

Parallel SAT Solving

Youssef Hamadi,

Microsoft Research, Cambridge, United Kingdom

École Polytechnique, Palaiseau, France

Outline

- Definitions, motivation
- Sequential SAT solving
- Parallel relaxation
- Parallel search
- Control-based clause sharing
- Summary and perspectives

The Propositional Satisfiability Problem (SAT)

- For a given Boolean formula f :
 - Find an assignment of the variables which evaluates to true
 - Prove that such assignment does not exist
 - Example: $f = (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_5) \dots$
- Established as the first NP-complete problem [Cook 1971]
 - SAT used to prove NP-completeness of other problems π : $\text{SAT} \leq \pi$
- Tremendous performance gains in last 10-15 years
 - Off-the-shelf SAT solver (black-box)
 - Routinely used to practically solve NP-complete problems with 10 of thousands variables and millions of clauses
 - $\pi \leq \text{SAT}$

Definitions

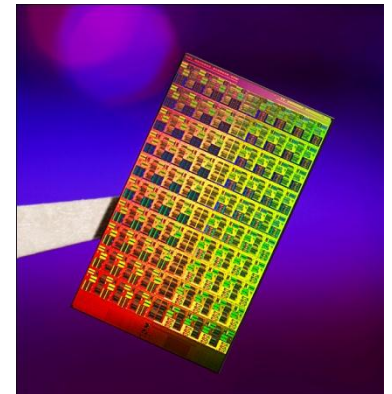
- **Parallel system**: parallel algorithm + parallel architecture
- **Scalability**: how well a parallel system takes advantage of increased computing resources
 - Definitions:
 - Sequential runtime T_s
 - Parallel runtime T_p (with p procs)
 - Speedup $S = T_s/T_p$
 - Efficiency $E = S/p$
 - Typical objective: divide the sequential runtime by the number of resources, i.e., $E \approx 1$

Definitions

- **Knowledge**: information generated during the execution of a parallel algorithm
- **Knowledge sharing**: mechanisms used to share the information. Tradeoffs:
 - Cost of sharing:
 - Ramp up time
 - Communication overhead
 - Cost of not sharing:
 - Redundant work
 - Task starvation

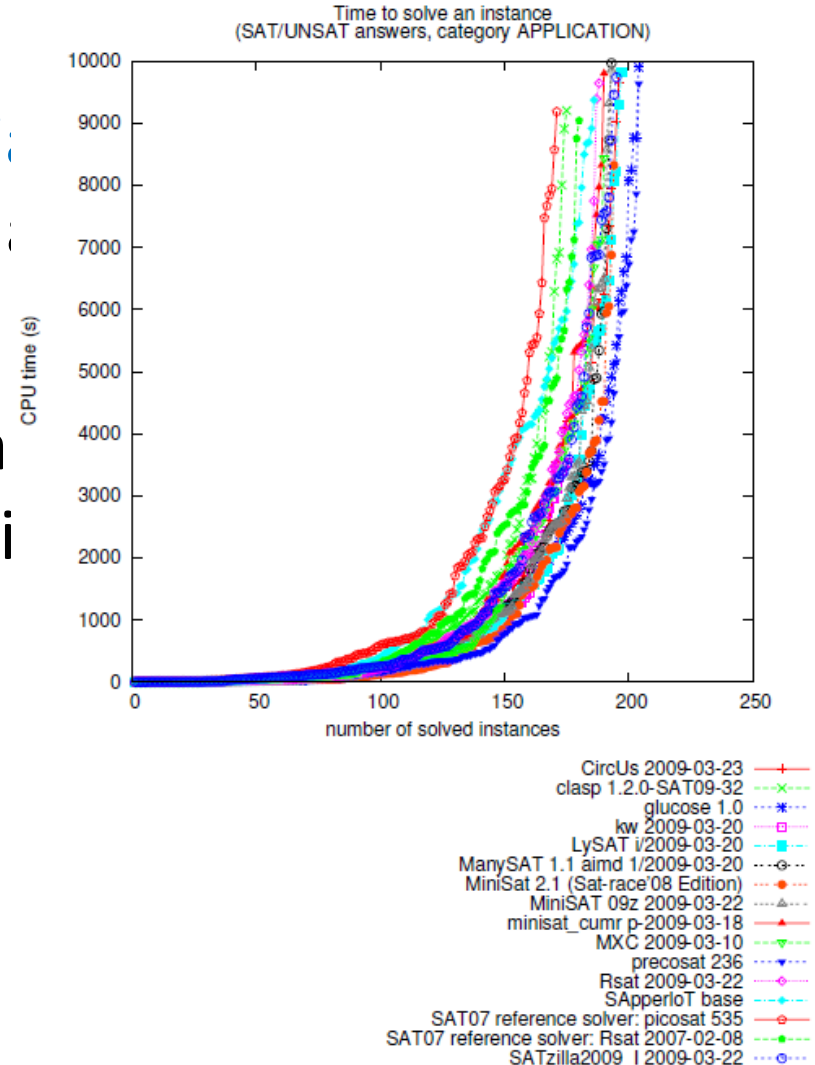
Motivation: technological

- Microprocessors architectures have changed
 - Clock frequency are stalling
 - **Thermal wall**: in many cases frequencies are being decreased to reduce power consumption
 - Transistor are still getting smaller
 - **Moore's law**: the number of transistors that can be inexpensively placed on an integrated circuit is increasing exponentially, doubling approximately every two years
- Consequences
 - Sequential software won't be getting faster
 - **Scalability** through more computing units
 - Exponential growth



Motivation: algorithmic

- State of the art **sequential** SAT solvers
improve (no orders of magnitude)
- SAT is applied to larger applications
which cannot be solved in reasonable time



PARALLEL RELAXATION

Parallel Relaxation

- Binary Unit Propagation
 - Unit-clause* rule: an unsatisfied clause is unit if it has exactly one unassigned literal
- 80-90% of solving time
- Operates *locally*
 - i.e., obvious candidate for parallel algorithm

Parallel Relaxation

- Worst case:

$$f = (x1 \vee x2) \wedge (x1 \vee \neg x2 \vee x3) \wedge (x1 \vee \neg x3 \vee x4) \wedge \dots$$

$$x1 = \text{false} \Rightarrow x2 = \text{true} \Rightarrow x3 = \text{true} \Rightarrow x4 = \text{true} \Rightarrow \dots$$

- Chain of successive (sequential) and unique implications
- BUP is *inherently* sequential

Parallel Relaxation

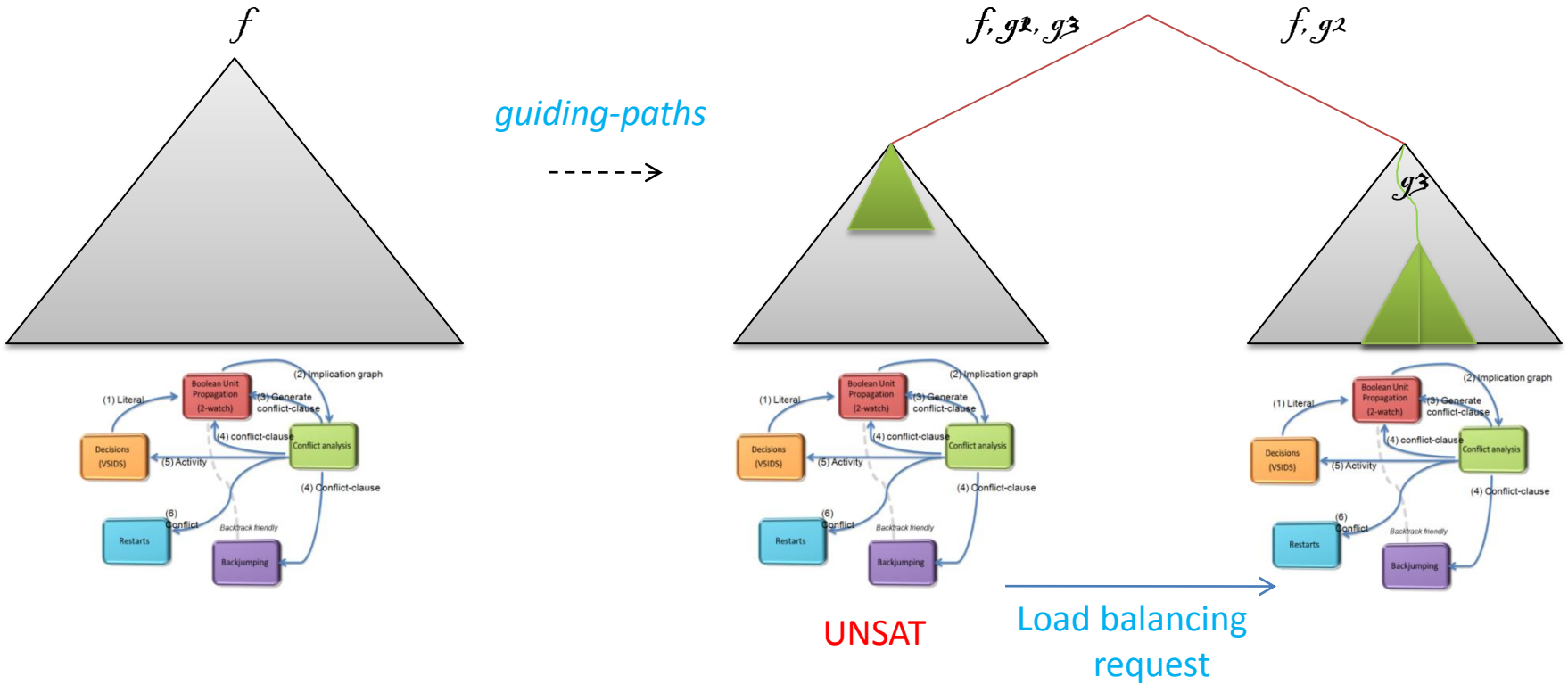
- Theorem[Kasif 90]: Parallel Relaxation (BUP) is log-space complete for P (i.e., $BUP \notin NC$)
- Parallel algorithm (polynomial number of resources) is unlikely to improve the sequential algorithm by much

