

From Parallel SAT to Distributed SAT

Youssef Hamadi,

Microsoft Research, Cambridge, United Kingdom

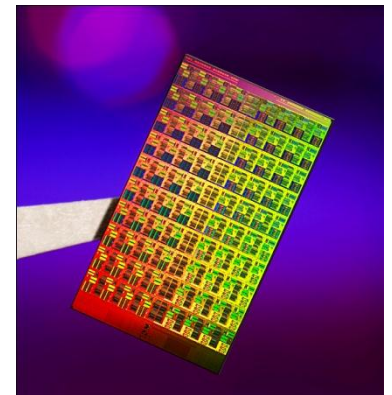
Ecole Polytechnique, Palaiseau, France

Outline

- Motivation, definitions
- Parallel tree-based search
- Control-based clause sharing
- Diversification and Intensification
- Conclusion, perspectives

Motivation: technological

- **Thermal wall**: increases in processor clock frequency are slowing and in many cases frequencies are being decreased to reduce power consumption.
- **Moore's law**: the number of transistors that can be inexpensively placed on an integrated circuit is increasing exponentially, doubling approximately every two years.
- **Scalability** through more computing units.



Motivation: algorithmic

- State of the art **sequential** algorithm looks difficult to improve (no orders of magnitude improvements).
- SAT is applied to larger and **more ambitious problems** which cannot be solved in reasonable time.

SAT Competition 2009: ~30% of the industrial instances were not solved in nearly 3h.

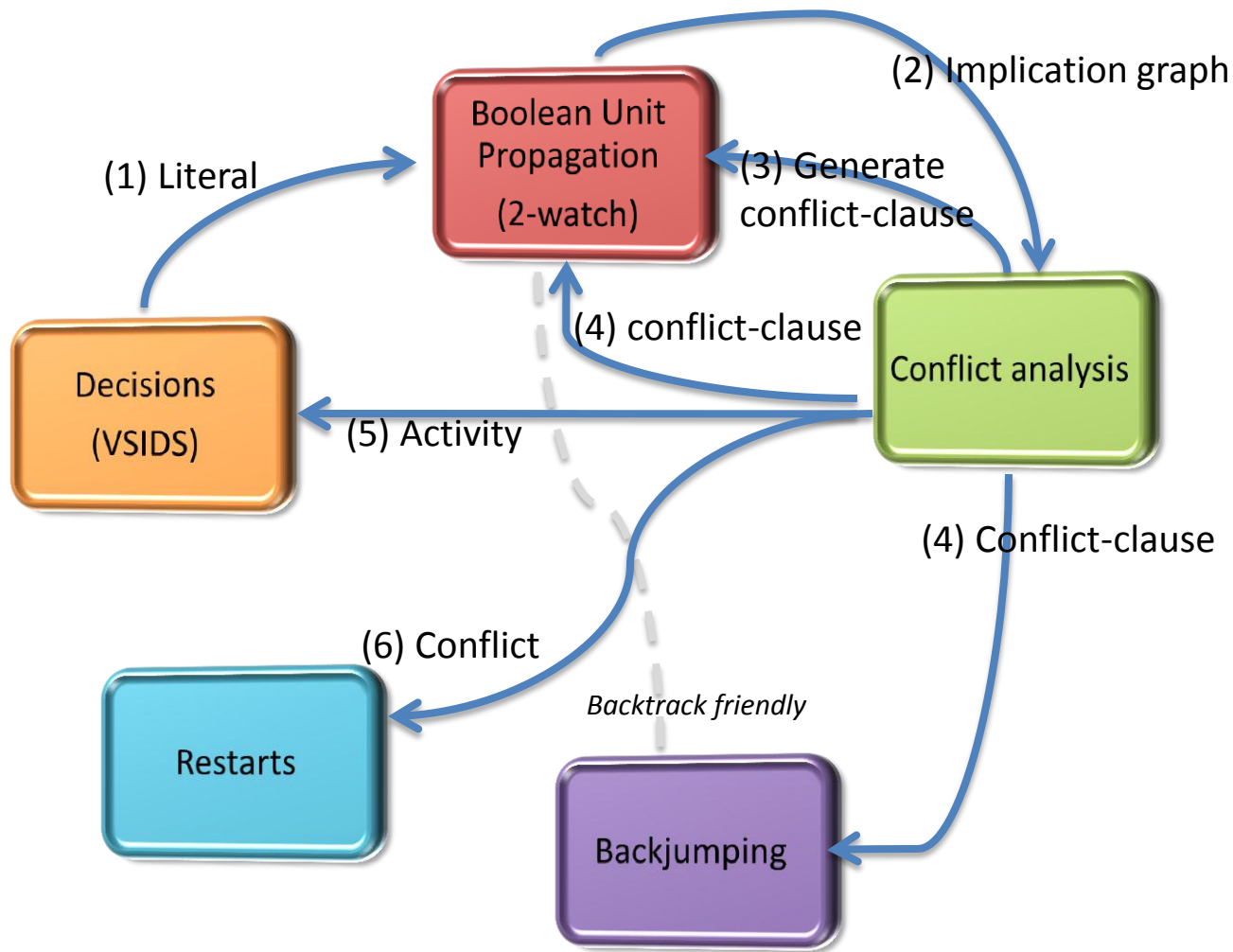
Definitions

- **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)
 - Parallel overhead $T_o = pT_p - T_s$
 - Speedup $S = T_s/T_p$
 - Efficiency $E = S/p$
 - Typical objective: divide the sequential runtime by the number of resources, i.e., E close to 1.

Definitions

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

Modern SAT Solver

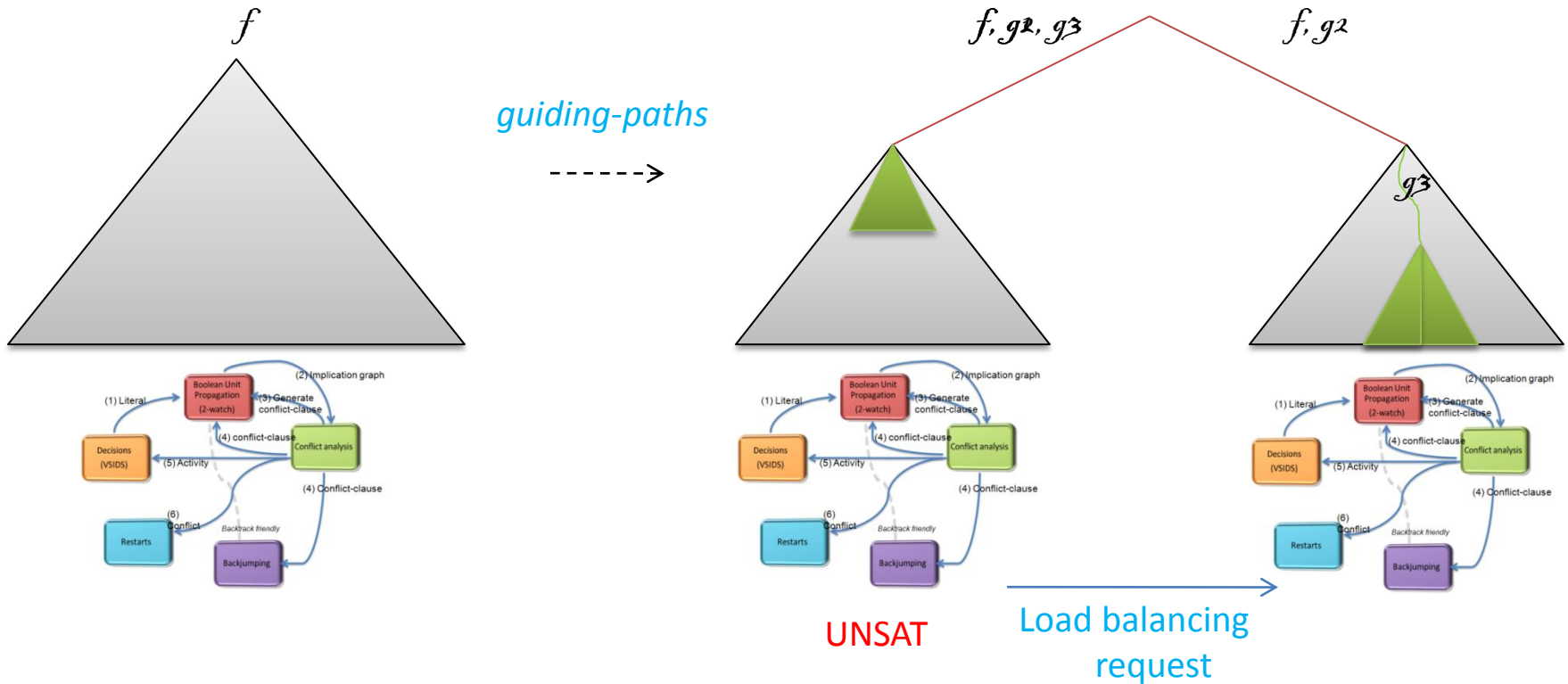


PARALLEL TREE-BASED SEARCH

Divide and conquer

Principles:

