Portfolio-based parallel solvers

Lakhdar Saïs[1]

Joint work with Youssef Hamadi[2], Saïd Jabbour[1],

(1) CRIL - CNRS, Université Lille Nord de France

(2) Microsoft Research, Cambridge UK.

Outline

- Propositional Satisfiability Problem (SAT)
- Modern SAT solvers
- Parallel SAT
  - Divide and conquer based parallel solvers
  - **Portfolio based Parallel solvers [Hamadi et al. JSAT'09]**
- **Control-based clause sharing [Hamadi et al. IJCAI'09]**
- Conclusion/future work.

Propositional Satisfiability Problem(SAT)

- Given a CNF formula *F*

  $(a \vee b \vee c) (\neg a \vee b) (\neg b \vee c) (\neg c \vee a)$

- F admits a model ?
  - F is Satisfiable : a = true, b = true, c = true
  - $F \wedge (\neg a \vee \neg b \vee \neg c)$ is Unsatisfiable

- **Bad news**: SAT is NP-Complete [Cook 71]
- **Good news**: Modern SAT solvers can solve instances with million of variables and clauses in few seconds

  => Widely used in formal verification, planning, bioinformatics, cryptography, ...

An example

Instance name: post-cbmc-zfcp-2.8-u2.cnf

p cnf **11 483 525 (variables)** **32 697 150 (clauses)**

1 -3 0

2 -3 0

-1 -2 3 0

x3 = (x1 ∧ x2)

.

.

.

-11482897 -11483041 -11483523 0

11482897 11483041 -11483523 0

11482897 -11483041 11483523 0

-11482897 11483041 11483523 0

(x4 <=> x5 <=> x6)

-11483518 11483524 0

-11483519 11483524 0

-11483520 11483524 0

-11483521 11483524 0

-11483522 11483524 0

-11483523 11483524 0

x7 = (x8 v … v x13)

11483518 11483519 11483520 11483521 11483522 11483523 -11483524 0

Solved in less than 1 minute !!

Modern SAT solvers: four basic break

1. Heavy tailed phenomena [Gomes et al. 97]

   → **Restarts**

1. Resolution based conflict analysis [Marques Silva et al. 96]

   → **Learning**

1. Activity-based heuristics [Brisoux & Sais 99, Moskewicz et al. 01]

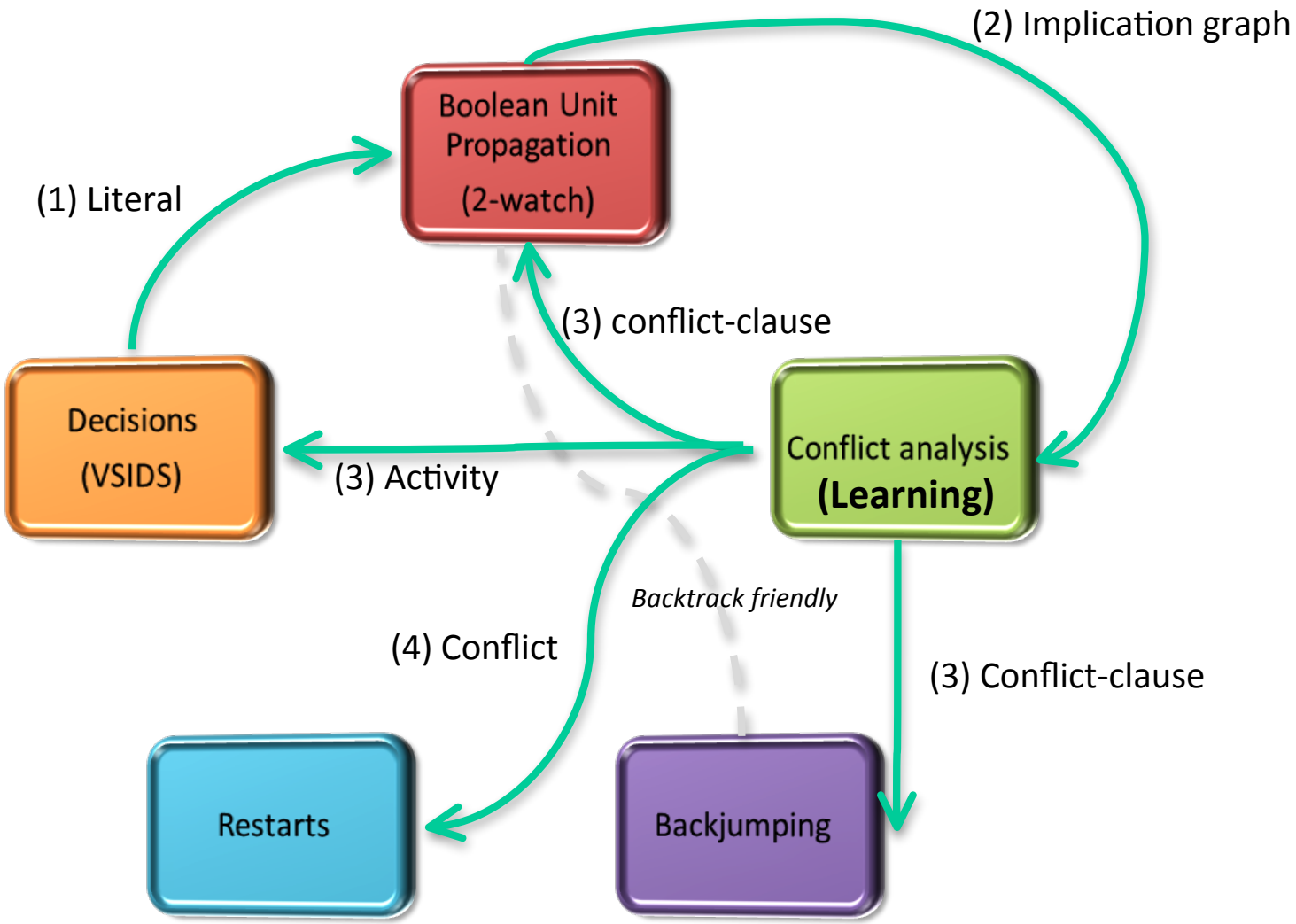
   → **VSIDS**

1. Efficient Boolean Constraint propagation (BCP) [Zhang et al. 97, Moskewicz et al. 01]

   → **Watched literals**

– Four components proposed in four years

Modern SAT solvers: architecture



Outline

- Propositional Satisfiability Problem (SAT)
- Modern SAT solvers
- Parallel SAT
  - Divide and conquer based parallel solvers
  - **Portfolio based Parallel solvers [Hamadi et al. JSAT'09]**
- **Control-based clause sharing [Hamadi et al. IJCAI'09]**
- Conclusion/future work.