PARALLEL SEARCH

Divide and conquer

Principles:

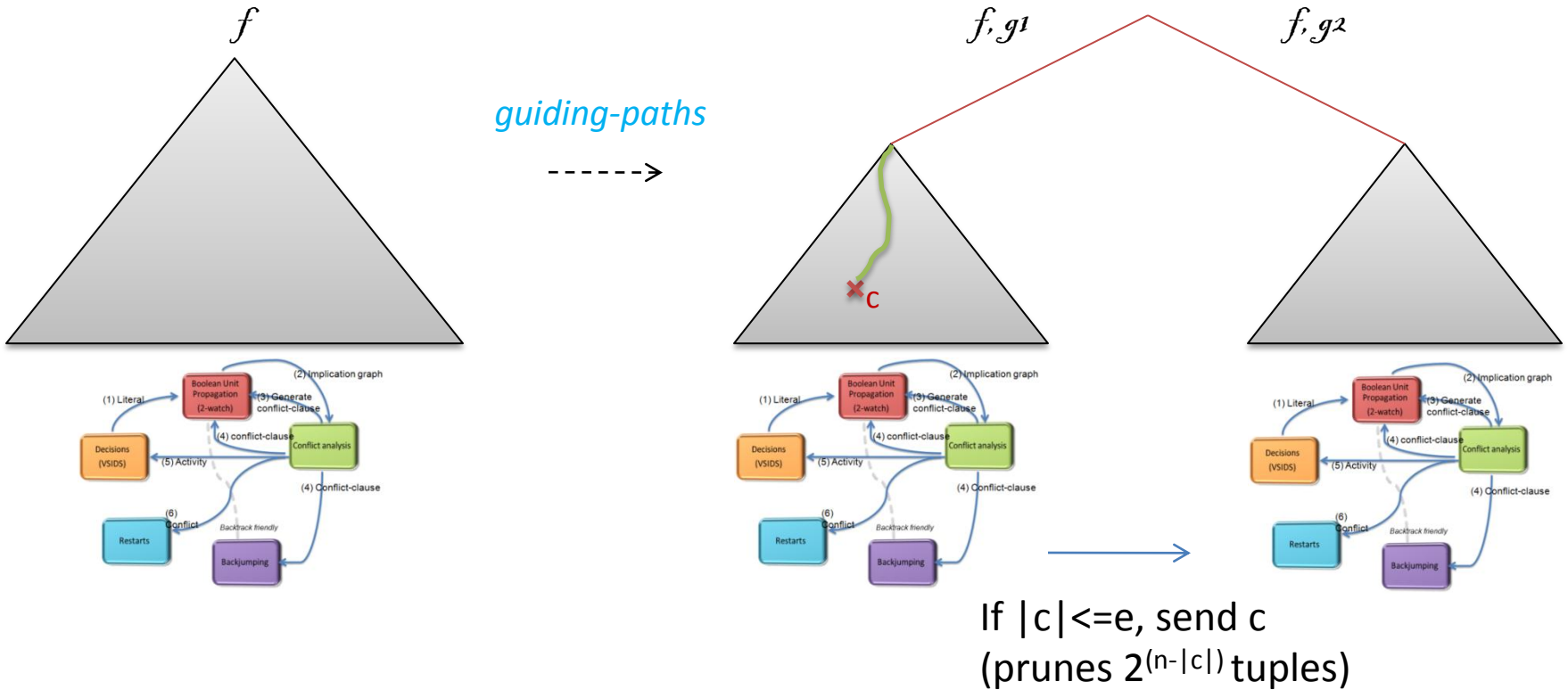
1. Allocate independent subspaces to different resources, organize load-balancing



Divide and conquer

Principles:

1. Allocate independent subspaces to different resources, organize load-balancing
2. Share learnt-clauses



Divide-and-conquer: algorithms

```
SlaveDPLL() {
1:get and enforce guiding-path;
  limit = c;
  while(!end) {
    <import foreign-units-clauses>;
    while(#conflicts < limit && !end){
      <import foreign-clauses>;
      lit = decide();
      if(!lit)
        end = true;
        SAT = true;
      if(!BUP(lit)){
        cl = conflict-analysis();
        if(!cl) goto 1;
        export cl;
        #conflicts++;
      }
    }
    undoDecisions();
    increase(limit);
  }
}
```

```
MasterDPLL() {
  produce initial guiding-paths;
  end = false;
  while(!end) {
    if(guiding-path-required())
      if(!guiding-path())
        end = true;
        SAT = false;
    <SlaveDPLL>
  }
}
```

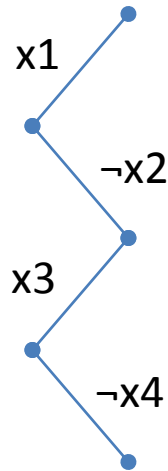
4 cases

end, SAT: shared memory variables

Integration of foreign-clauses

Decisions

e.g.,

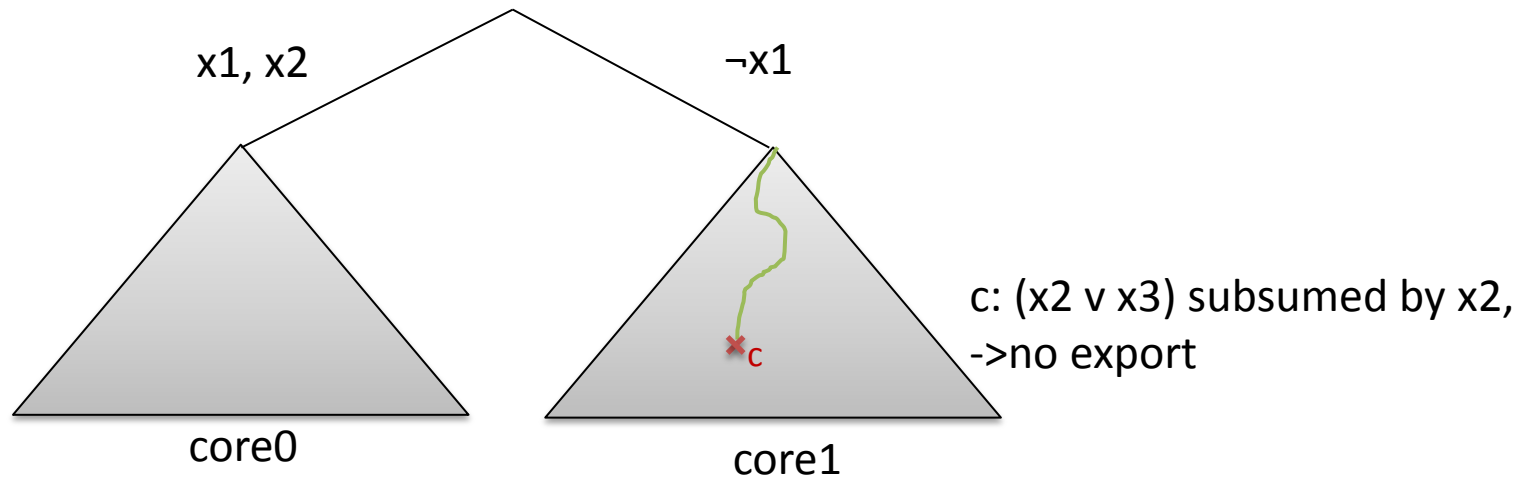


Foreign-clauses

1. False, e.g., $(\neg x1 \vee x2)$
conflict-analysis
2. Unit, e.g., $(\neg x1 \vee \mathbf{x5})$
unit propagation
3. Satisfied, e.g., $(x5 \vee x6 \vee \mathbf{x3})$
watch (last satisfied..)
4. Otherwise,
watch any pair of literals