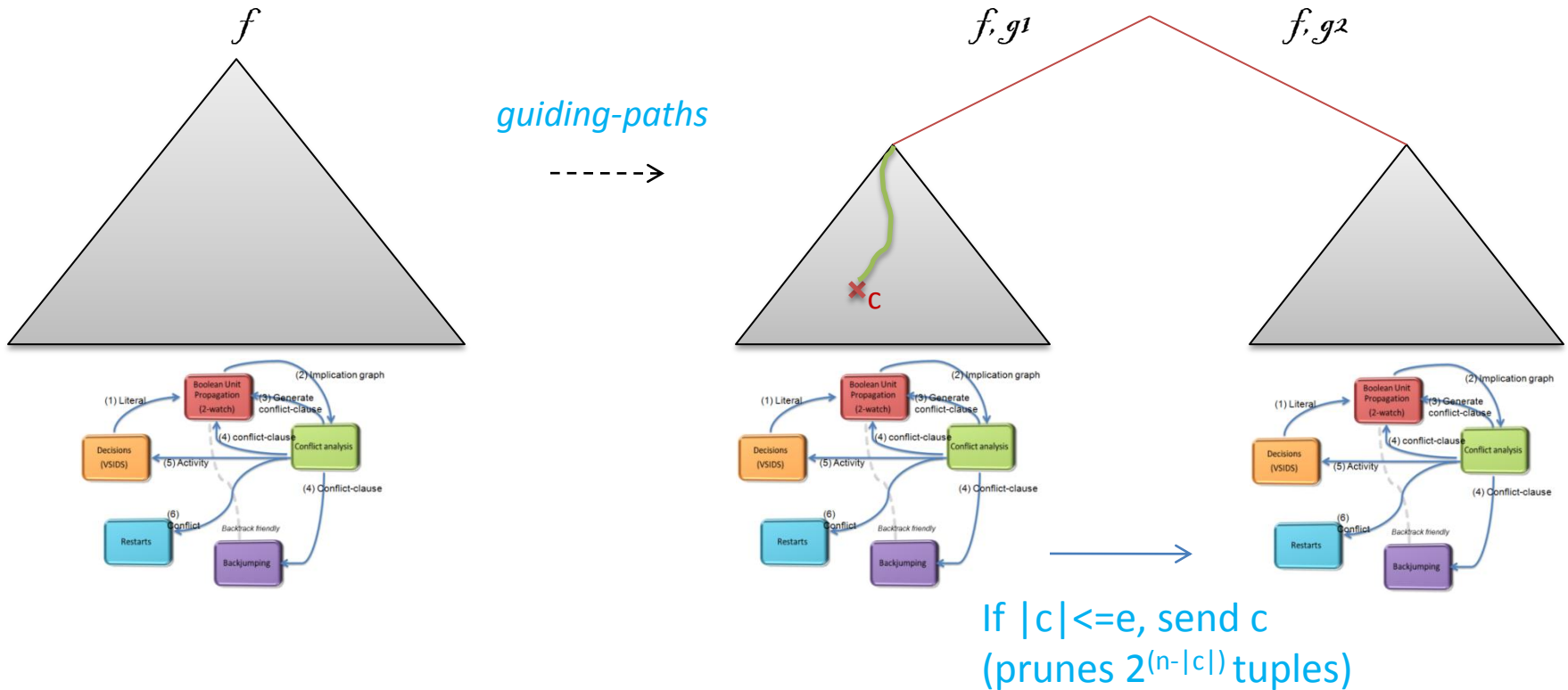
1. Allocate independent subspaces to different resources, organize load-balancing.



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-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>  
  }  
}
```

end, SAT: shared memory variables.

Clause sharing

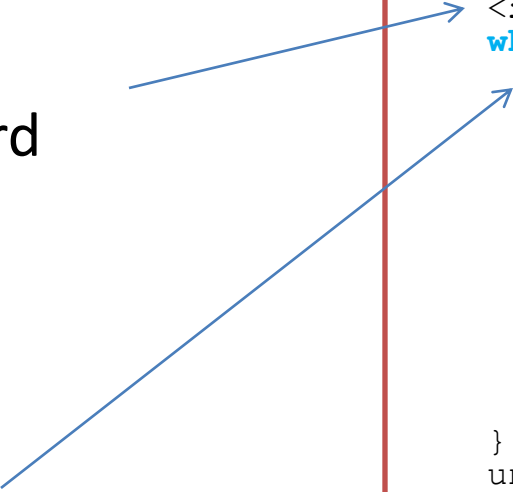
Integration of shared clauses:

1. Top level

Straight forward
e.g., units

2. On the fly

4 cases



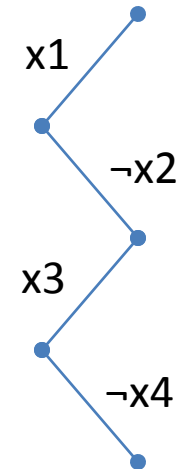
```
SlaveDPLL() {  
  1: get and enforce guiding-path;  
  limit = c;  
  while (!end) {  
    <import foreign-clauses>;  
    while (#conflicts < limit && !end) {  
      <import foreign-clauses>;  
      lit = decide();  
      if (!lit)  
        end = true;  
      SAT = true;  
      if (!BUP(lit))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

Clause sharing

Integration of shared clauses: on the fly

- False

- > Conflict analysis

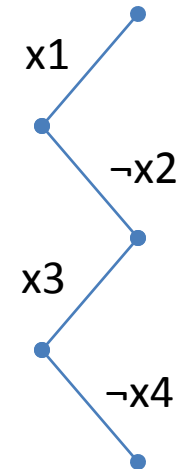


$(\neg x_1 \vee x_2)$

Clause sharing

Integration of shared clauses: on the fly

- False
 - > Conflict analysis
- Unit
 - > BUP

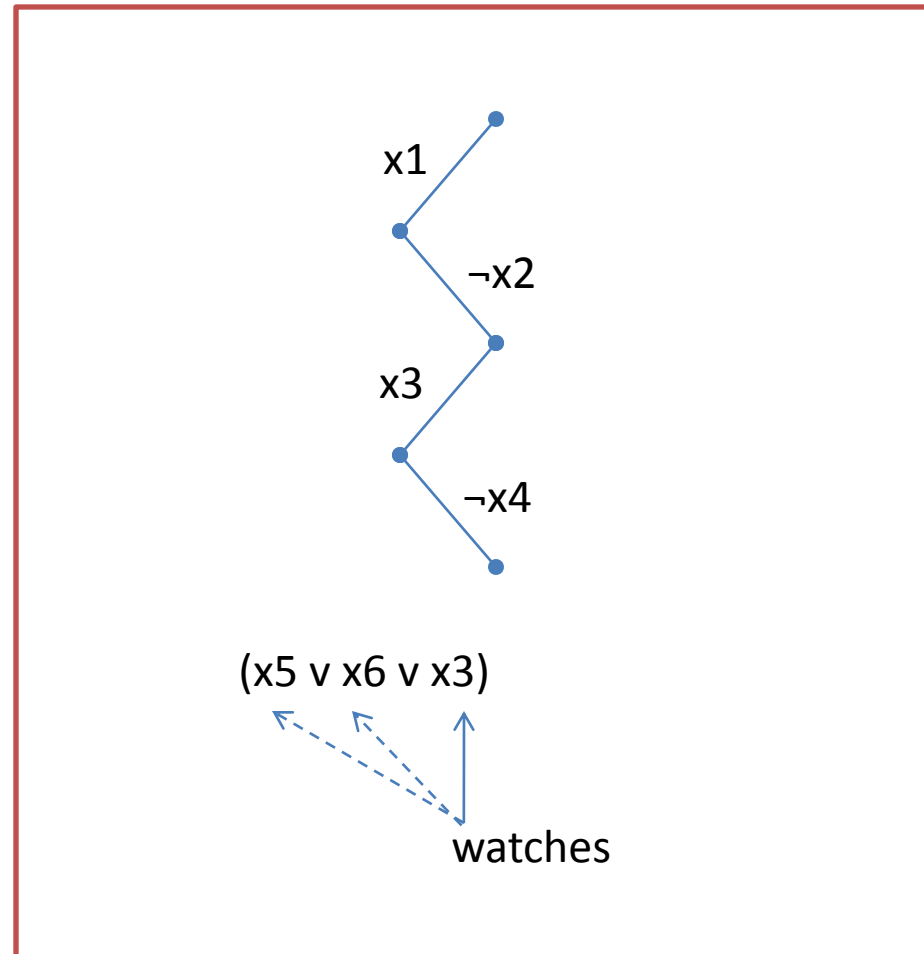


$(\neg x1 \vee x5)$

Clause sharing

Integration of shared clauses: on the fly

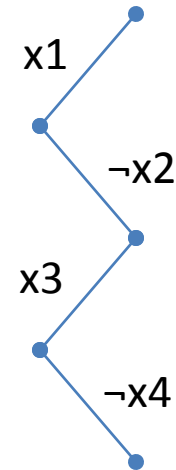
- False
 - > Conflict analysis
- Unit
 - > BUP
- Satisfied
 - > Watch the last satisfied



