unit ⇒ ***top-level unit propagation.***

Eliminate "x" if leads to fewer clauses.

- Backward subsumption and BCP applied eagerly.
- Made efficient by tracking clauses that are...
  - strengthened by *selfSubsume()* or BCP.
  - removed by subsumption or BCP.
  - added or removed by *eliminateVars()*.

Details on Variable Elimination

- Try variable x where $\#occur(x) \times \#occur(\sim x)$ is smallest first.
- If both $\#occur(x)$ and $\#occur(\sim x)$ are >10, skip.
- For pragmatic reasons, detect definitions:
  - Clauses: $(t \rightarrow x)$, $(t \rightarrow y)$, $(x \,\&\, y \rightarrow t)$ mean "$t \leftrightarrow x \,\&\, y$" $\Rightarrow$ eliminate t by inlining.
  - Natural to do *hyper-unary-resolution* during this step.

Details on Subsumption

- Maintain *occurs*-lists. [Biere04]
  - *occur(p)* is the set of clauses where *p* occurs.
- For each clause, store a 64-bit signature.
  - Hash each literal to a number 0..63. Take bitwise OR.

```
subset(Clause C, Clause C')
    if (size(C) > size(C')) return FALSE
    if (sig(C) & ~sig(C')) return FALSE
    return result of complete (expensive) subset test

findSubsumed(Clause C)
    pick the literal p in C with the shortest occur list
    for each C' ∈ occur(p) do
        if (C ≠ C' && subset(C, C' ))
            add C' to result
    return result
```

Demo!

- Small BMC problem unrolled 4 time-frames.
- Resulting boolean circuit clausified by introducing one variable per binary AND/XOR.

SMV file

```
module main()
{
   num : array 4..0 of boolean;

   init  (num) := 6;
   next(num) := (~num[0]) ? (num >> 1) : (num * 3 + 1);

   test : assert G (num ~= 1);
}
```

Missing variables in model?

- We need to extend the satisfying assignment. This is cheap!
- In an incremental SAT solver, we need to "freeze" variables.