Restricting clause sharing

- Static rule: If $|c| \leq e$, send c
- Take advantage of guiding-paths
 $|c| \leq e$, but do not export if c subsumed by a guiding-path



Divide-and-conquer in SAT

	Base algorithm	Parallel architecture	Knowledge sharing
Psato [Zhang et al. 1996]	Sato	workstations	Load-balancing
[Bohm et al. 1996]	ad-hoc	workstations	Load-balancing
Gradsat [Chrabakh et al. 2003]	zChaff	workstations	Load-balancing, clause sharing
[Blochinger et al. 2003]	zChaff	workstations	Load-balancing, restricted clause sharing
MiraXT [Lewis et al. 2007]	Minisat	multicore	Load-balancing, systematic clause sharing
Pminisat [Chu et al. 2008]	Minisat	multicore	Load-balancing, clause sharing generalized

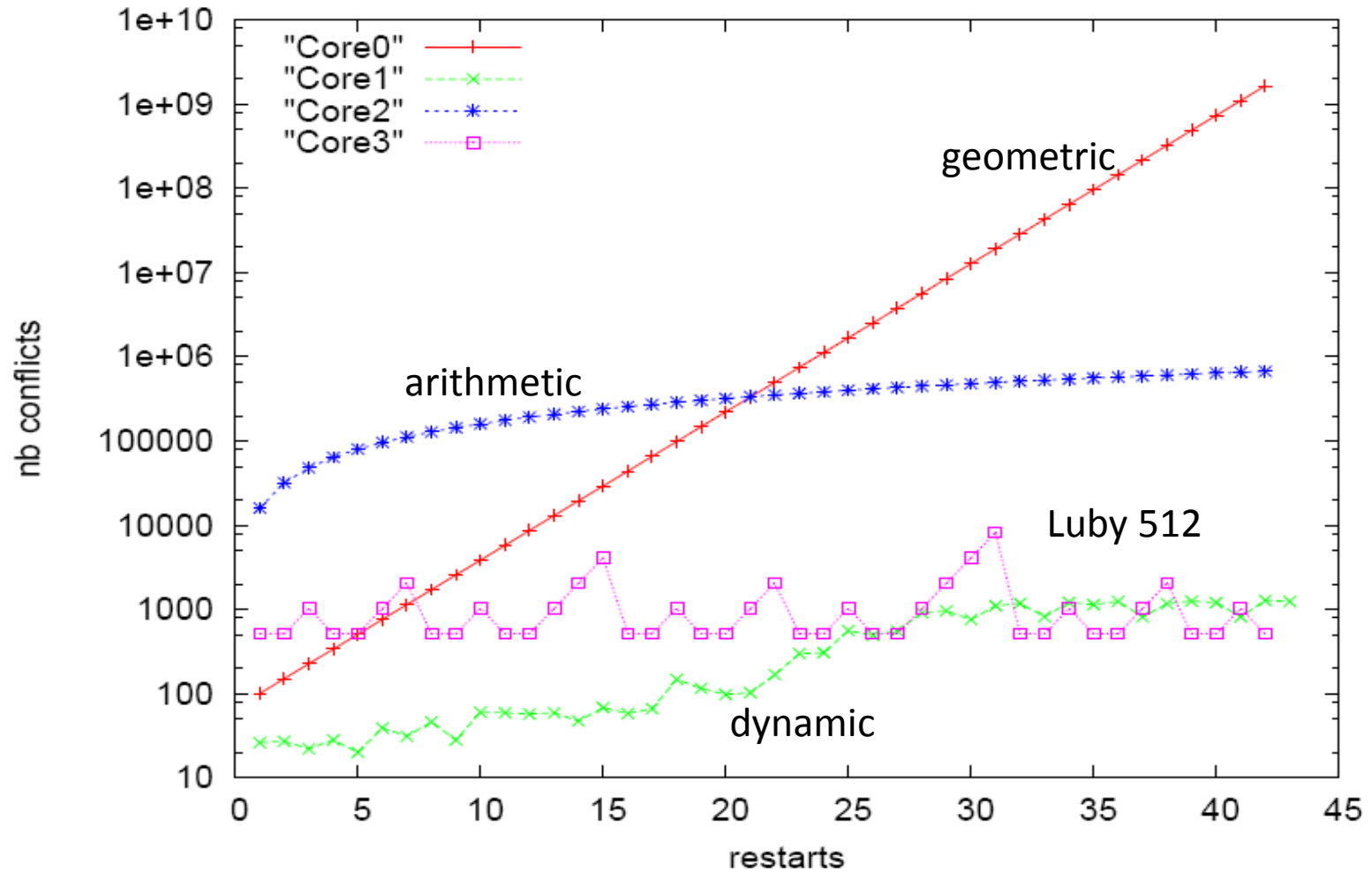
Portfolio of solvers

- Portfolio approach: let several *differentiated* but related DPLLs **compete** and **cooperate** to be the first to solve a given instance
- Tradeoff:
 - Cover the space of search strategies
 - Exchange useful information
- State-of-the-art:
 - Plingeling [Biere 2010], Antom [Schubert et al. 2010], SArTagnan [Kottler 2010], etc.
 - ManySAT [Hamadi, Jabbour, Sais 2008]

ManySAT: internals

Strategies	Core 0	Core 1	Core 2	Core 3
Restart	Geometric $x_1 = 100$ $x_i = 1.5 \times x_{i-1}$	Dynamic (Fast) $x_1 = 100, x_2 = 100$ $x_i = f(y_{i-1}, y_i), i > 2$ if $y_i > y_{i-1}$ $f(y_{i-1}, y_i) =$ $\frac{\alpha}{y_i} \times \cos(1 + \frac{y_{i-1}}{y_i}) $ else $f(y_{i-1}, y_i) =$ $\frac{\alpha}{y_i} \times \cos(1 + \frac{y_i}{y_{i-1}}) $ $\alpha = 1200$	Arithmetic $x_1 = 16000$ $x_i = x_{i-1} + 16000$	Luby 512
Heuristic	VSIDS (3% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)
Polarity	if $\#occ(l) > \#occ(\neg l)$ $l = true$ else $l = false$	Progress saving	false	Progress saving
Learning	CDCL (extended [1])	CDCL	CDCL	CDCL (extended [1])
Cl. sharing	size 8	size 8	size 8	size 8