Why should we move to Parallel SAT?

- Sequential solvers looks difficult to improve (minor improvements, no orders of magnitude).

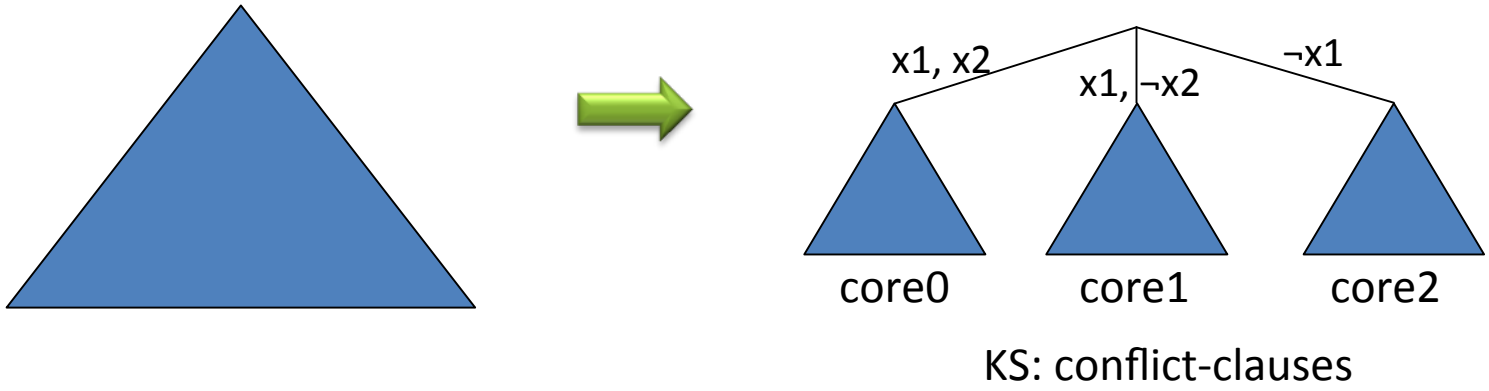
CPU time						
	SAT+UNSAT		SAT		UNSAT	
Rank	Solver	#	Solver	#	Solver	#
Ref	precosat	206	precosat	96	precosat	110
1	glucose	215	contrasat	99	glueminisat	126
2	glueminisat	211	cirminisat	99	glucose	121
3	lingeling	208	mphasesat64	99	qutersat	120

SAT 2011 Competition: application category

- SAT is applied to larger and more ambitious problems which cannot be solved in reasonable time.
  - ~90% of the industrial problems solved in less than 15min.
  - ~10% of the industrial problems solved in 1h.
  - **~30% of problems are open !**

Divide-and-conquer

- Principle: allocate independent sub-trees to different resources.
  - Use a set of constraints (a.k.a. guiding paths) to split the original search space.



KS: conflict-clauses

- Problem: load imbalance

Divide-and-conquer

	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, restricted clause sharing
ManySAT [Hamadi et al. 2008]	Minisat	multicore	Restricted clause sharing

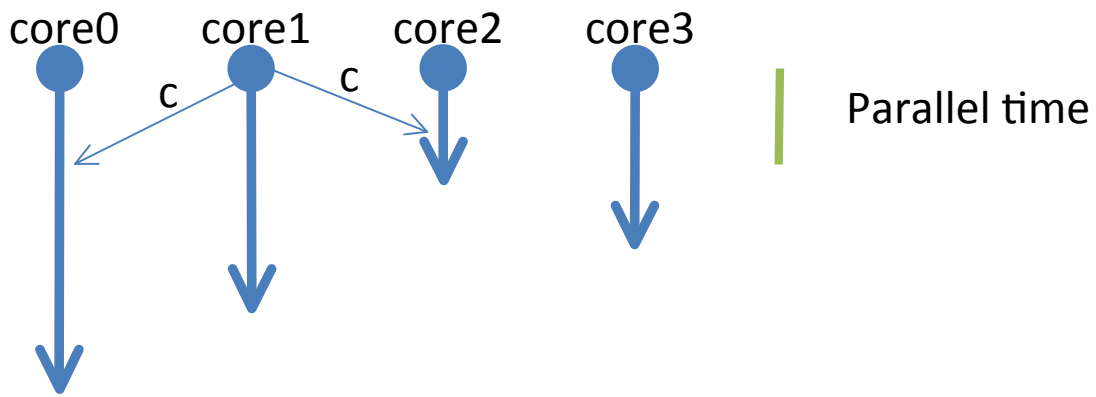
¬Divide-and-conquer: ManySAT

- [Hamadi, Jabbour, Sais 2008]
- Observation:
  - Modern SAT solvers became highly stochastic.
  - They are sensible to their parameters, i.e., lack of robustness.

- ManySAT's principle: let several (differentiated) DPLLs compete and cooperate to be the first to solve a given instance.

¬Divide-and-conquer: ManySAT

- Use different but related strategies.
- Cooperation through clause-sharing.
- Performance: better than the best.



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 $y_i > y_{i-1}$ $f(y_{i-1}, y_i) = \frac{\alpha}{y_i} \times \lvert cos(1 + \frac{y_{i-1}}{y_i}) \rvert$ else $f(y_{i-1}, y_i) = \frac{\alpha}{y_i} \times \lvert cos(1 + \frac{y_i}{y_{i-1}}) \rvert$ $\alpha = 1200$	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

Measuring Performance

- **Scalability**: how well a parallel system takes advantage of increased computing resources.

- **Terms**
  - Sequential runtime — Ts
  - Parallel runtime — Tp (with p procs)
  - Parallel overhead — To = pTp –Ts
  - Speedup — S = Ts/Tp
  - Efficiency — E = S/p

- **Typical objective**: divide the sequential runtime by the number of resources, i.e., E close to 1.