Clause sharing

Integration of shared clauses: on the fly

- False
 - > Conflict analysis
- Unit
 - > BUP
- Satisfied
 - > Watch the last satisfied
- Otherwise
 - Watch any pair of literals



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, restricted clause sharing

Portfolio approach

- Principle: 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
- ManySAT [Hamadi, Jabbour, Sais 2008]
- Distributed CSP, *M-Framework* [Hamadi, Ringwelski 2005]

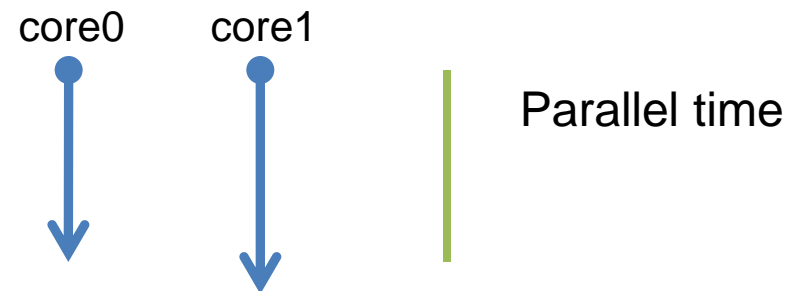
ManySAT: internals

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

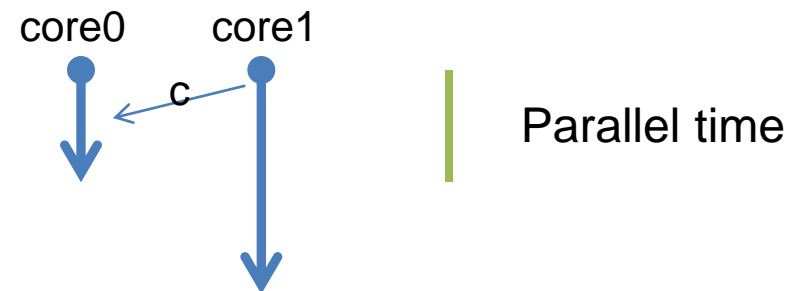
Portfolio approach

- Knowledge sharing: conflict-clause

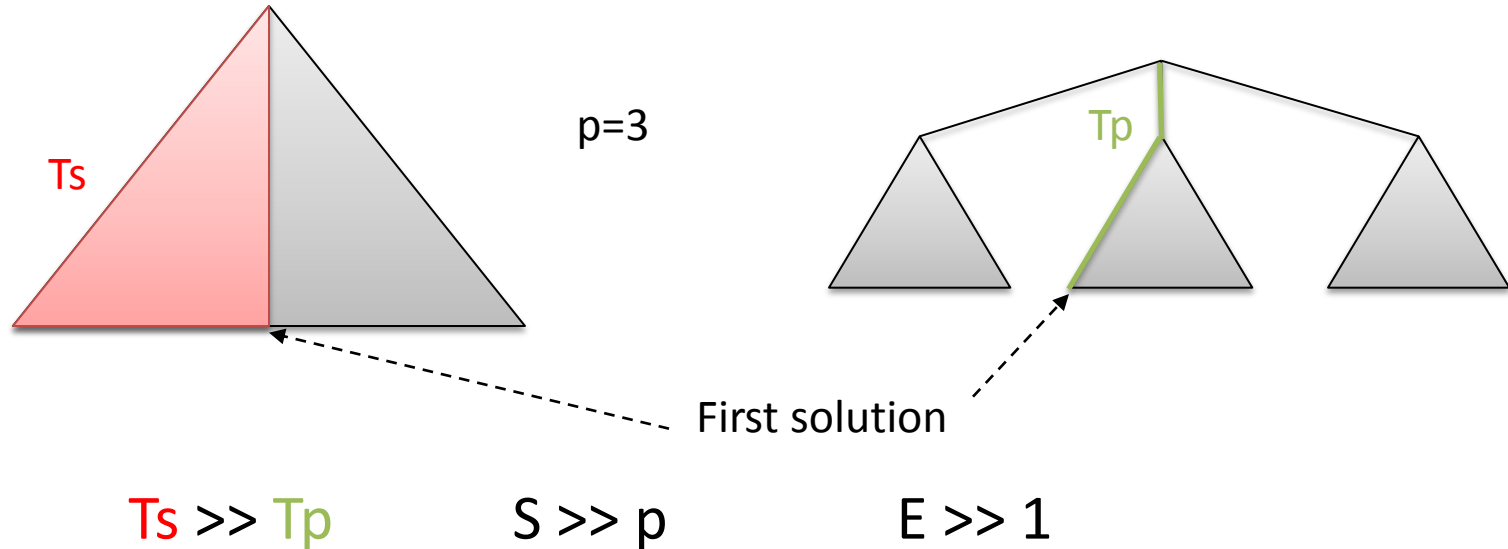
- Without: as good as the best



- 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) [Mesequer 97]

Practical Performance I

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



	ManySAT (MSR-INRIA)	pMinisat (NICTA)	MiraXT (U. Freiburg)
#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
Average efficiency	1.5	0.77	0.45

Practical Performance II



- SAT-Race 2008
 - Non determinism

	ManySAT	pMinisat	MiraXT
Runtime variation	13.7%	14.7%	15.2%
by SAT/UNSAT	22.2%/5.5%	23.1%/5.7%	19.5%/9.7%

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.

Problem 1

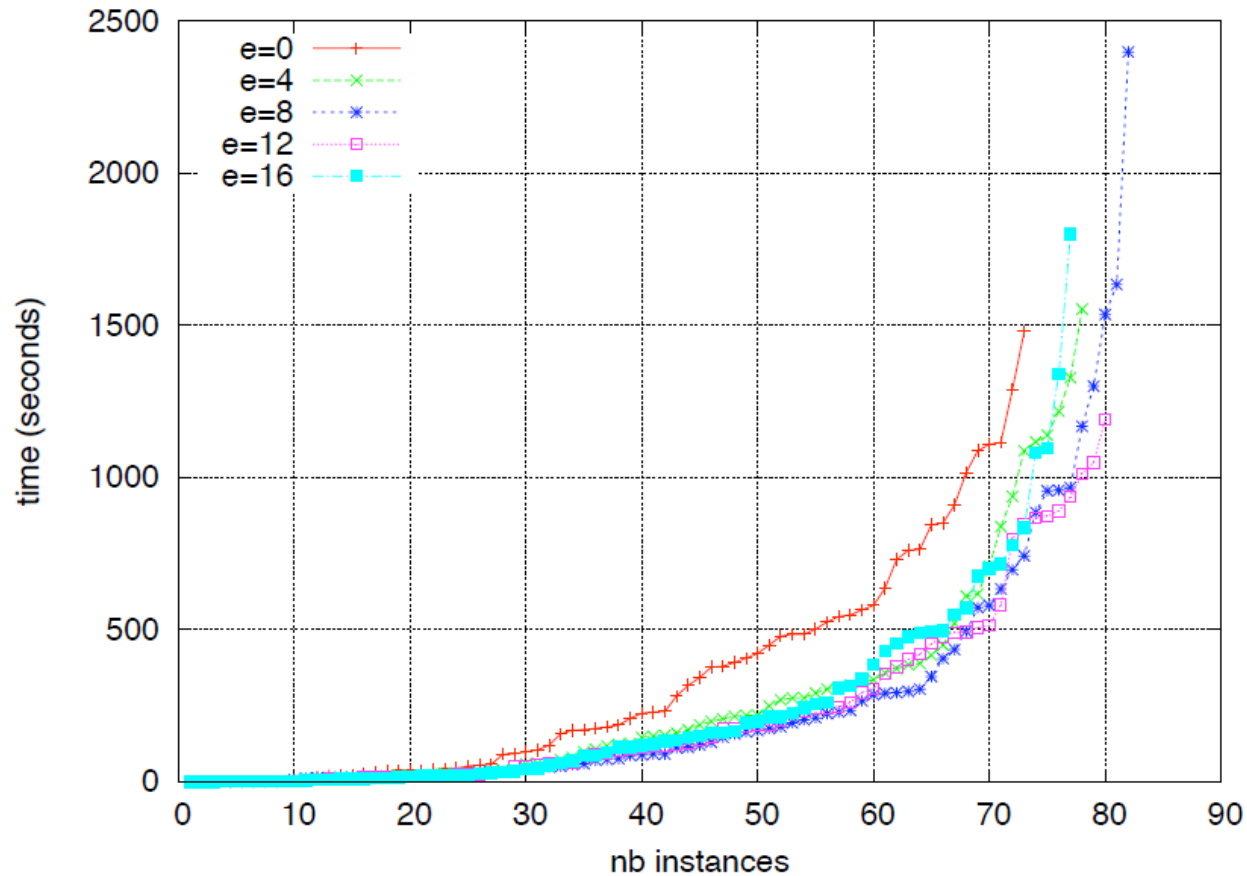
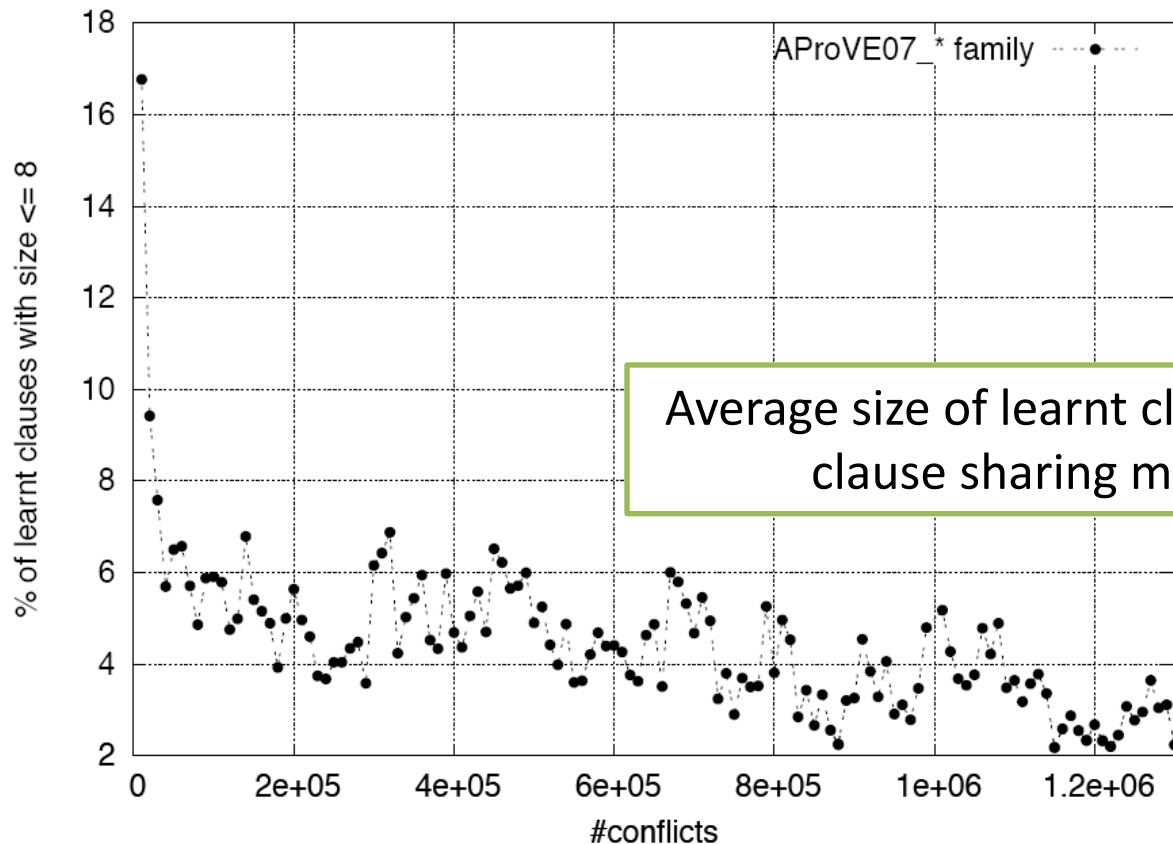