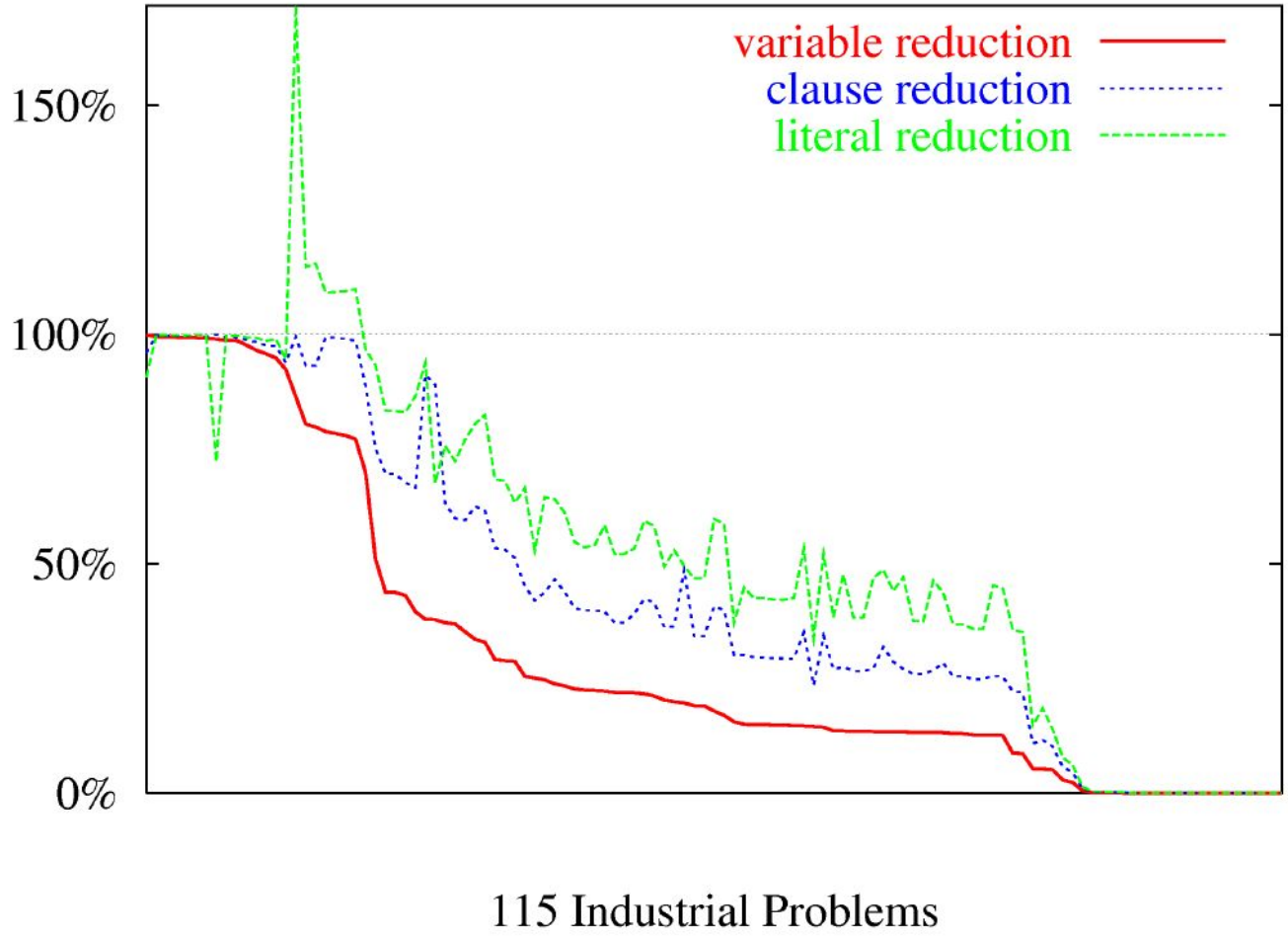
Experimental Results

- *MINISAT* solved 376 industrial problems in 1st-stage. *SATELITE* + *MINISAT* solved 402 problems.
- We investigate:
  - size reduction
  - runtime effect