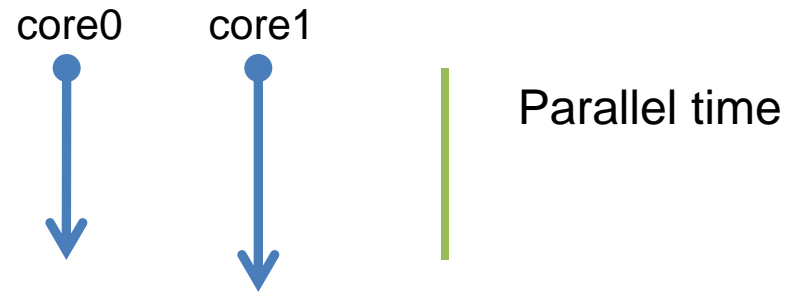
ManySAT: restart policies



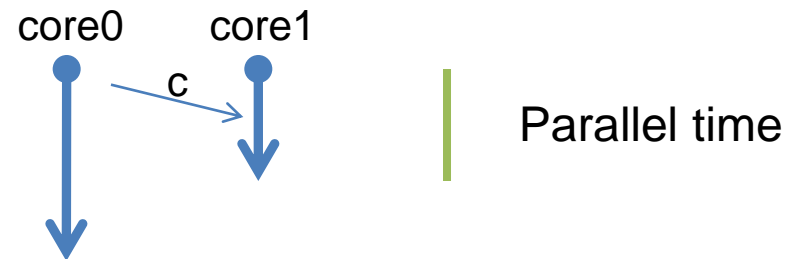
Portfolio approach

- Knowledge sharing: conflict-clause

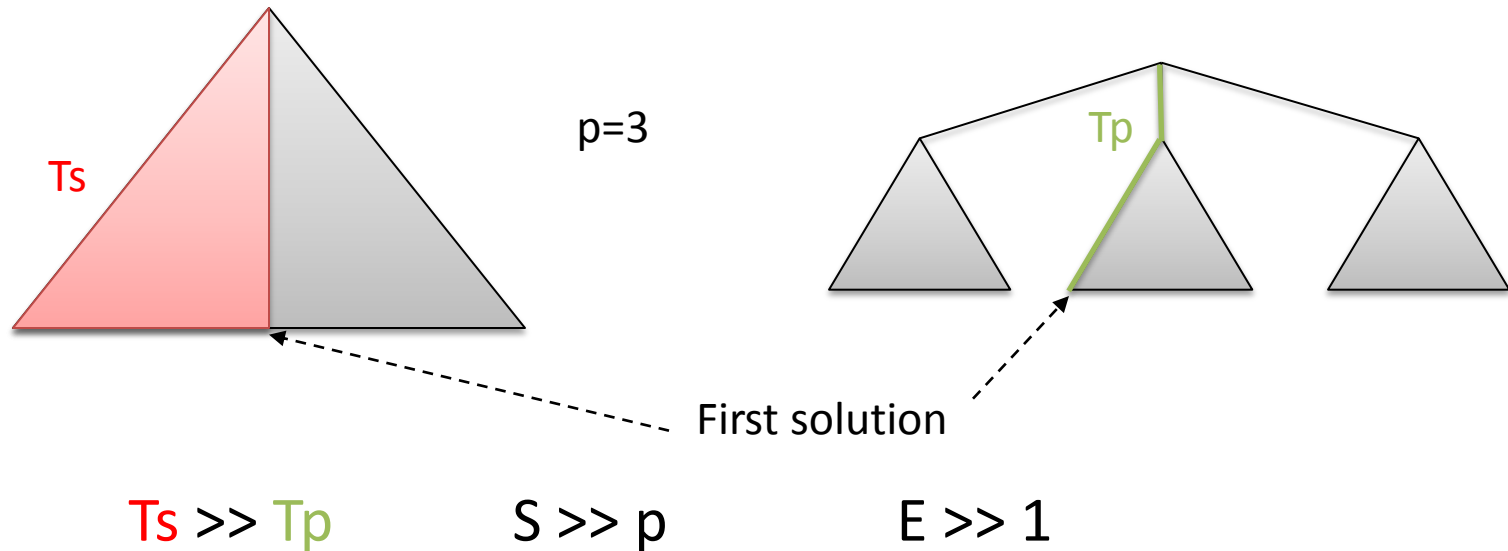
- Without: as good as the best



- With: better than the best



Theoretical Performance



- “Speed-up anomalies in parallel tree search”, first reported identification circa 1975 [Pruul 88]
- [Rao et al. 93]: “... sequential DFS is sub-optimal...”
-> Interleaved DFS (sequential) [Meseguer 97]

Practical Performance

- SAT-Race 2008
 - 100 industrial problems, 4 cores, 15min timeout
 - **Absolute speed-up** (vs. Minisat 2.1, best 2008 Sequential)



	ManySAT	pMinisat	MiraXT
Solved	90	85	73
by SAT/UNSAT	45/45	44/41	43/30

Practical Performance

- SAT-Race 2008
 - 100 industrial problems, 4 cores, 15min timeout
 - **Absolute speed-up** (vs. Minisat 2.1, best 2008 Sequential)



	ManySAT	pMinisat	MiraXT
Solved	90	85	73
by SAT/UNSAT	45/45	44/41	43/30
Average speed-up	6.02	3.10	1.83
by SAT/UNSAT	8.84/3.14	4.00/2.18	1.85/1.81
Minimal speed-up	0.25	0.34	0.04
by SAT/UNSAT	0.25/0.76	0.34/0.46	0.04/0.74
Maximal speed-up	250.17	26.47	7.56
by SAT/UNSAT	250.17/4.74	26.47/10.57	7.56/4.26

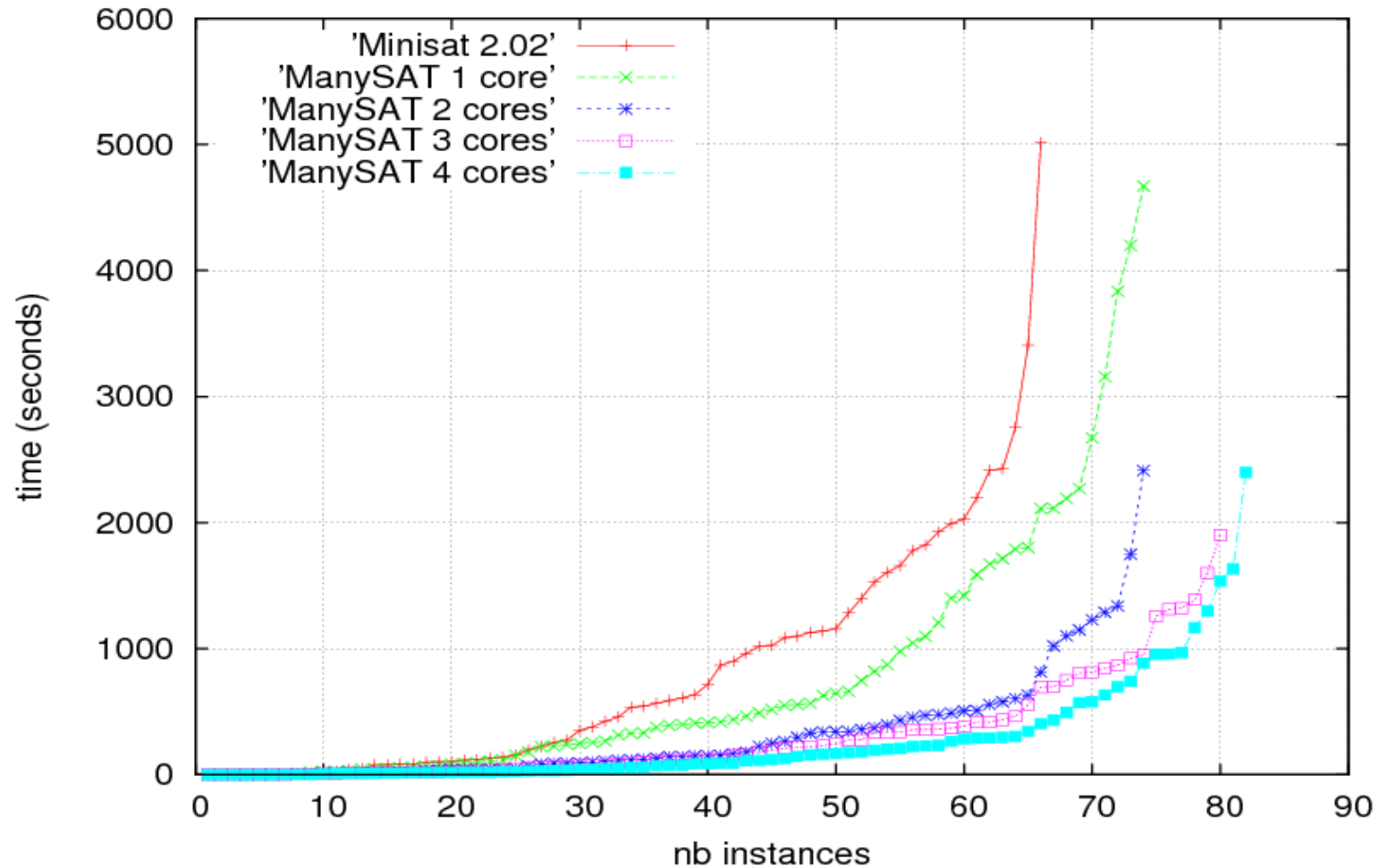
Practical Performance

- SAT-Race 2008
 - 100 industrial problems, 4 cores, 15min timeout
 - **Absolute speed-up** (vs. Minisat 2.1, best 2008 Sequential)