Figure 3. SAT-Race 2008: different limits for clause sharing

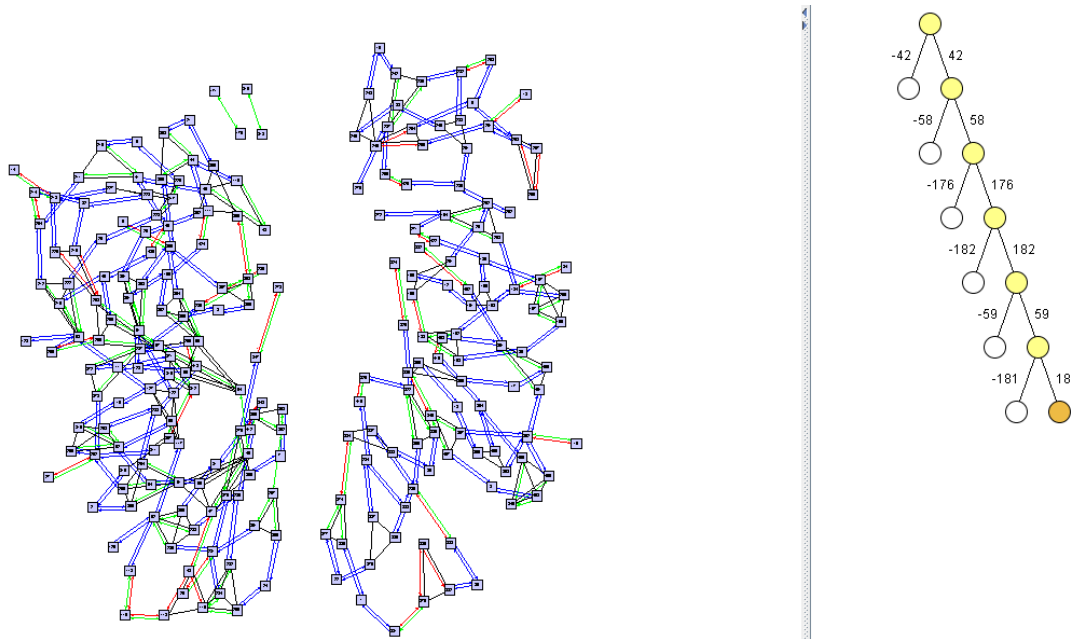
Problem 2

- Simple experiment with Minisat 2.0 (sequential)



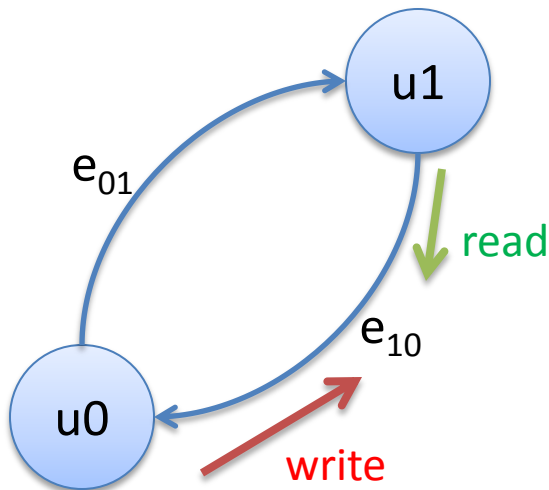
Problem 3

- Exchange between unrelated search efforts:



[DPVis, Sinz 05]

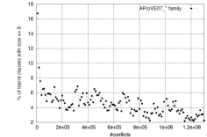
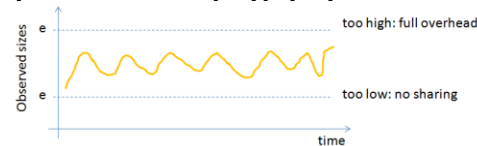
Dynamic limits



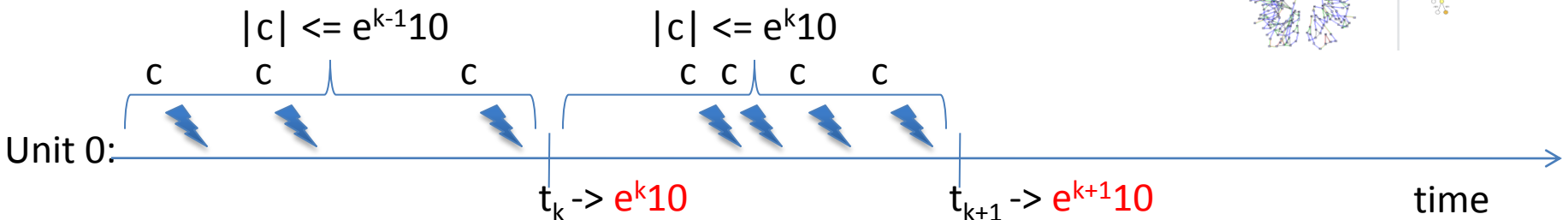
1. Pairwise size limits e_{ij} to control clause sharing from i to j .
2. Each unit performs (lock-free) periodic revisions of incoming limits.