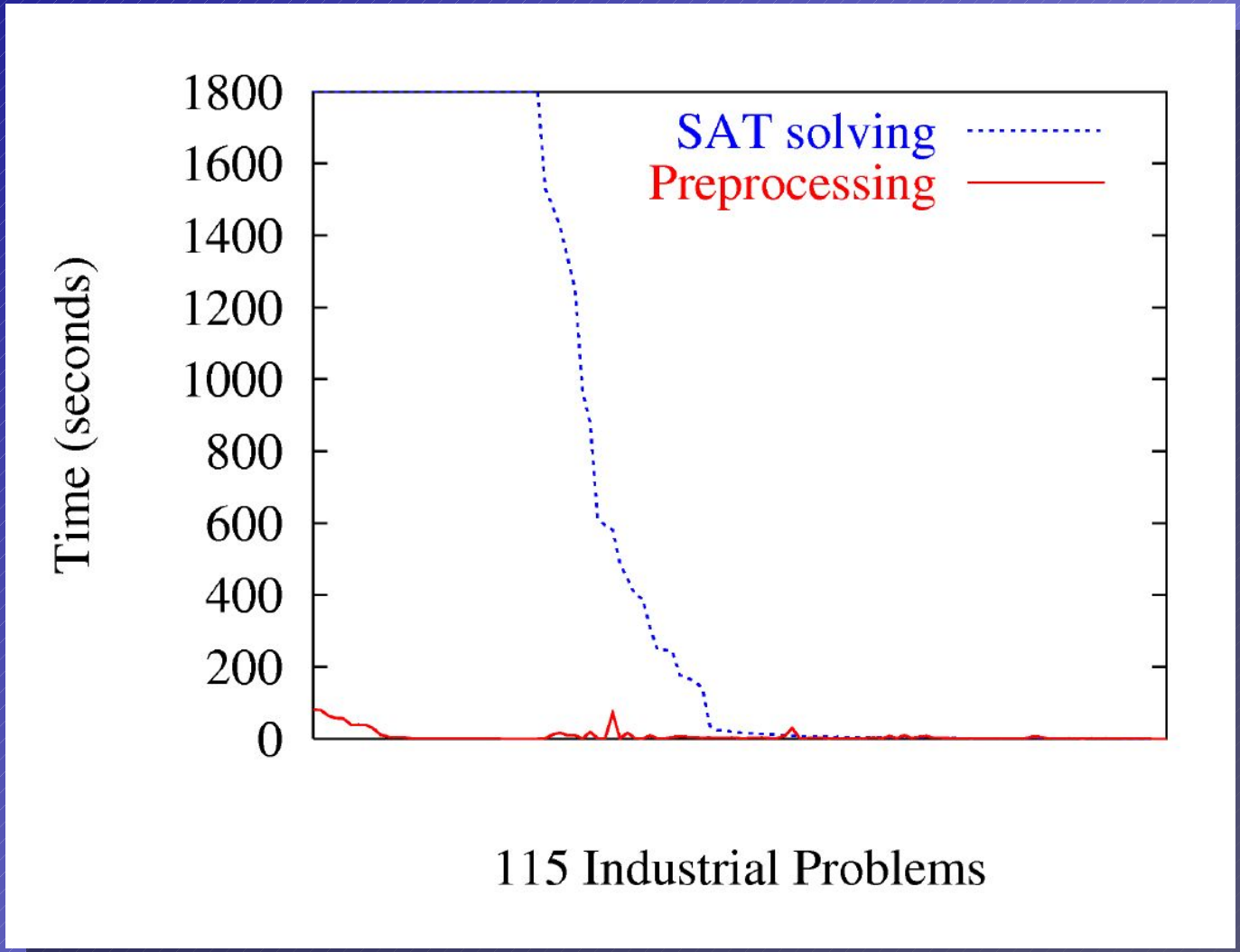
Size Reductions

	Original			NiVER			SatELite		
	#vars	#clauses	#literals	#vars	#clauses	#literals	#vars	#clauses	#literals
6pipe	15,469	394,739	1,157,225	15,067	393,239	1,154,868	**11,863**	**322,584**	**1,017,510**
abp1-1-k31	14,809	48,483	123,522	8,183	34,118	97,635	**3,075**	**17,687**	**63,252**
barrel9	8,903	36,606	102,370	4,124	20,973	**66,244**	**1,712**	**16,195**	87,294
cache_10	227,210	879,754	2,191,576	129,786	605,614	1,679,937	**28,902**	**177,868**	**747,696**
comb2	31,933	112,462	274,030	20,238	89,100	230,537	**3,072**	**18,327**	**63,466**
f2clk_40	27,568	80,439	186,255	10,408	44,302	125,040	**4,273**	**24,791**	**81,251**
fifo8_400	259,762	707,913	1,601,865	68,790	300,842	858,776	**23,120**	**129,358**	**445,717**
guidance-1-k	98,746	307,346	757,661	45,111	193,087	553,250	**22,939**	**129,662**	**442,767**
ibm-rule03_k	88,641	375,087	972,575	55,997	307,728	887,363	**28,320**	**190,387**	**629,470**
ibm-rule20_k	90,542	373,348	945,389	46,231	281,252	832,479	**19,782**	**155,907**	**545,851**
ip50	66,131	214,786	512,828	34,393	148,477	398,319	**7,925**	**42,965**	**138,610**
longmult15	7,807	24,351	58,557	3,629	16,057	45,899	**1,442**	**8,725**	**28,497**
w08_14	120,367	425,316	1,038,230	69,186	323,985	859,105	**33,934**	**220,220**	**688,301**