	ManySAT	pMinisat	MiraXT
Solved	90	85	73
by SAT/UNSAT	45/45	44/41	43/30
Average speed-up	6.02	3.10	1.83
by SAT/UNSAT	8.84/3.14	4.00/2.18	1.85/1.81
Minimal speed-up	0.25	0.34	0.04
by SAT/UNSAT	0.25/0.76	0.34/0.46	0.04/0.74
Maximal speed-up	250.17	26.47	7.56
by SAT/UNSAT	250.17/4.74	26.47/10.57	7.56/4.26
Runtime variation	13.7%	14.7%	15.2%
by SAT/UNSAT	22.2%/5.5%	23.1%/5.7%	19.5%/9.7%

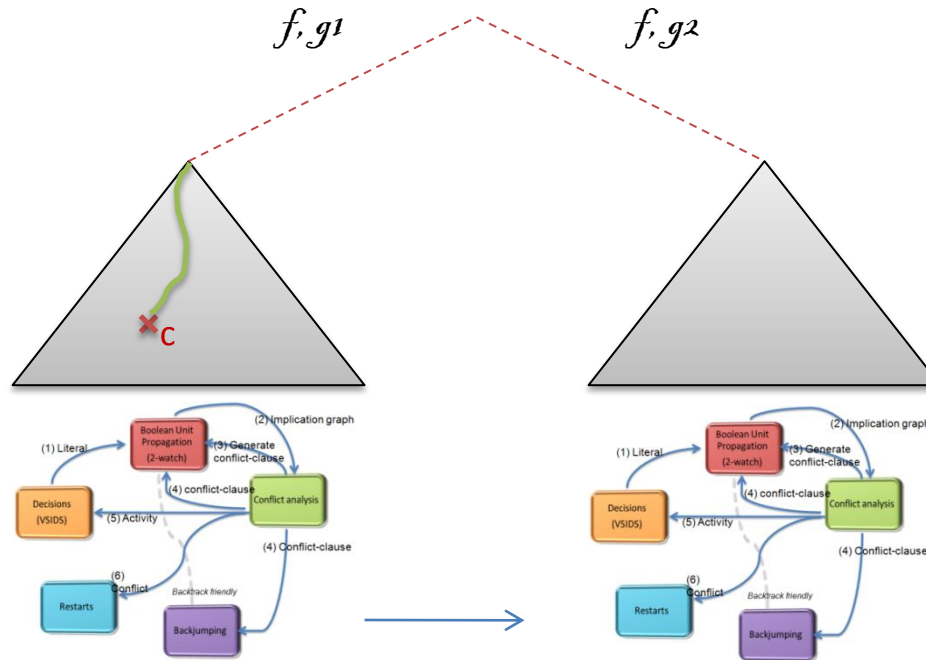
ManySAT: performance /core



CONTROL-BASED CLAUSE SHARING

Control-based Clause Sharing in Parallel SAT Solving, Y. Hamadi, S. Jabbour, and L. Sais, Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09), July 2009, Pasadena, USA

Clause sharing



If $|c| \leq e$, send c
 (prunes $2^{(n-|c|)}$ tuples)

Problem 1: offline tuning

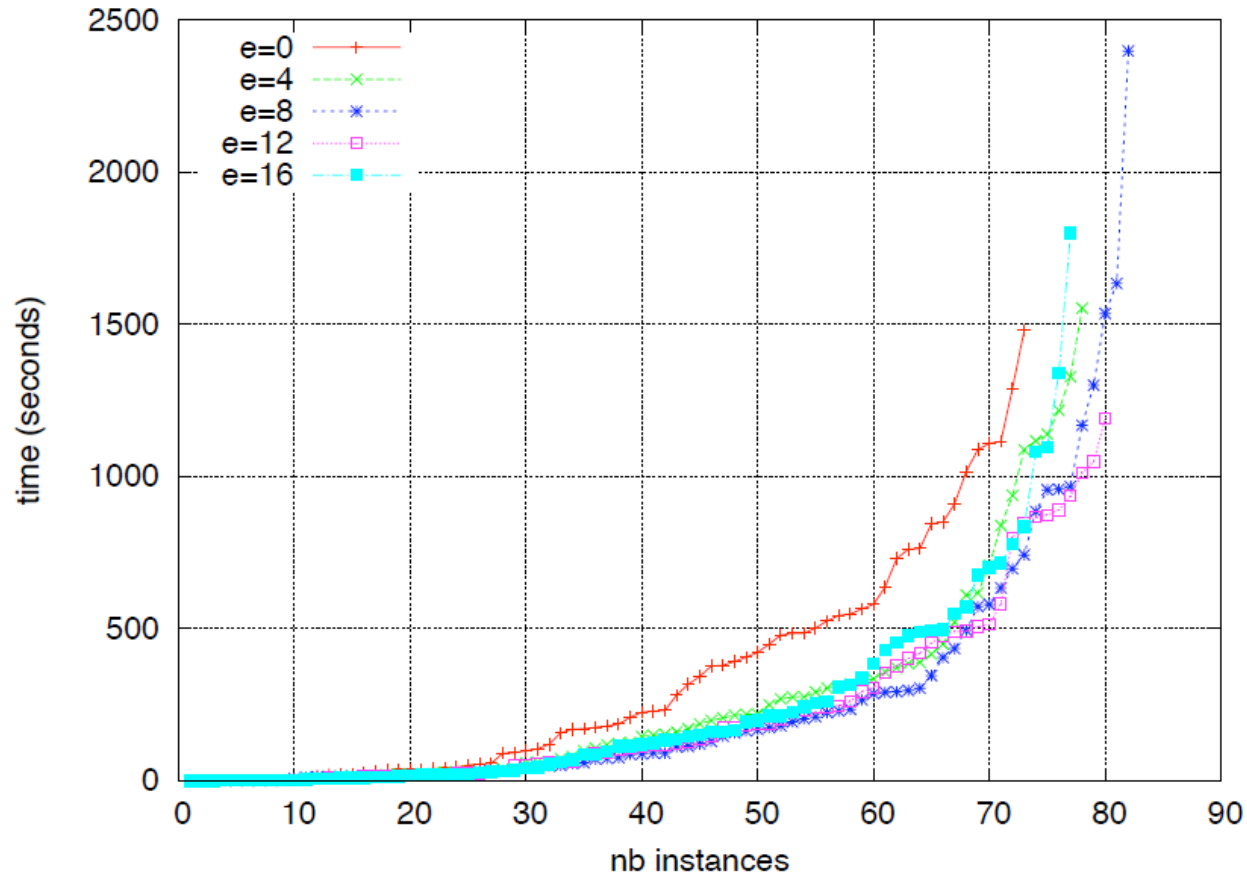
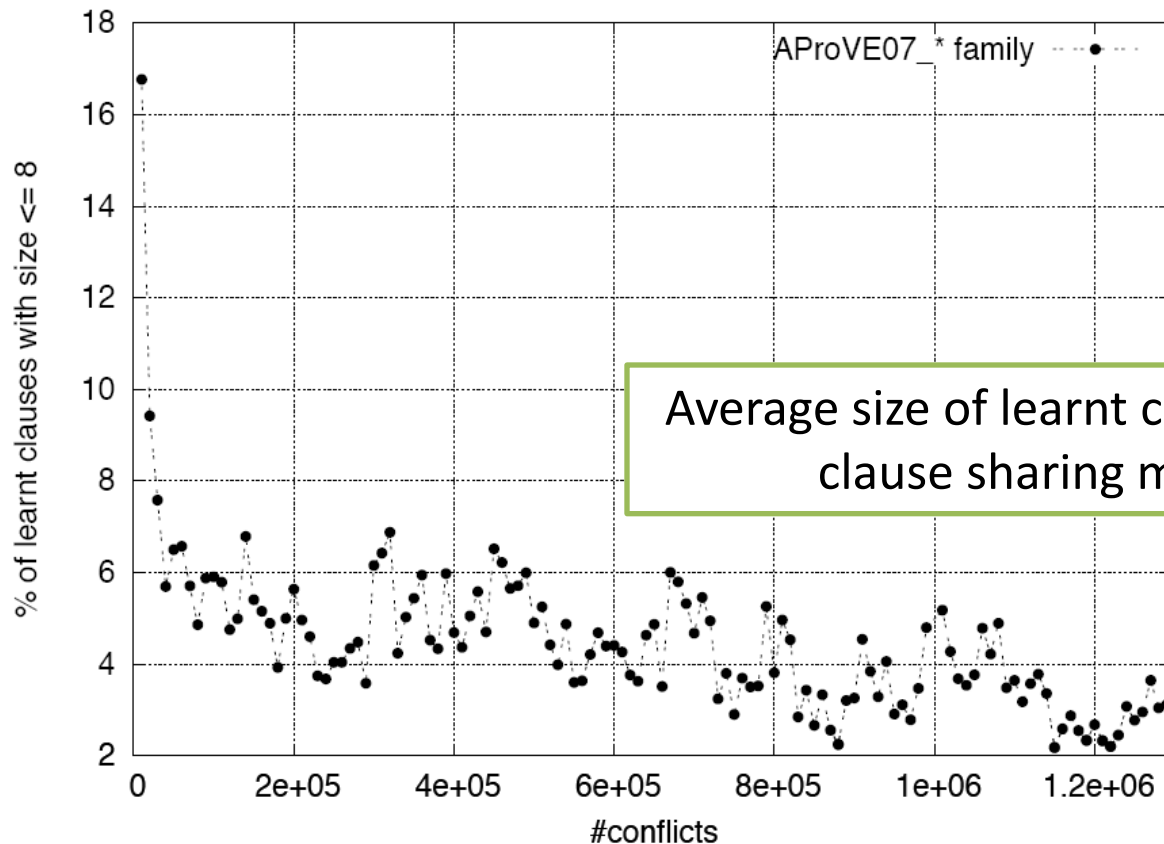