Two objectives:

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



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

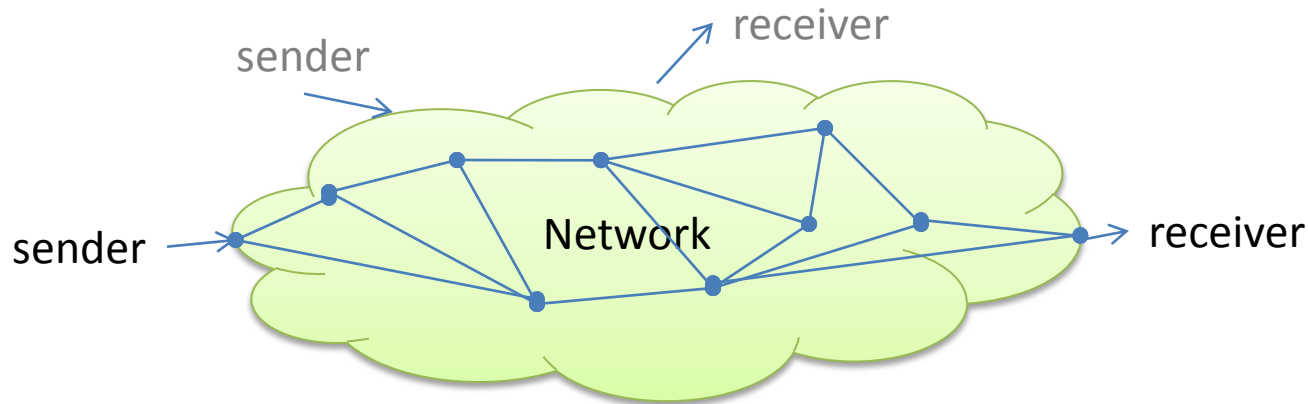


Objective 1: Maintain a throughput T

- 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

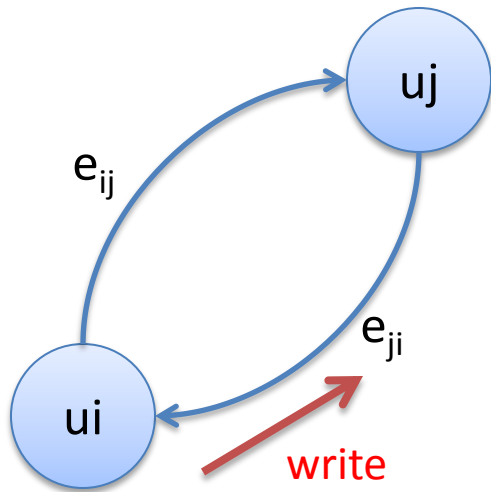
- Problem: guess the available bandwidth, i.e., find the correct communication rate w .



- 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.



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

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

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

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.

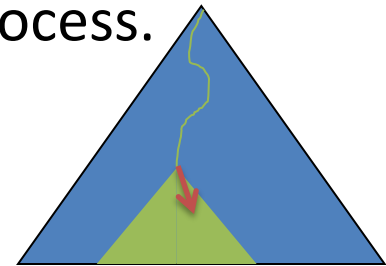
- Maximum VSIDS activity: A_i^{max}
- Set of active literals of a foreign clause c:

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

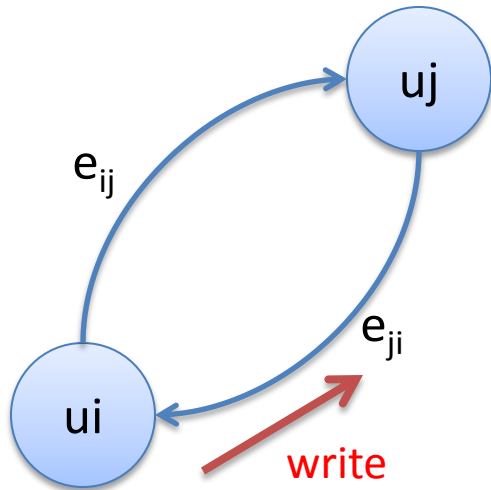
- Set of clauses received from j with at least Q active literals:

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

- 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



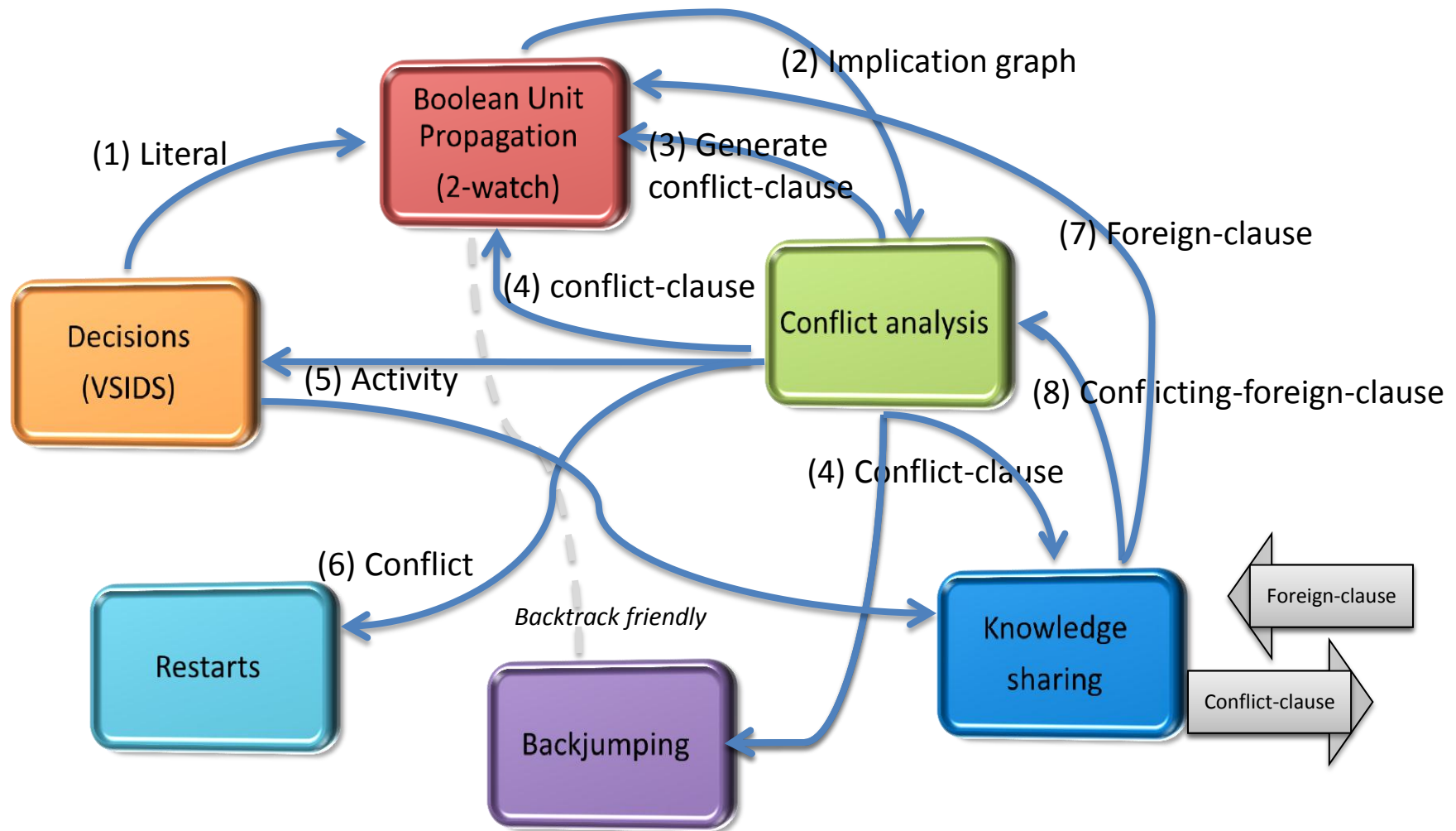
$$aimdTQ(R_i^k) \{$$

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

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

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

Parallel SAT Solving



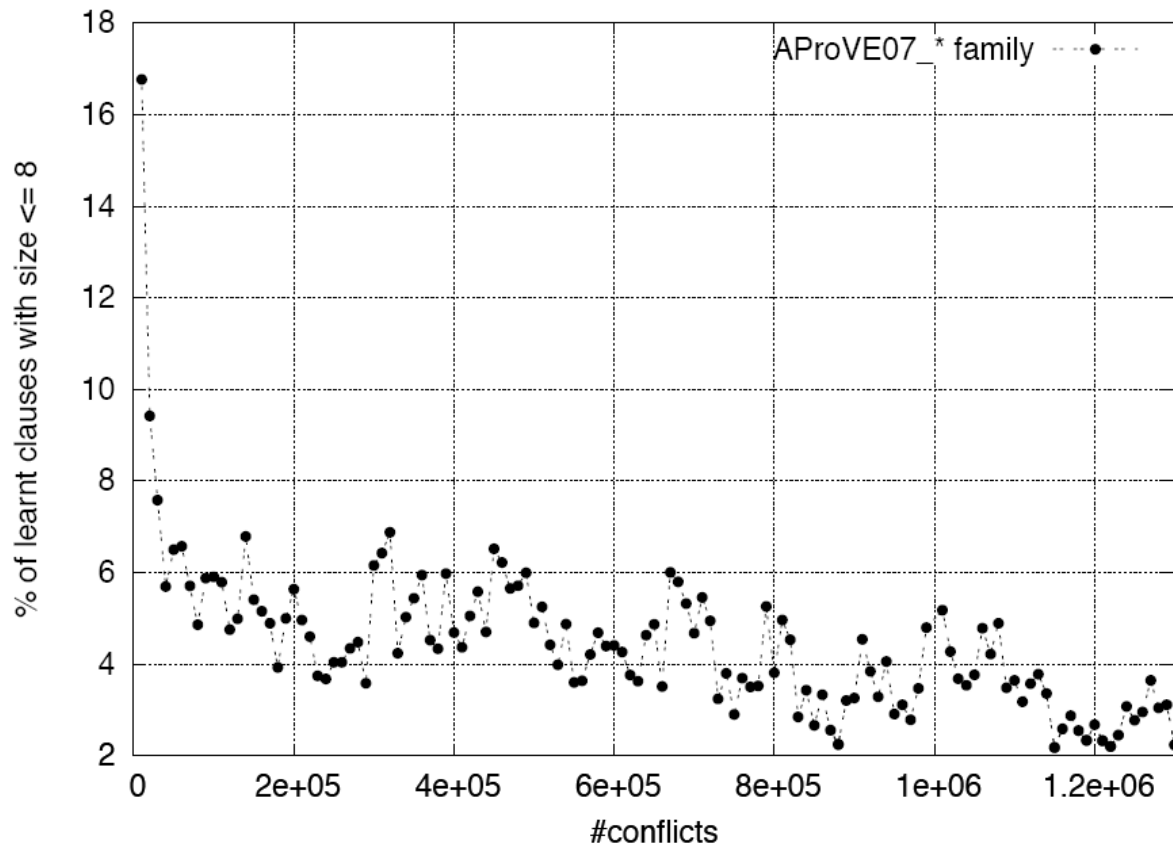
Performance on Industrial problems

		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)

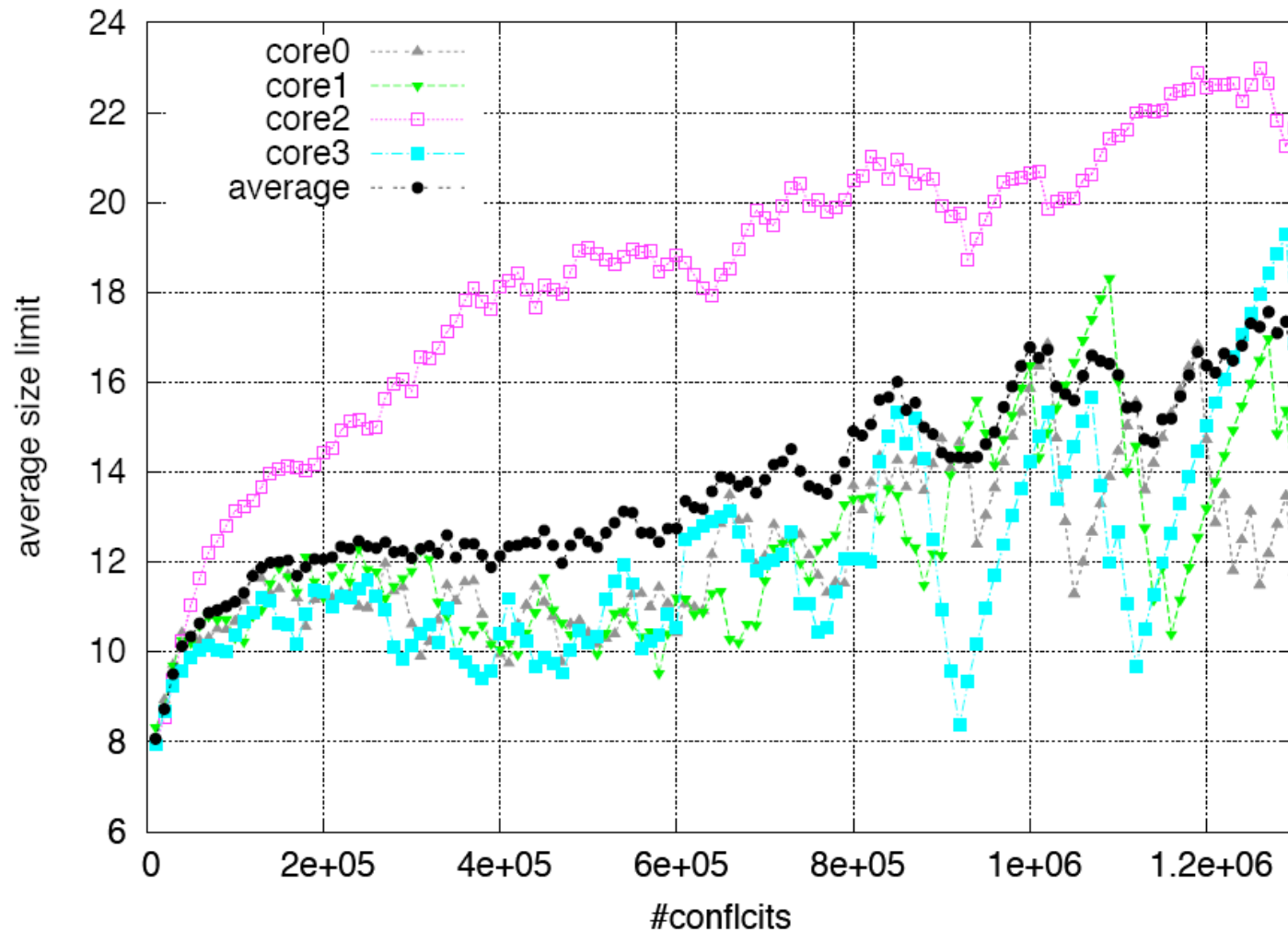
Table 1: SAT-Race 2008, industrial problems

Problems with clause sharing (2)

- Simple experiment with Minisat 2.0 (sequential)



The dynamic of the pairwise limits



DIVERSIFICATION AND INTENSIFICATION

Diversification and Intensification in Parallel SAT Solving, L. Guo, Y. Hamadi, S. Jabbour, and L. Sais, 16th International Conference on Principles and Practice of Constraint Programming (CP'2010) to appear.

Intensification

- Portfolio search = full diversification
- Question: how can we integrate intensification and find the right balance with diversification?
- Proposal:
 - Introduce “roles”
 - Masters, conduct an original search process (diversification)
 - Slaves, intensify their master search process.

Intensification

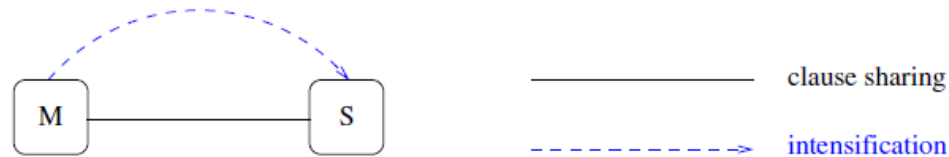
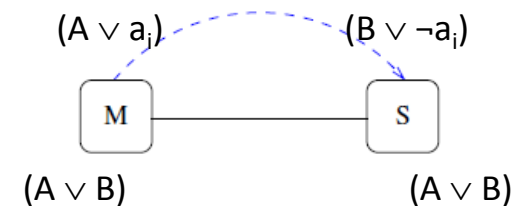


Figure 1: Intensification topology

- Question 1: what **information** should be given to a slave?
- Question 2: **how often** do we have to communicate information?
- Question 3: **tradeoff** Masters/Slaves?

What information should be given to a slave?