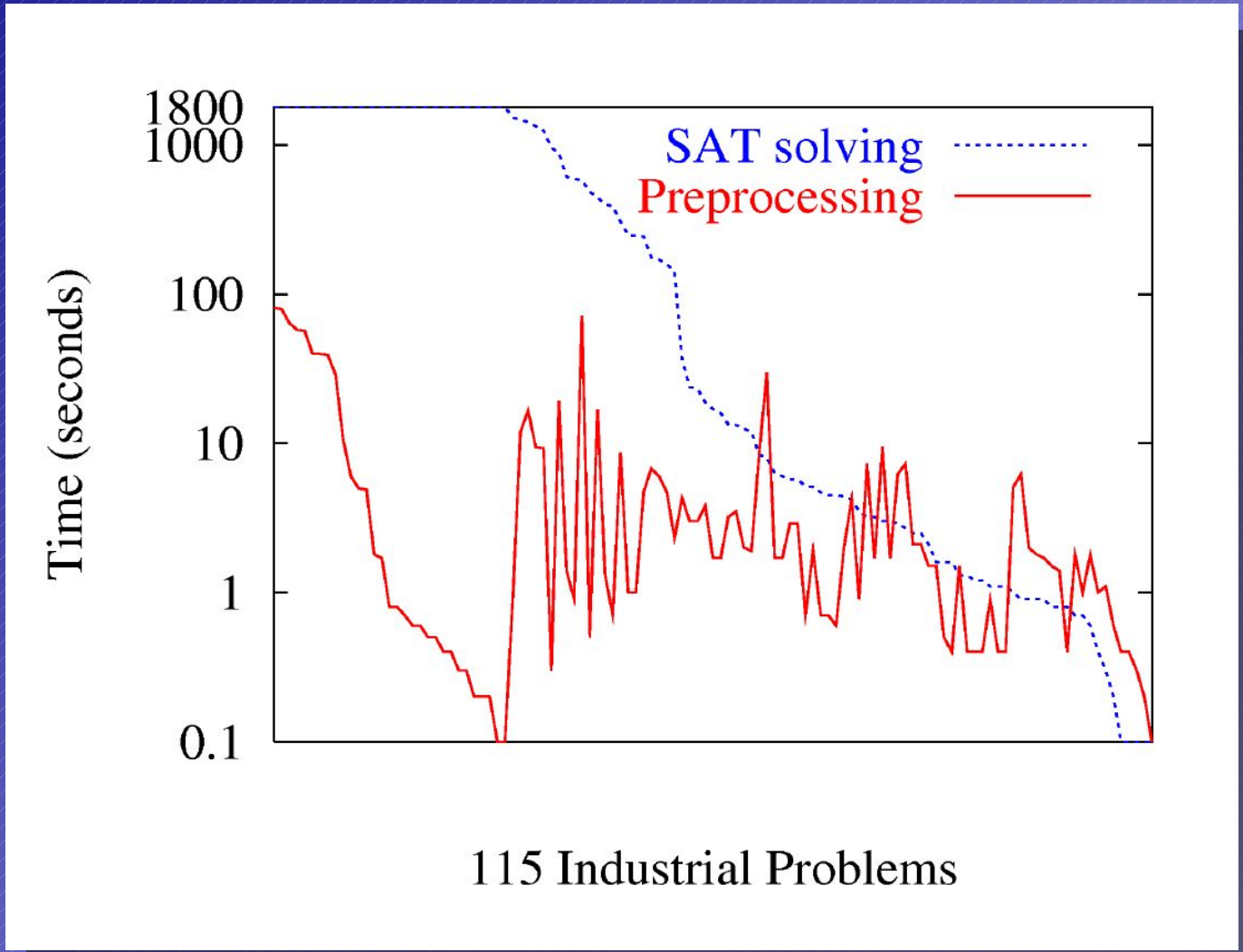
Size Reductions



Preprocessing Time