Figure 3. SAT-Race 2008: different limits for clause sharing

Problem 2: saturation

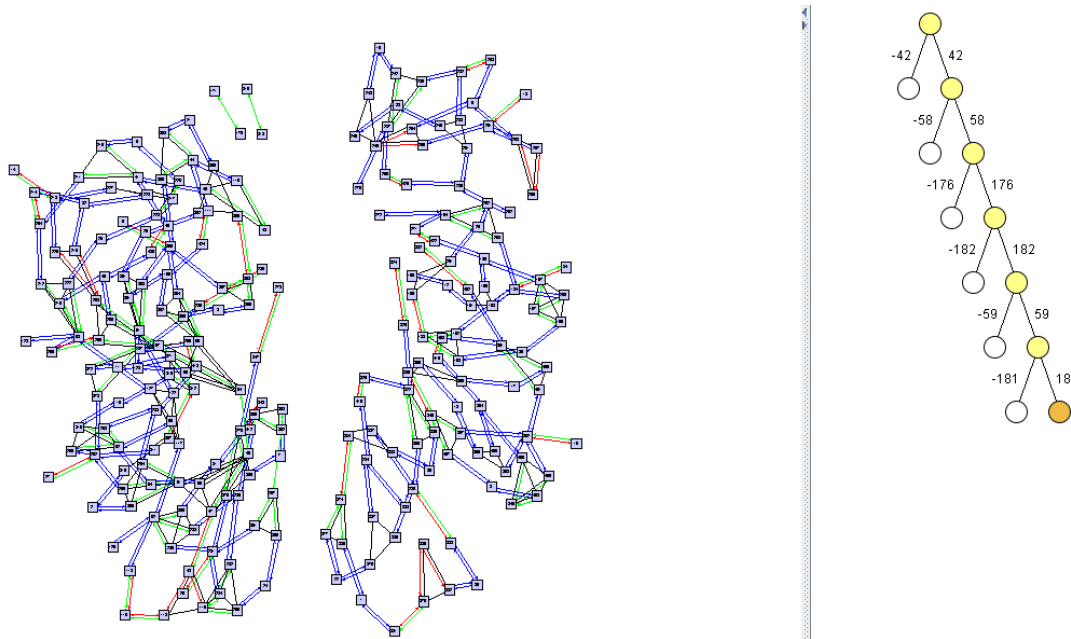
- Simple experiment with Minisat 2.0 (sequential)



Average size of learnt clauses is raising:
clause sharing might halt

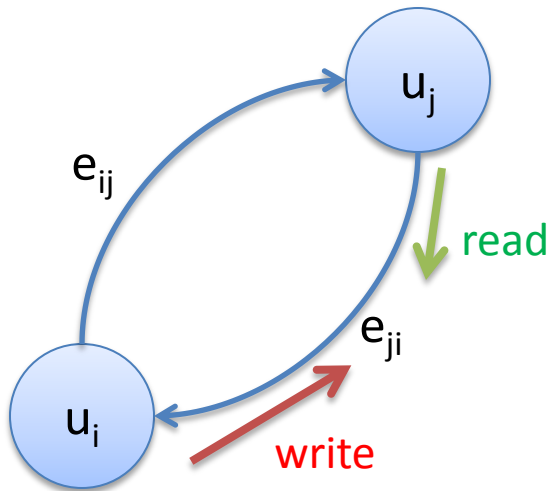
Problem 3: relevance

- Exchange between unrelated search efforts:



[DPVis, Sinz 05]

Dynamic limits



1. Pairwise size limits e_{ij} to control clause sharing from i to j
2. Each unit performs (lock-free) periodic revisions of incoming limits

Two objectives:

1. Maintain a **throughput T**. Solves problems (1), (2):

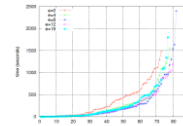
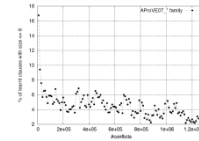
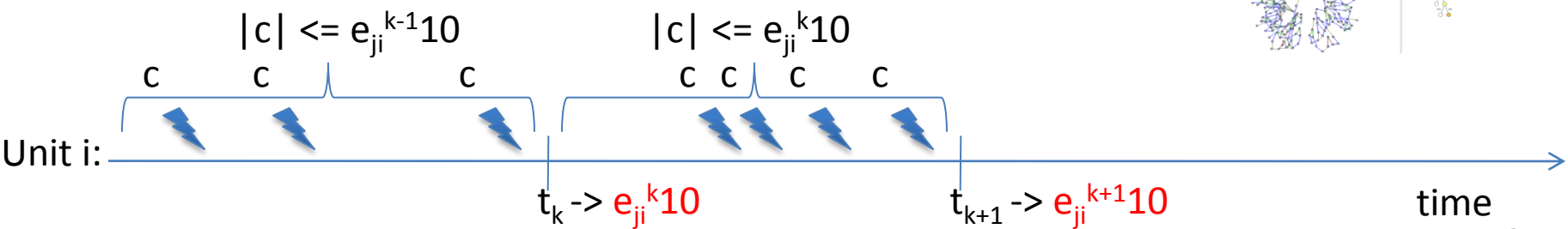
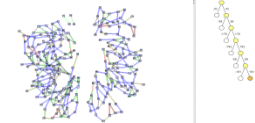


Figure 9: SAT over 200 different benchmarks for clause density



2. Maintain a **throughput T of a given Quality Q**. Solves (3):

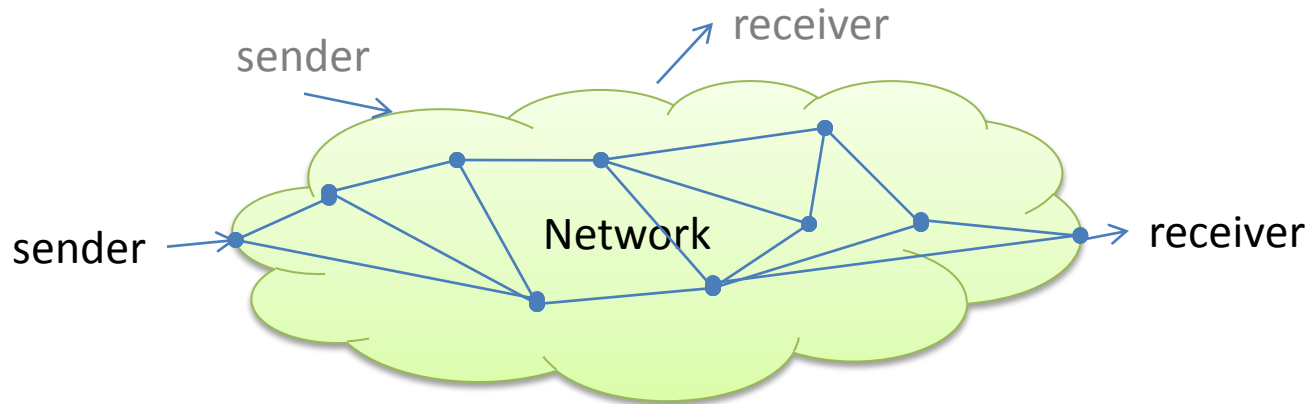


Objective 1: Maintain a throughput T

- Throughput T is a number of foreign clauses received in each time interval
 - Time interval = α conflicts
 - Typically, $T = \alpha/c$
- Unit i , at step t_k :
 - R_k is the set of foreign clauses received during t_{k-1}
 - If $|R_k| < T$, uniform increase of e_{ji}^k limits
 - If $|R_k| > T$, uniform decrease of e_{ji}^k limits
- How do we update the limits?