ManySAT performance

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

	ManySAT	pMinisat	MiraXT
#problems solved	**90**	85	73
Average speed-up	**6.02**	3.10	1.83
Minimal speed-up	**0.25**	0.34	0.04
Maximal speed-up	**250.17**	26.47	7.56

Outline

- Propositional Satisfiability Problem (SAT)
- Modern SAT solvers
- Parallel SAT
  - Divide and conquer based parallel solvers
  - **Portfolio based Parallel solvers [Hamadi et al. JSAT'09]**
- **Control-based clause sharing [Hamadi et al. IJCAI'09]**
- Conclusion/future work.

Clause sharing

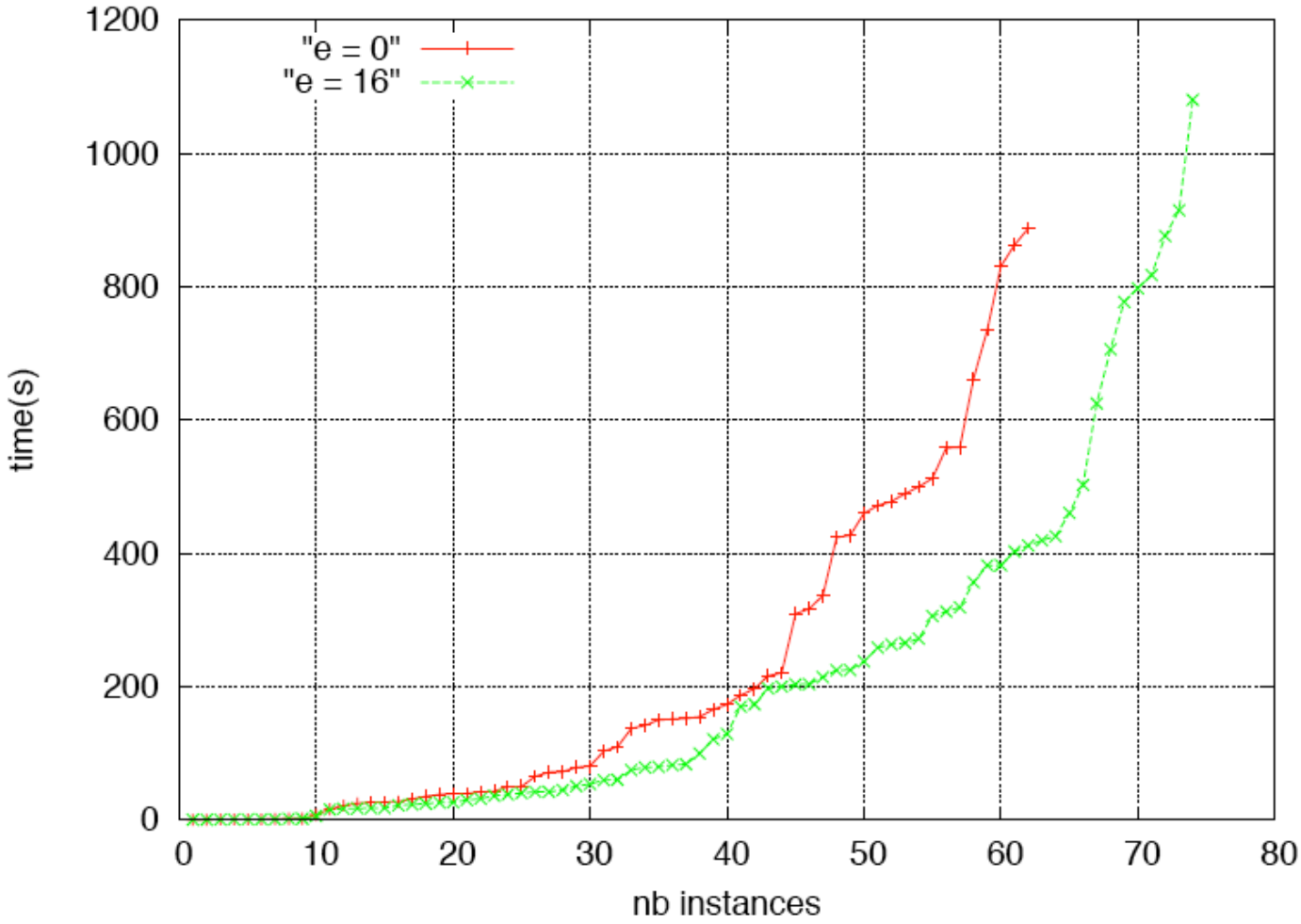
- Particular KS mechanism: each unit exchanges its conflict-clauses.
- Benefit: better pruning.
- Problem: exponential number of clauses.
- Solution: export up to some fixed size limit.

  -> reduce the overhead

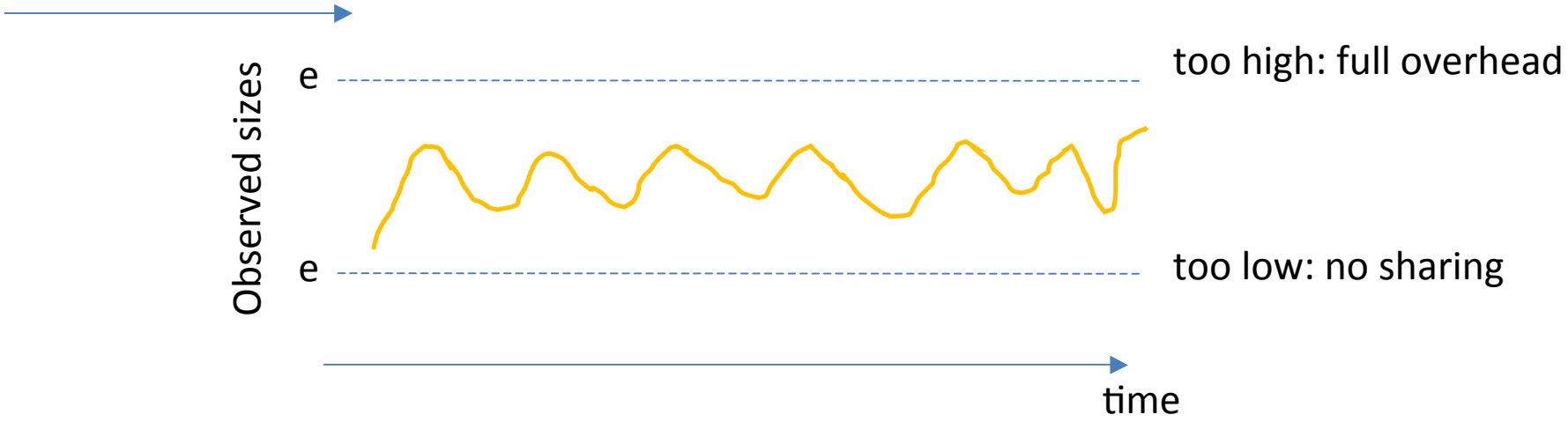
  -> less is more

  remember: $|c|=k$: removes $2^{(n-k)}$ tuples.