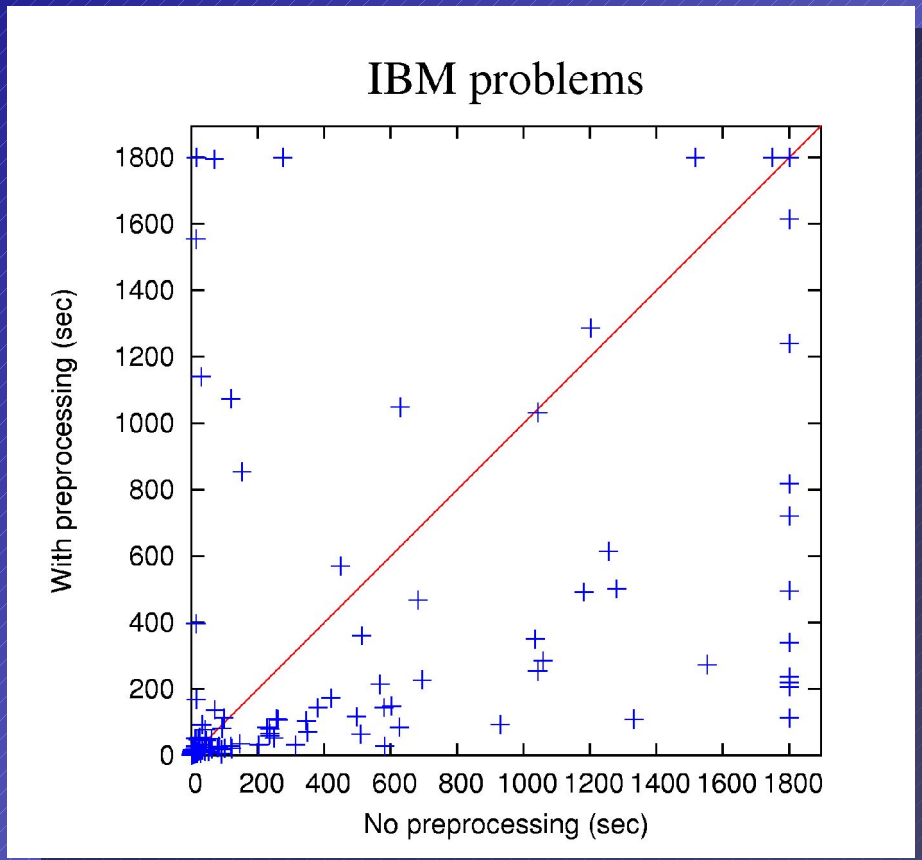


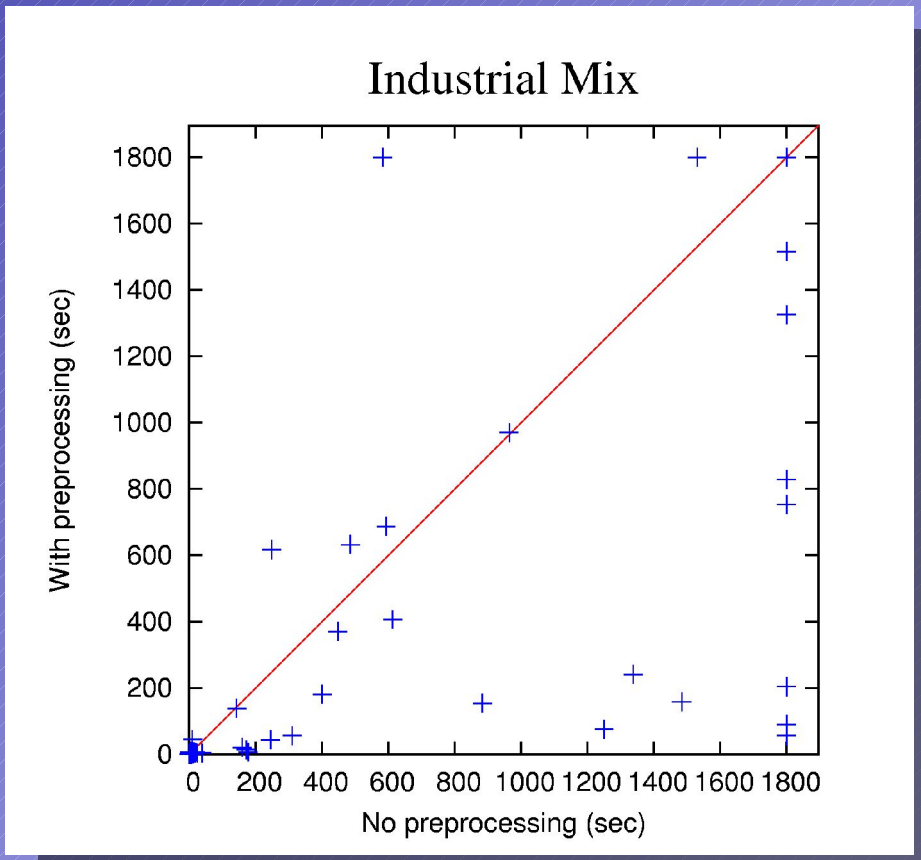
Preprocessing Time (log-scale)