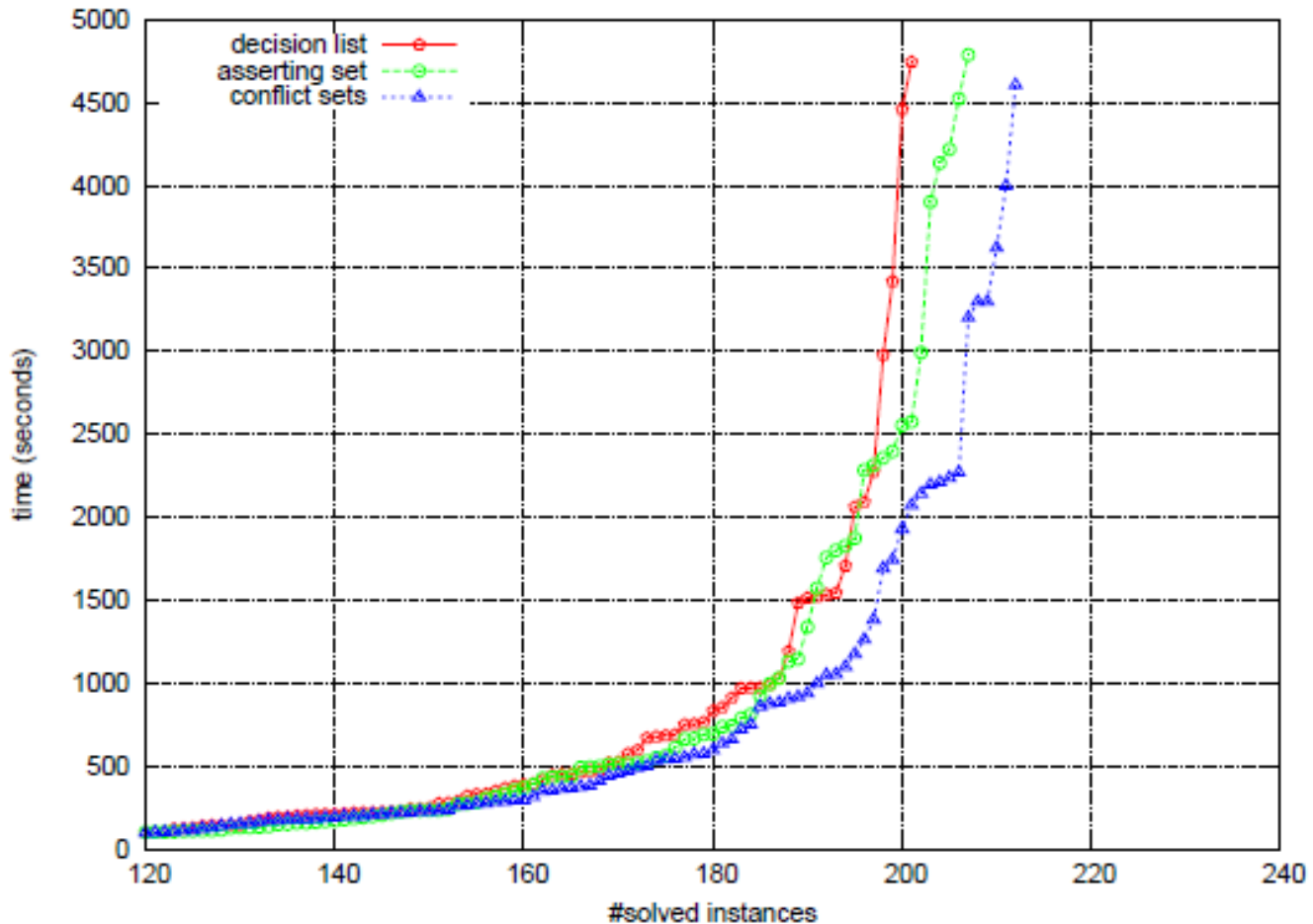
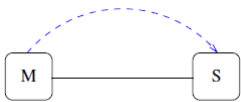
1. Decision list: D_M
 - Activities are not transferred
 - Branching on D_M explores the same area in a different way
2. Asserting set: $A_M = (a_k, a_{k-1}, \dots, a_1)$
 1. Master's learnt: $(a_{nk} \vee a_k), (a_{nk-1} \vee a_{k-1}), \dots, (a_{n1} \vee a_1)$
 2. Branching on a_i can generate learnt $(\dots \vee \neg a_i)$
 3. Connects resolution proofs:
3. Ordered conflict-sets: $C_M = (s_k, s_{k-1}, \dots, s_1)$
 - s_i contains the literals collected during the Master's conflict analysis
 - Directs the slave towards the same conflicts



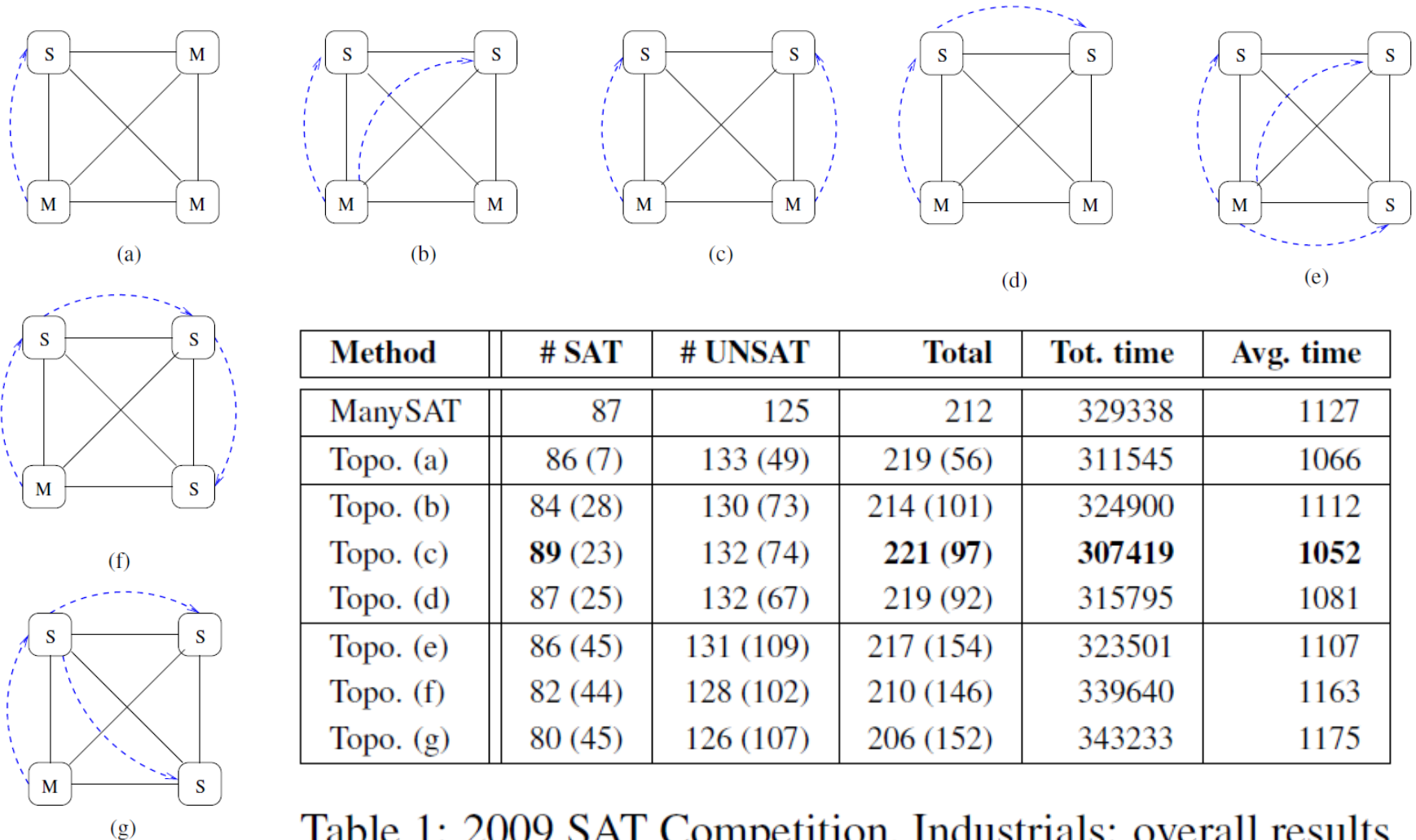
How often do we have to communicate information?

- Objectives
 1. Increase the quality (size) of clauses generated by the slaves
 2. Maintain a tight synchronization of the efforts
- Frequent updates, ~ rapid restarts strategy