The benefit of clause sharing (on ManySAT)

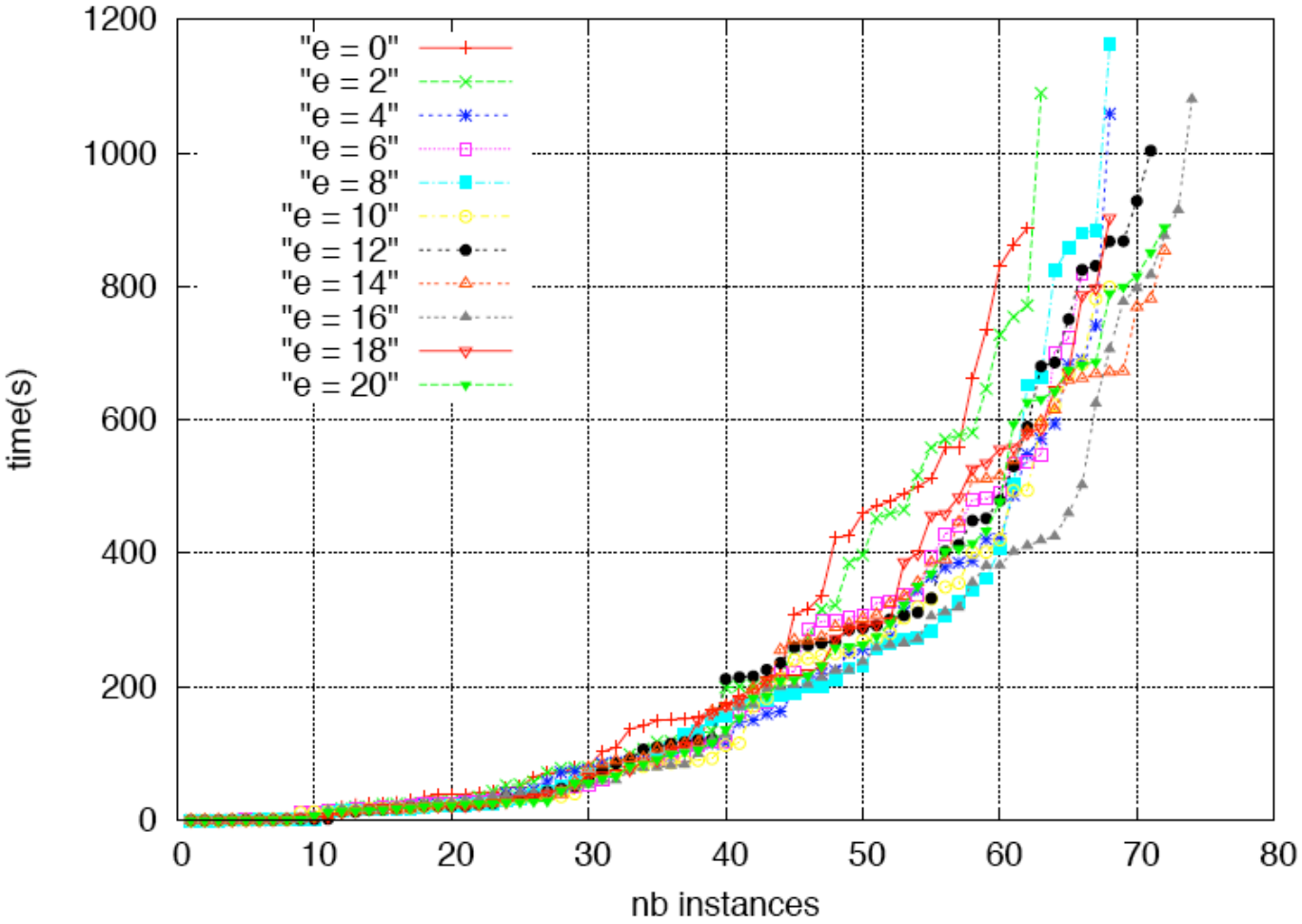


Problems with clause sharing (1)