Heads-up Comparison

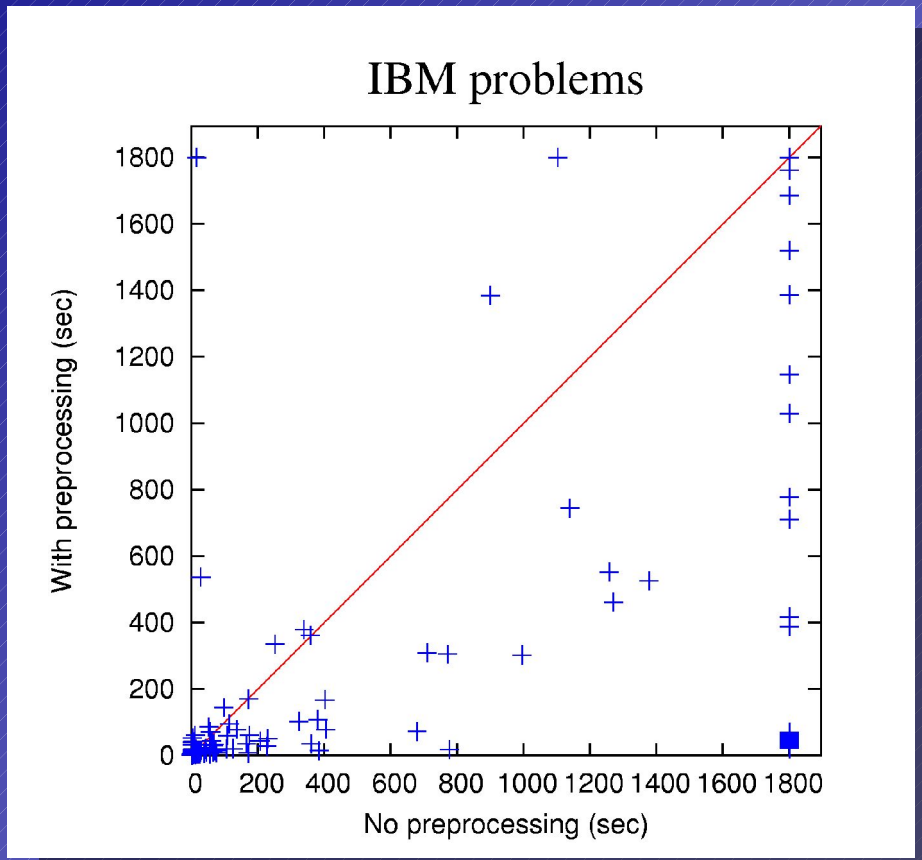
MiniSat

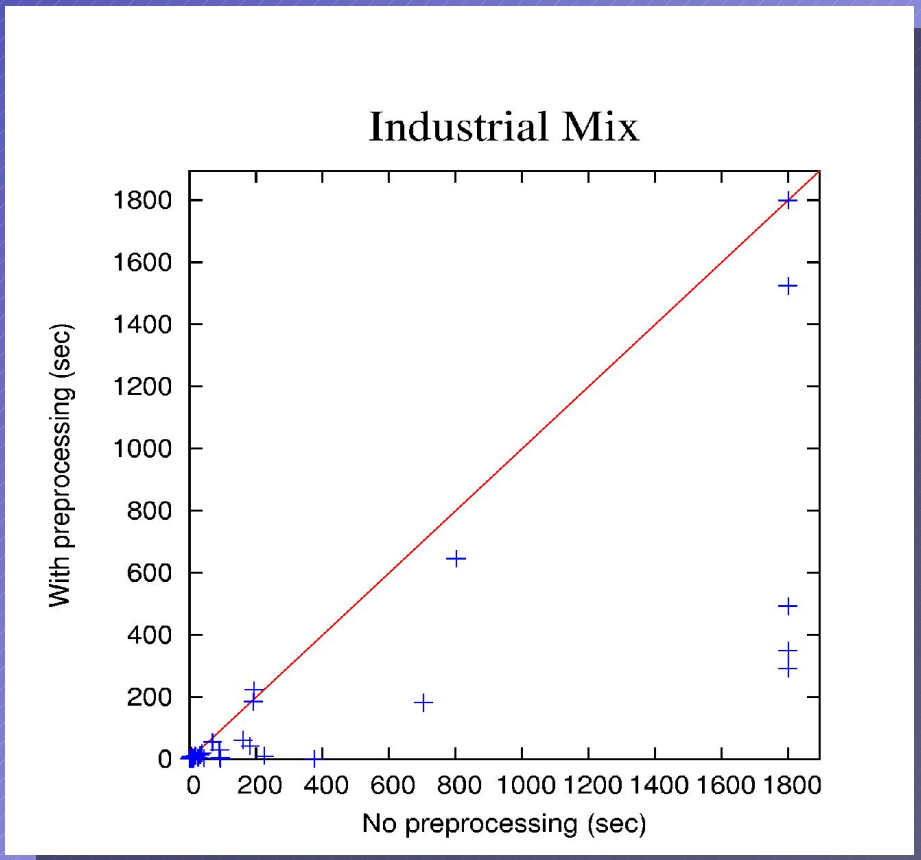




Heads-up Comparison

ZChaff





Conclusions

- Many SAT problems inefficiently encoded.
- Big performance gain can be achieved by improving the encoding.
- A solution at the SAT level can be efficient and effective.
- The presented method is completely resolution based, which facilitates proof-logging.
- Other rewrite rules and implementation tricks are of course possible. We show by construction that the general direction is worth looking at.

Future Work

- Look-ahead strategy
- Scheduling
  - of application of self-subsumption
  - of variable elimination
- Evaluating the differences and benefits of SAT preprocessing vs. a good clausifier.

Bibliography

[JS04] "Clause Form Conversions for Boolean Circuits"
Paul Jackson and Daniel Sheridan (SAT 2004)

[Vel05] "Efficient Translation of Boolean Formulas to CNF in Formal Verification of Microprocessors"
Miroslav N. Velev (Tech Report 2005)

[Biere04] "Expand and Resolve"
Armin Biere (SAT 2004)