Intensification strategies



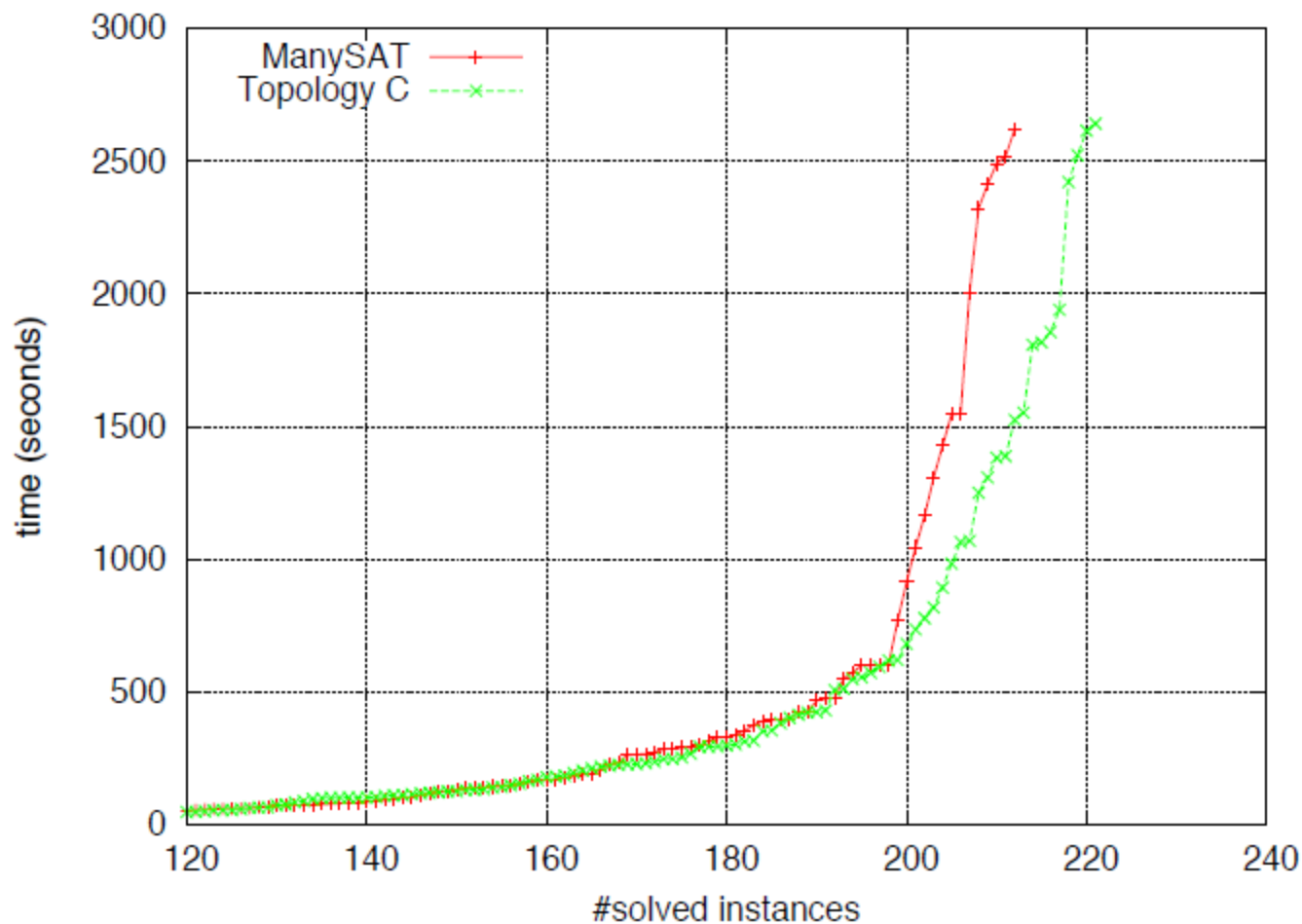
Tradeoff Masters/Slaves



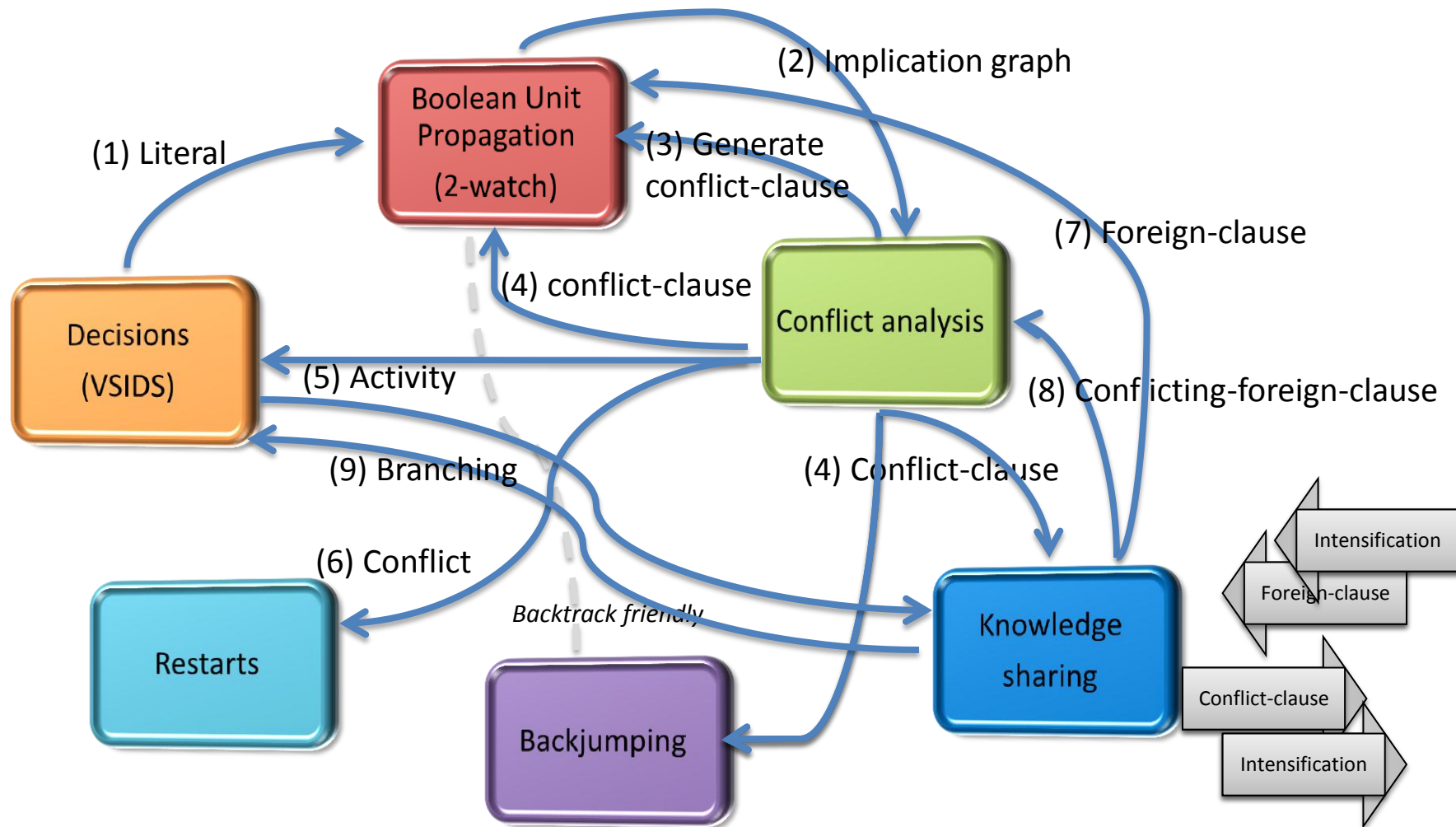
Method	# SAT	# UNSAT	Total	Tot. time	Avg. time
ManySAT	87	125	212	329338	1127
Topo. (a)	86 (7)	133 (49)	219 (56)	311545	1066
Topo. (b)	84 (28)	130 (73)	214 (101)	324900	1112
Topo. (c)	89 (23)	132 (74)	221 (97)	307419	1052
Topo. (d)	87 (25)	132 (67)	219 (92)	315795	1081
Topo. (e)	86 (45)	131 (109)	217 (154)	323501	1107
Topo. (f)	82 (44)	128 (102)	210 (146)	339640	1163
Topo. (g)	80 (45)	126 (107)	206 (152)	343233	1175

Table 1: 2009 SAT Competition, Industrials: overall results

Industrial problems



Parallel SAT Solving



Conclusion

- Parallel SAT effectively extends the modern DPLL architecture
- Performance improvements
- Runtime variability
- Importance of controlled knowledge sharing
- Two techniques
 - Divide and conquer
 - Identification of important variables to split the space
 - Overhead caused by load-balancing
 - Portfolio
 - Scalability (quadratic sharing)

Perspectives

- SMT engines
 - A Concurrent Portfolio Approach to SMT Solving, C. M. Wintersteiger, Y. Hamadi, L. M. de Moura, CAV'09
- Distributed Search
 - Decomposition of the formula
 - Distributed BCP, P-complete ☹
 - Important for very large formulas

Short bibliography

- [1] Y. Hamadi, S. Jabbour, and L. Sais. Control-based clause sharing in parallel sat solving. In Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pages 499–504, 2009.
- [2] Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: a parallel SAT solver. Journal on Satisfiability, Boolean Modeling and Computation, 6:245–262, 2009.
- [3] Luis Gil, Paulo Flores, and Luis Miguel Silveira. PMSat: a parallel version of minisat. Journal on Satisfiability, Boolean Modeling and Computation, 6:71–98, 2008.
- [4] G. Chu and P. J. Stuckey. Pminisat: a parallelization of minisat 2.0. Technical report, Sat-race 2008, solver description, 2008.
- [5] M. Lewis, T. Schubert, and B. Becker. Multithreaded sat solving. In 12th Asia and South Pacific Design Automation Conference, 2007.
- [6] Y. Hamadi, G. Ringwelski. Boosting distributed constraint satisfaction. In Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP 2005), Springer LNCS 3709, 549–562, 2005.
- [7] Bernard Jurkowiak, Chu Min Li, and Gil Utard. A parallelization scheme based on work stealing for a class of sat solvers. Journal of automated Reasoning, 34(1):73–101, 2005.
- [8] Wahid Chrabakh and Rich Wolski. GrADSAT: A parallel sat solver for the grid. Technical report, UCSB Computer Science Technical Report Number 2003-05, 2003.
- [9] W. Blochinger, C. Sinz, and W. Kuchlin. Parallel propositional satisfiability checking with distributed dynamic learning. Parallel Computing, 29(7):969–994, 2003.
- [10] Max Böhm and Ewald Speckenmeyer. A fast parallel sat-solver - efficient workload balancing. Annals of Mathematics and Artificial Intelligence, 17(3-4):381–400, 1996.
- [11] H. Zhang, M. P. Bonacina, and J. Hsiang. Psato: a distributed propositional prover and its application to quasigroup problems. Journal of Symbolic Computation, 21:543–560, 1996.