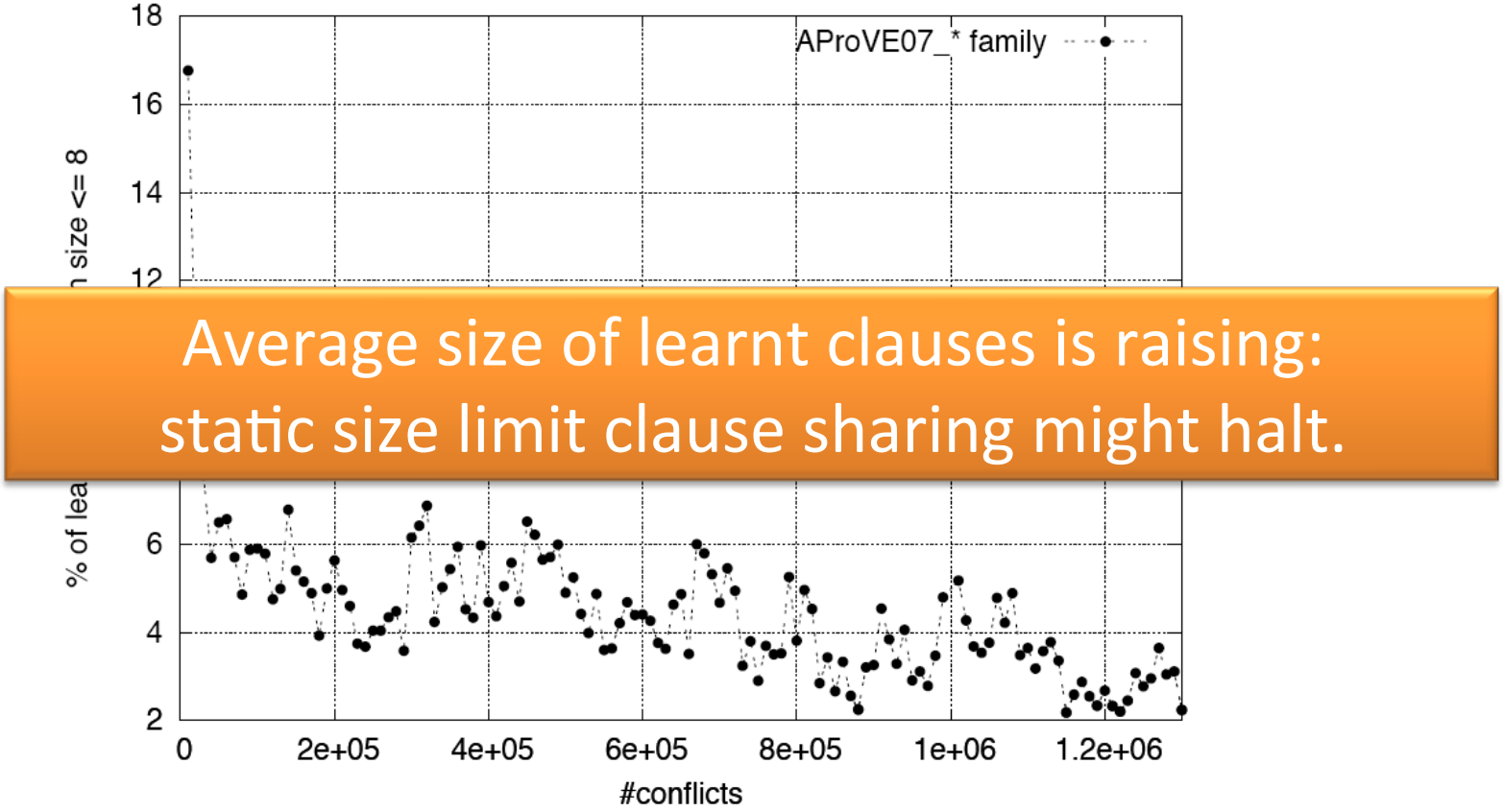
- Offline tuning of the static size limit.

The benefit of clause sharing (on ManySAT)



Problems with clause sharing (2)

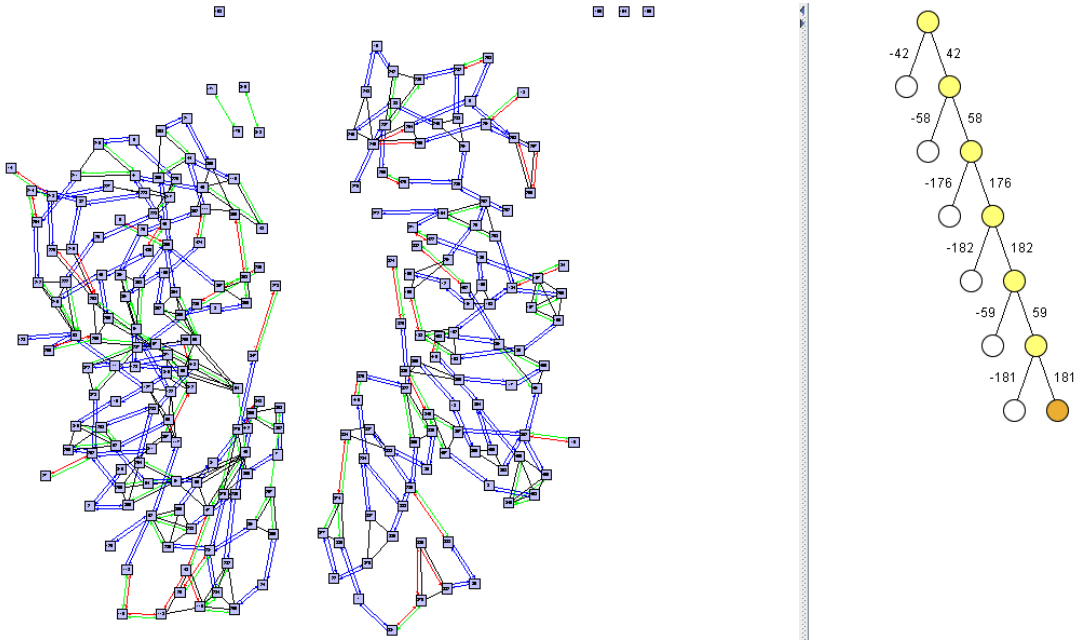
- Simple experiment with Minisat 2.0 (sequential)



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

Problems with clause sharing (3)

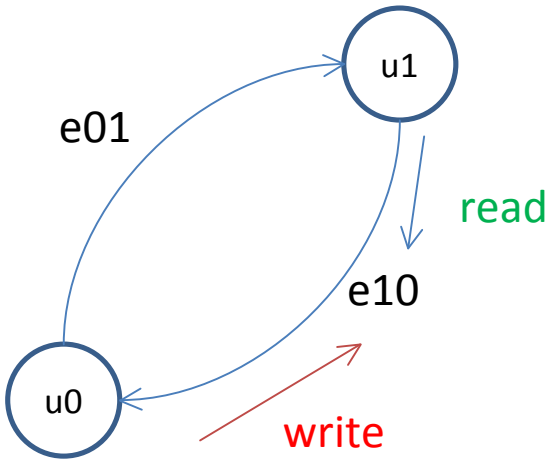
- Exchange between unrelated search efforts:



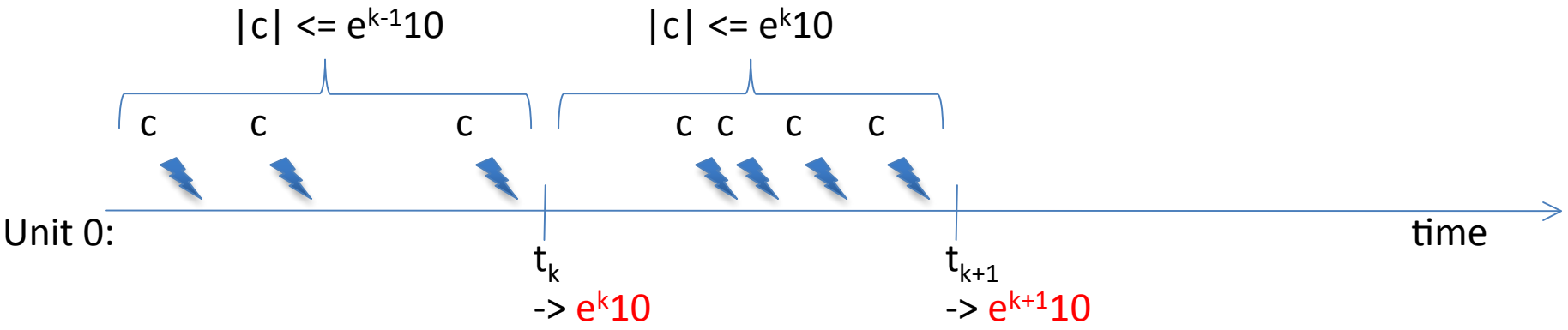
Our Solution

- Pairwise size limits to control clause sharing.
- Start with high limits to enforce sharing.
- Each unit performs periodic revisions of incoming pairwise limits.
  1. Maintain throughput *T (solves (1) and (2)).*
  2. Maintain throughput *T* of a given quality >= Q (solves (3)).

Our Solution



Rem: no shared memory lock.



Maintain a throughput T

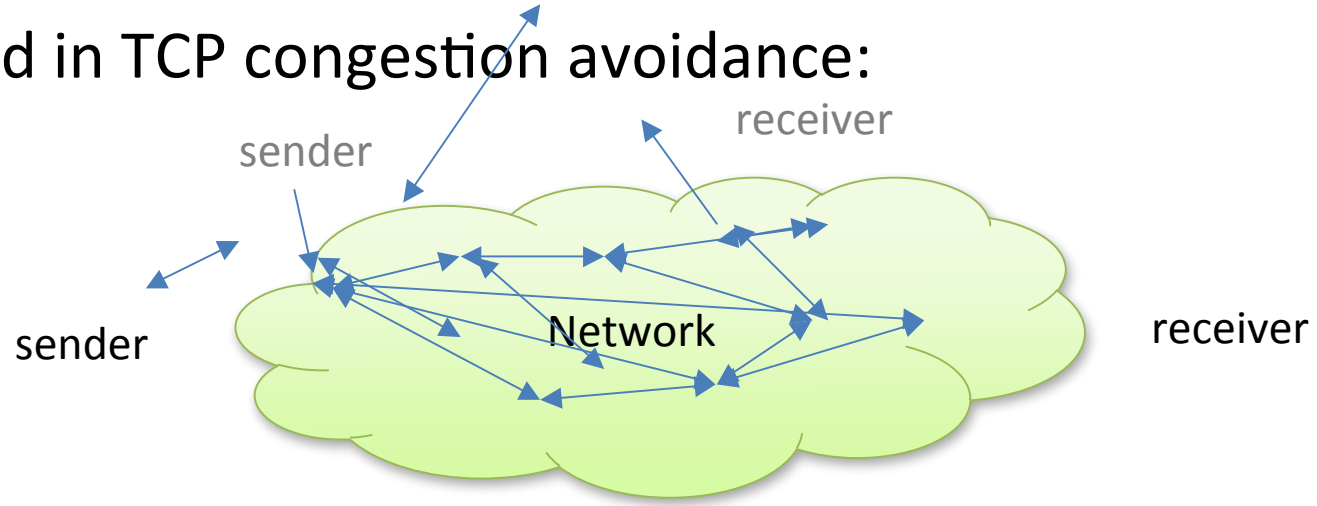
- T is a number of foreign clauses received in each time interval.

- Unit i, at step tk:
  - If Rk < T, uniform increase of $e^k ji$ limits.
  - If Rk > T, uniform decrease of $e^k ji$ limits.

- How do we update the limits?

Additive Increase Multiplicative Decrease (AIMD)

- Used in TCP congestion avoidance:



- Guess the available bandwidth, i.e., find the communication rate w:
  - 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.

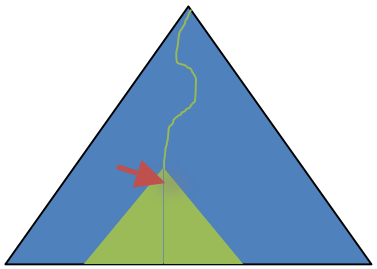
Additive Increase Multiplicative Decrease (AIMD)

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

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

Maintain a throughput T of quality >= Q

- VSIDS heuristic: variables with the best ranking are related to the current part of the search space.
- Def.
  - Maximum VSIDS activity: $\mathcal{A}_i^{max}$
  - Set of active literals of a foreign clause c:



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

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

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

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

Maintain a throughput T of quality >= Q

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

- Increase/Decrease:
  - Favor units which give good quality clauses.