TCP Congestion Avoidance

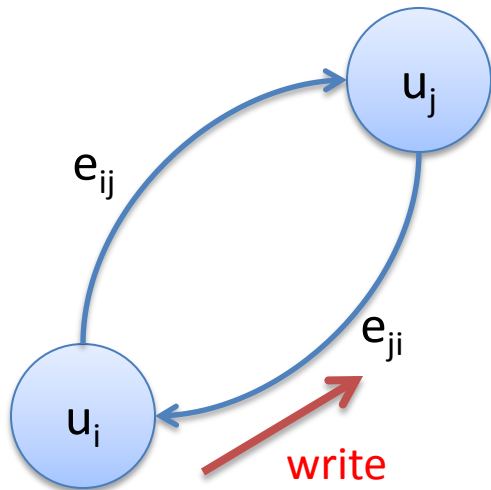
- Problem: guess the available bandwidth, i.e., find the correct communication rate w



- Additive Increase Multiplicative Decrease (AIMD):
 - Slow increase as long as no packet loss: $w = w + b/w$
 - i.e., probe for available bandwidth
 - Exponential decrease if a loss is encountered: $w = w - a*w$
 - i.e., congestion: quick decrease for faster recovery

Additive Increase Multiplicative Decrease (AIMD)

- Clause sharing: an increase of the limits can generate a very large number of incoming clauses.
 - Slow increase, as long as T not met
 - Exponential decrease, if T is met



$$aimd_T(R_i^k) \{$$

$$\forall j | 0 \leq j < n, j \neq i$$

$$e_{j \rightarrow i}^{k+1} = \begin{cases} e_{j \rightarrow i}^k + \frac{b}{e_{j \rightarrow i}^k}, & \text{if } (|R_i^k| < T) \\ e_{j \rightarrow i}^k - a \times e_{j \rightarrow i}^k, & \text{if } (|R_i^k| > T) \end{cases}$$

Objective 2: Maintain a throughput T of quality Q

- VSIDS heuristic: unassigned variables with the highest activity are related to the future evolution of the search process.

- Def.

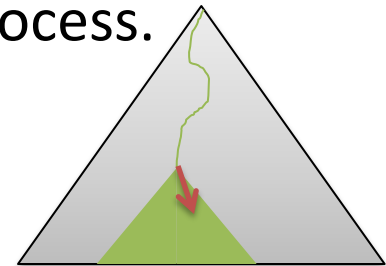
- Maximum VSIDS activity: A_i^{max}
- Set of active literals of a foreign clause c :

$$\mathcal{L}_{\mathcal{A}_i}(c) = \{x/x \in c \text{ s.t. } \mathcal{A}_i(x) \geq \frac{A_i^{max}}{2}\}$$

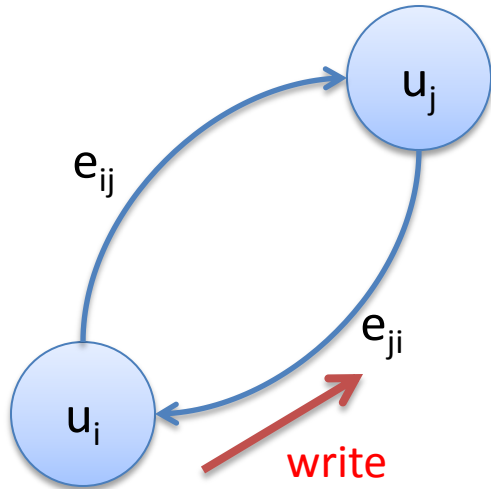
- Set of clauses received from j with at least Q active literals:

$$\mathcal{P}_{j \rightarrow i}^k = \{c/c \in \Delta_{j \rightarrow i}^k \text{ s.t. } |\mathcal{L}_{\mathcal{A}_i}(c)| \geq Q\}$$

- Quality of clauses received from j at step k : $Q_{j \rightarrow i}^k = \frac{|\mathcal{P}_{j \rightarrow i}^k| + 1}{|\Delta_{j \rightarrow i}^k| + 1}$



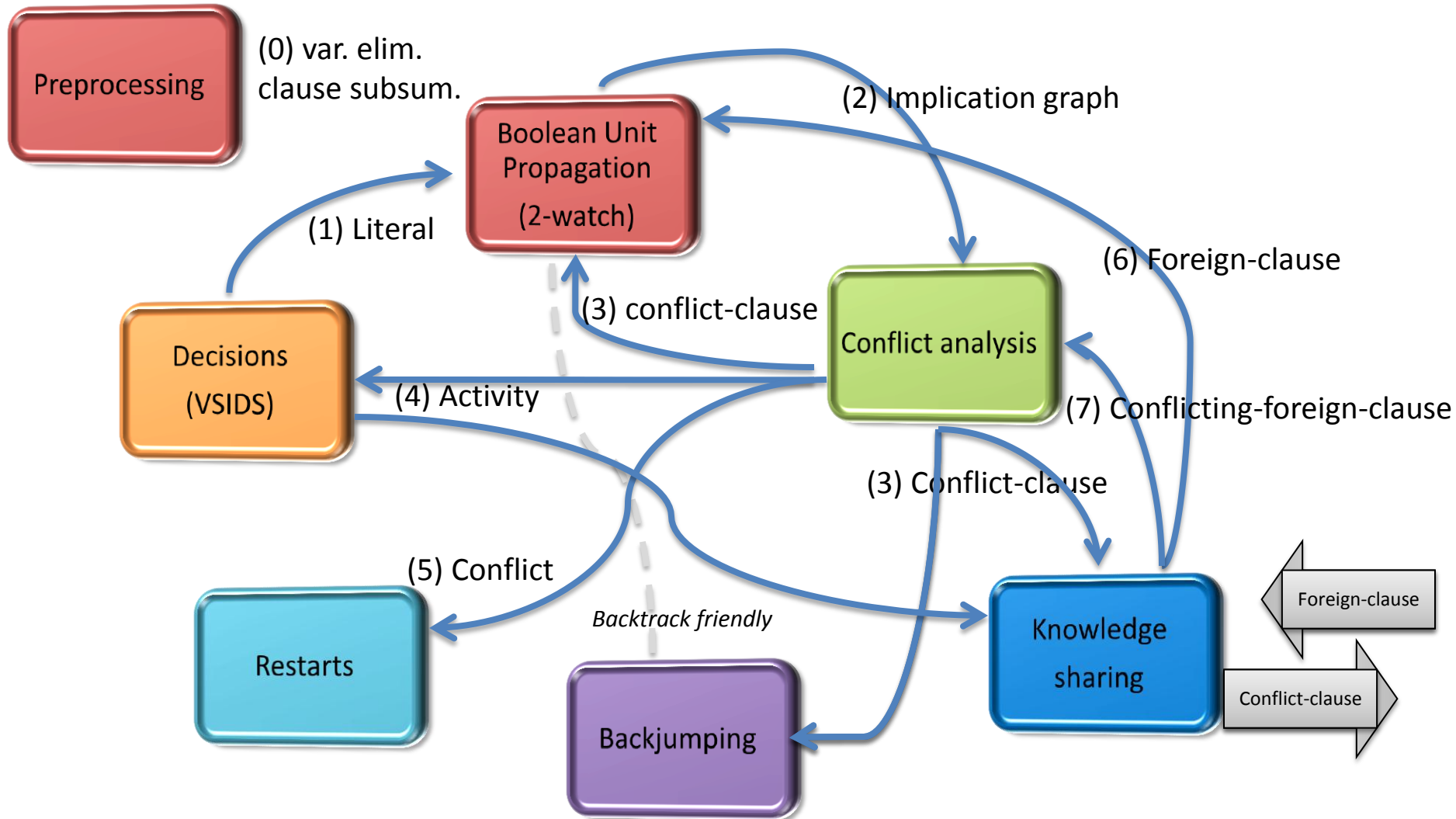
Maintain a throughput T of quality Q



$$\begin{aligned}
 & \text{aimd}TQ(R_i^k) \{ \\
 & \quad \forall j | 0 \leq j < n, j \neq i \\
 & e_{j \rightarrow i}^{k+1} = \begin{cases} e_{j \rightarrow i}^k + \left(\frac{Q_{j \rightarrow i}^k}{100}\right) \times \frac{b}{e_{j \rightarrow i}^k}, \text{ if } (|R_i^k| < T) \\ e_{j \rightarrow i}^k - \left(1 - \frac{Q_{j \rightarrow i}^k}{100}\right) \times a \times e_{j \rightarrow i}^k, \text{ if } (|R_i^k| > T) \end{cases}
 \end{aligned}$$