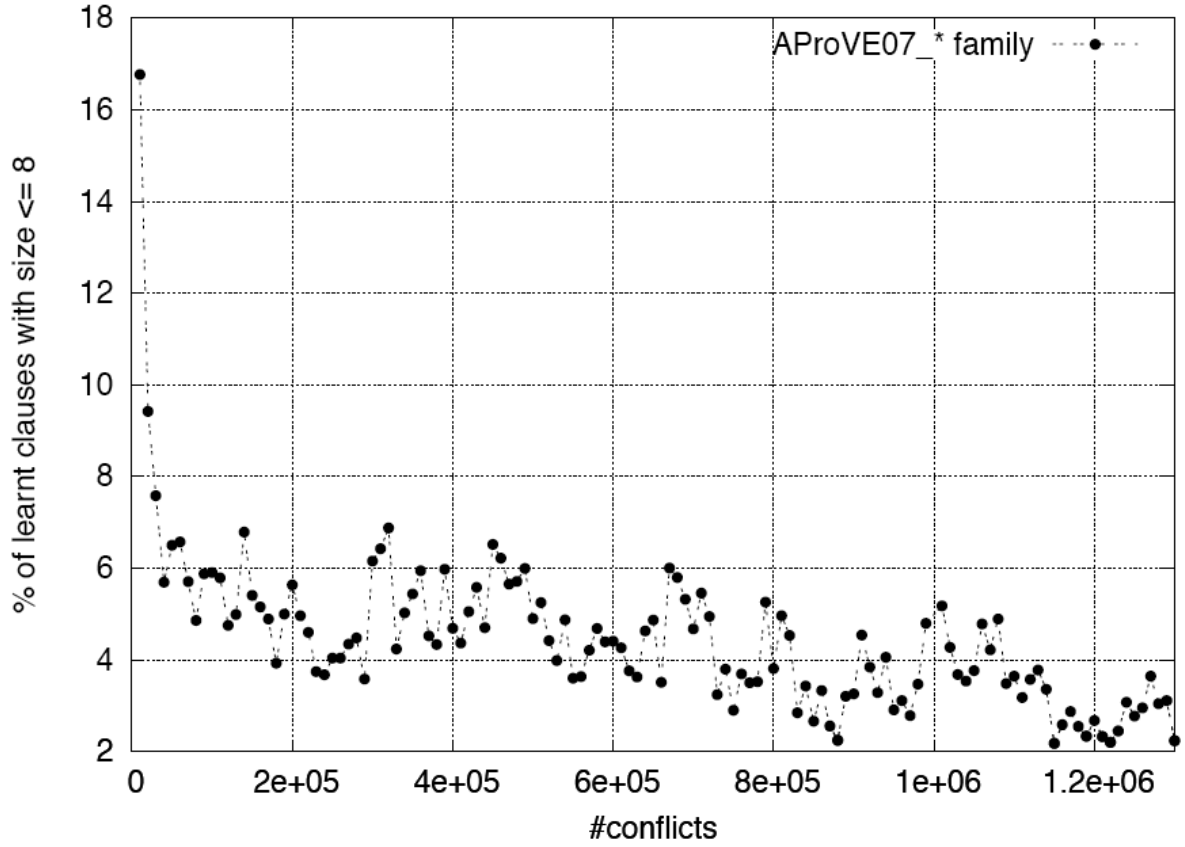
Parameters

- 4 cores
- α = 10000 conflicts (tk+1 = tk+α)
- T= α/2
- Q=|c|/3
- a=0.125
- b=8
- Offline tuning for α.

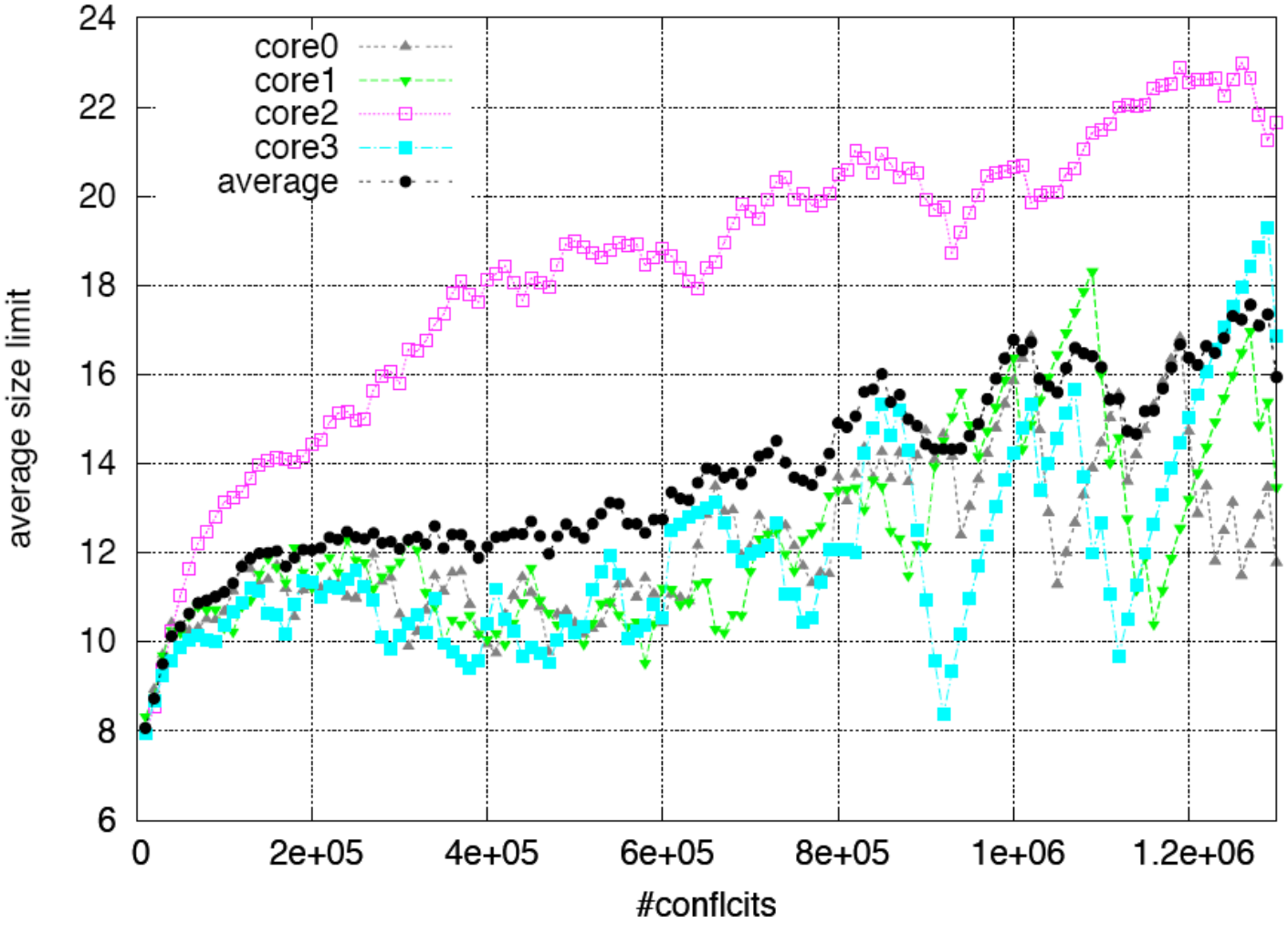
$$aimdTQ(R_i^k)\{$$
$$\forall j|0 \leq j < n, j \neq i$$
$$e_{j\rightarrow i}^{k+1} = \begin{cases} e_{j\rightarrow i}^k + (\frac{Q_{j\rightarrow i}^k}{100}) \times \frac{b}{e_{j\rightarrow i}^k}, if(|R_i^k| < T) \\ e_{j\rightarrow i}^k - (1 - \frac{Q_{j\rightarrow i}^k}{100}) \times a \times e_{j\rightarrow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

Problems with clause sharing (2)

- Simple experiment with Minisat 2.0 (sequential)



The dynamic of pairwise limits



Performance on Industrial problems

		ManySAT e=8		ManySAT aimdT			ManySAT aimdTQ		
family/instance	#inst	#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

Performance on Industrial problems

		ManySAT e=8		ManySAT aimdT			ManySAT aimdTQ		
family/instance	#inst	#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)**

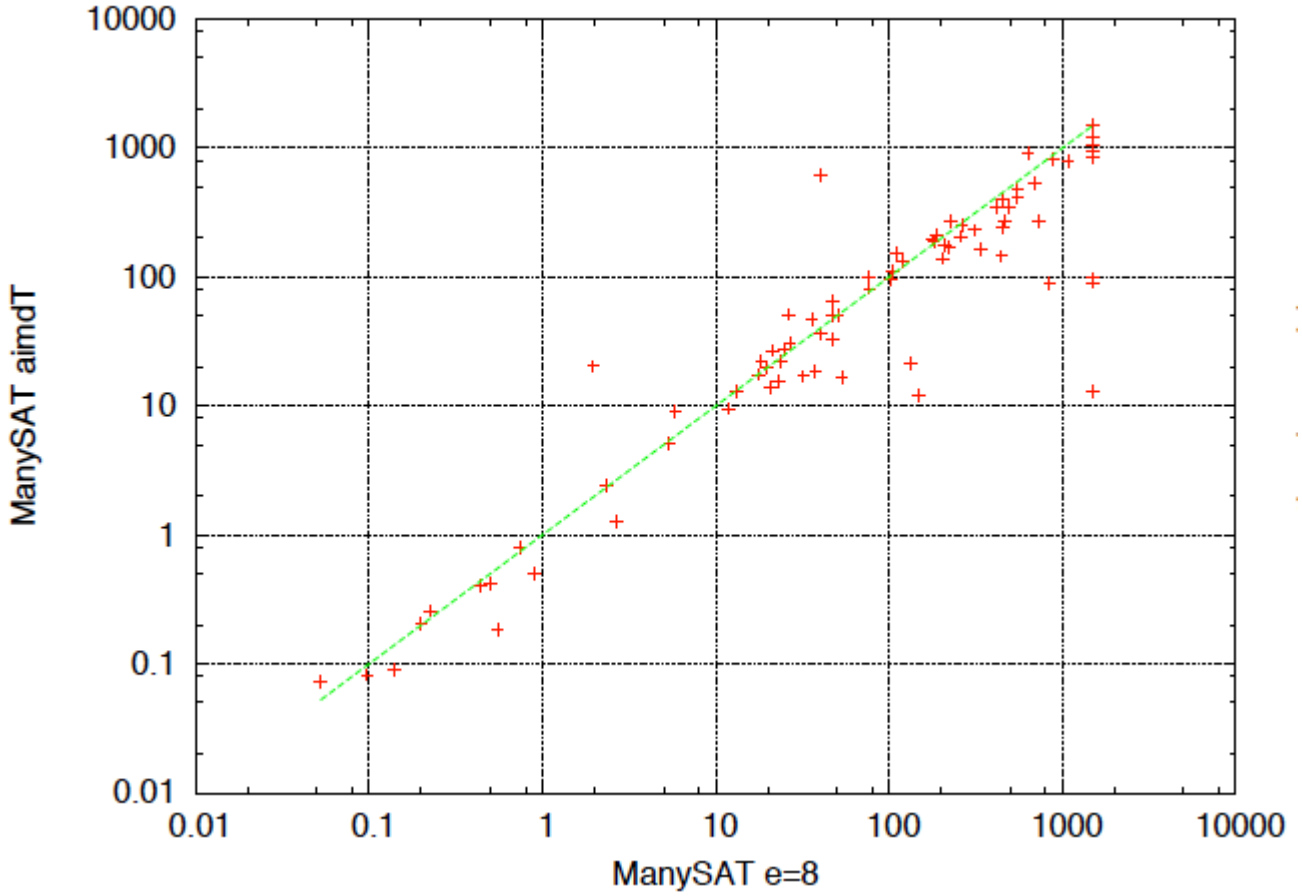
Table 1: SAT-Race 2008, industrial problems

Performance on Industrial problems

		ManySAT e=8		ManySAT aimdT			ManySAT aimdTQ		
family/instance	#inst	#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

Performance on Crafted problems (I)



Conclusion

- SAT allows the efficient encoding/resolution of important problems, e.g., software verification.
- Sequential SAT
  - Heavily improved in the last decade.
  - A few remaining paths to try, don't expect another 3 orders of magnitude gain.
- Parallelism is happening now: multicore PCs

Conclusion

- Knowledge sharing is crucial:
  - Better than the best
  - There are overheads
- Control theory to adapt the rate of p2p collaboration to a given instance.
  - Automatically 'disconnect' unrelated search efforts.
  - Automatically reinforce exchanges between related ones.
  - Effectively remove one parameter:
    - ./manysat.exe –m 4 –e 8
    - ./manysat.exe –m 4

Other contributions

[1] Long Guo, Youssef Hamadi, Said Jabbour, Lakhdar Sais **Diversification and Intensification in Parallel SAT Solving.** International Conference on Principles and Practice of Constraint Programming - CP 2010, LNCS, Springer, pages 252-265, St. Andrews, Scotland, UK, September 6-10 2010.

[2] Youssef Hamadi, Said Jabbour, Cédric Piette, and Lakhdar Sais**,Deterministic Parallel DPLL: System Description.** Int. Journal on Satisfiability, Boolean Modeling and Computation (JSAT), 2011.