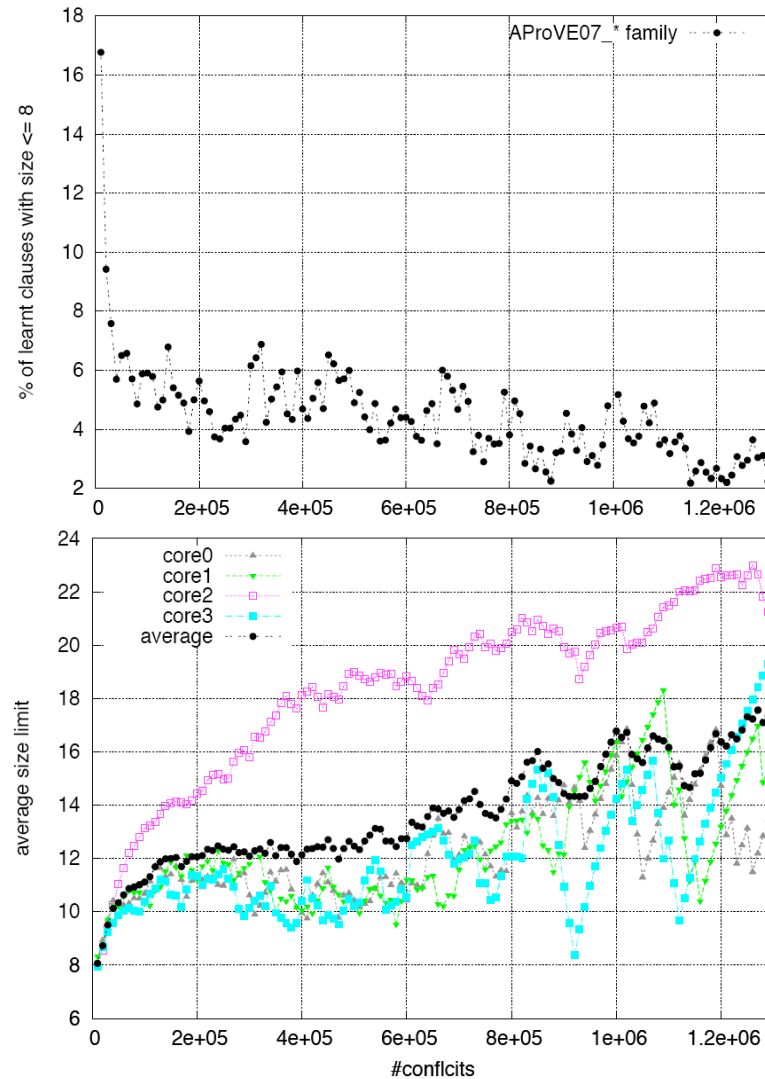
- Increase/Decrease:
 - Favour units which give *related* clauses

Parallel SAT Solving



EVALUATION

Problem 2: saturation



Performance on Industrial problems

family/instance	#inst	ManySAT e=8	
		#Solved	time(s)
ibm_*	20	19	204
manol_*	10	10	117
mizh_*	10	6	762
post_*	10	9	325
velev_*	10	8	585
een_*	5	5	2
simon_*	5	5	111
bmc_*	4	4	7
gold_*	4	1	1160
anbul_*	3	2	742
babic_*	3	3	2
schup_*	3	3	129
fuhs_*	2	2	90
grieu_*	2	1	783
narain_*	2	1	786
palac_*	2	2	20
aloul-chnl11-13	1	0	1500
jarvi-eq-atree-9	1	1	70
marijn-philips	1	0	1500
maris-s03-gripper11	1	1	11
vange-col-abb313gpia-9-c	1	0	1500
Total/(average)	100	83	10406

Table 1: SAT-Race 2008, industrial problems

Performance on Industrial problems

family/instance	#inst	ManySAT e=8		ManySAT aimdT		
		#Solved	time(s)	#Solved	time(s)	\bar{e}
ibm_*	20	19	204	19	218	7
manol_*	10	10	117	10	117	8
mizh_*	10	6	762	7	746	6
post_*	10	9	325	9	316	7
velev_*	10	8	585	8	448	5
een_*	5	5	2	5	2	8
simon_*	5	5	111	5	84	10
bmc_*	4	4	7	4	7	7
gold_*	4	1	1160	1	1103	12
anbul_*	3	2	742	3	211	11
babic_*	3	3	2	3	2	8
schup_*	3	3	129	3	120	5
fuhs_*	2	2	90	2	59	11
grieu_*	2	1	783	1	750	8
narain_*	2	1	786	1	776	8
palac_*	2	2	20	2	8	3
aloul-chnl11-13	1	0	1500	0	1500	11
jarvi-eq-atree-9	1	1	70	1	69	25
marijn-philips	1	0	1500	1	1133	34
maris-s03-gripper11	1	1	11	1	11	10
vange-col-abb313gpia-9-c	1	0	1500	0	1500	12
Total/(average)	100	83	10406	86	9180	(10.28)

Table 1: SAT-Race 2008, industrial problems

Performance on Industrial problems

family/instance	#inst	ManySAT e=8		ManySAT aimdT			ManySAT aimdTQ		
		#Solved	time(s)	#Solved	time(s)	\bar{e}	#Solved	time(s)	\bar{e}
ibm_*	20	19	204	19	218	7	19	286	6
manol_*	10	10	117	10	117	8	10	205	7
mizh_*	10	6	762	7	746	6	10	441	5
post_*	10	9	325	9	316	7	9	375	7
velev_*	10	8	585	8	448	5	8	517	7
een_*	5	5	2	5	2	8	5	2	7
simon_*	5	5	111	5	84	10	5	59	9
bmc_*	4	4	7	4	7	7	4	6	9
gold_*	4	1	1160	1	1103	12	1	1159	12
anbul_*	3	2	742	3	211	11	3	689	11
babic_*	3	3	2	3	2	8	3	2	8
schup_*	3	3	129	3	120	5	3	160	5
fuhs_*	2	2	90	2	59	11	2	77	10
grieu_*	2	1	783	1	750	8	1	750	8
narain_*	2	1	786	1	776	8	1	792	8
palac_*	2	2	20	2	8	3	2	54	7
aloul-chnl11-13	1	0	1500	0	1500	11	0	1500	10
jarvi-eq-atree-9	1	1	70	1	69	25	1	43	17
marijn-philips	1	0	1500	1	1133	34	1	1132	29
maris-s03-gripper11	1	1	11	1	11	10	1	11	8
vange-col-abb313gpia-9-c	1	0	1500	0	1500	12	0	1500	12
Total/(average)	100	83	10406	86	9180	(10.28)	89	9760	(9.61)

Table 1: SAT-Race 2008, industrial problems

Summary

- Technological and algorithmic shift to multicores
 - Unlikely speed-up for sequential software
- Parallel SAT effectively *extends* the modern DPLL architecture
 - Clause-learning -> clause-sharing
- Portfolio search outperforms divide-and-conquer approaches
 - Identification of important variables to split the space
 - Overhead caused by load-balancing
- Performance improvements, super linear speedups on average 😊
 - Runtime variability 😞
 - Solution/core variability 😞
- Improved performance through *controlled* knowledge sharing

Perspectives

- Intensification mechanisms
 - *Diversification and Intensification in Parallel SAT Solving*, L. Guo, Y. Hamadi, S. Jabbour, and L. Sais, CP'10
- SAT is the workhorse of important formalisms, QBF, SMT
 - *A Concurrent Portfolio Approach to SMT Solving*, C. M. Wintersteiger, Y. Hamadi, L. M. de Moura, CAV'09
- Deterministic algorithms, e.g., synchronized sharing
- Think out-of-the-box: new algorithmic architecture
 - integrated variables elimination
 - decomposition of the formula
 - etc.