Approaches to Parallel SAT Solving

Youssef Hamadi
Microsoft Research, Cambridge, United Kingdom
École Polytechnique, Palaiseau, France

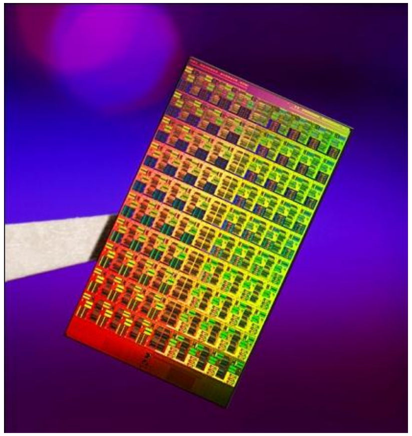
Overview

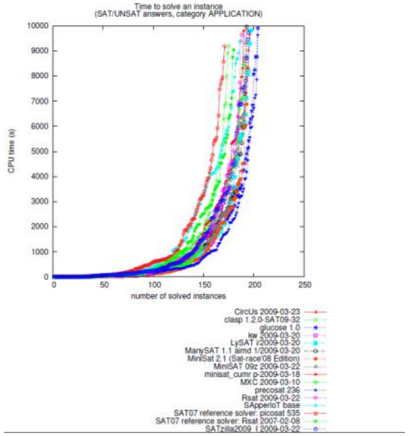
- General Introduction
  - Motivation, defs, parallel relaxation v search
- Knowledge Sharing
  - Control-based sharing
- Deterministic Parallel Search
  - DP2LL
- Summary and Perspectives

Motivation

1. Technological
   - Clock frequency are stalling (thermal wall)
     Sequential software won't be getting faster
   - Transistor are still getting smaller (Moore's law)
     Scalability through more computing units

2. 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





Definitions

- Parallel system: parallel algorithm + parallel architecture
- Scalability: how well a parallel system takes advantage of increased computing resources
  - Definitions:

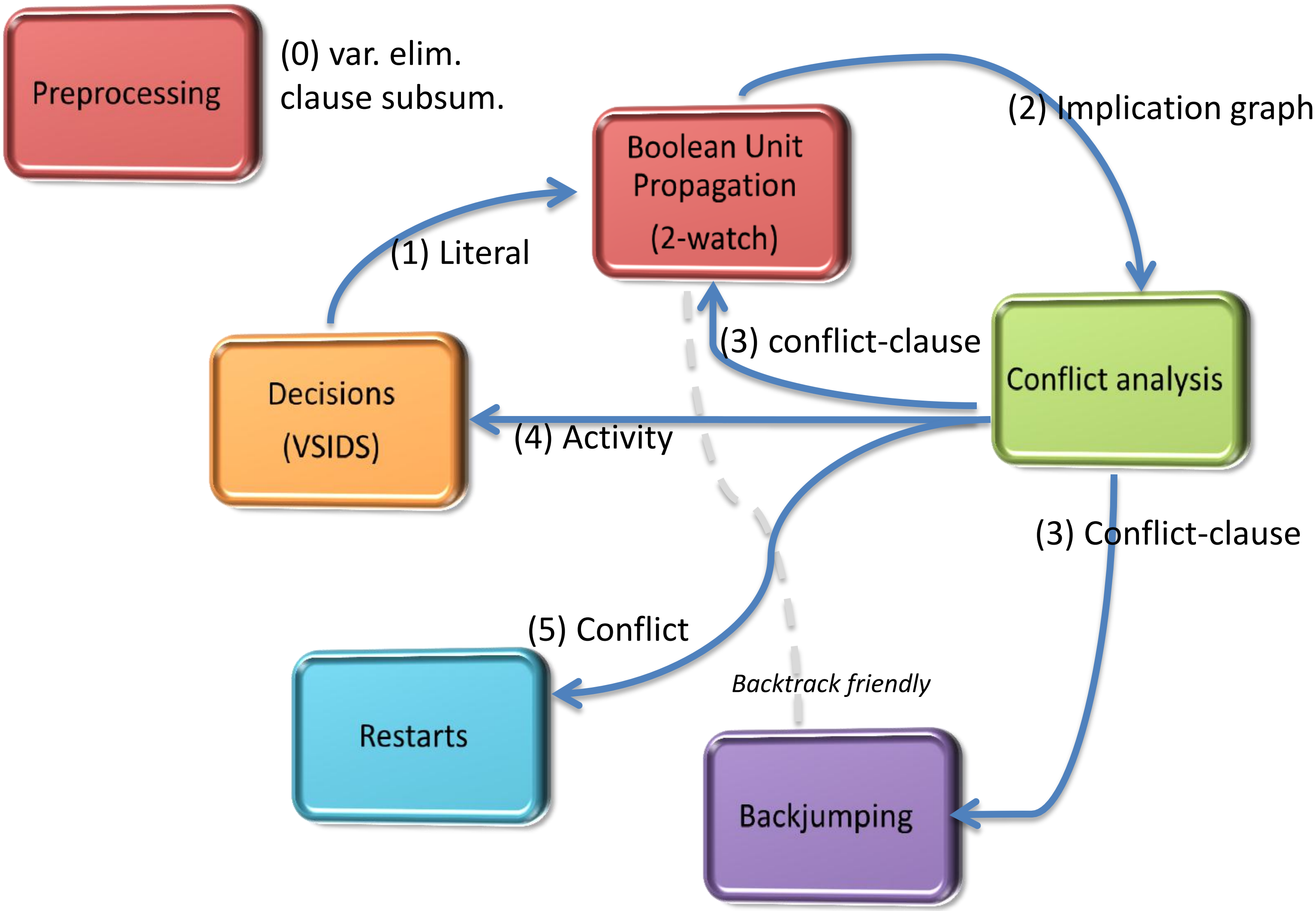
• Sequential runtime	T_s
• Parallel runtime	T_p (with p procs)
• Speedup	$S = T_s/T_p$
• Efficiency	$E = S/p$

  - 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

Sequential SAT Solver



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:

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

$$x1 = \mathit{false} \Rightarrow x2 = \mathit{true} \Rightarrow x3 = \mathit{true} \Rightarrow x4 = \mathit{true} \Rightarrow \ldots$$

- 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)

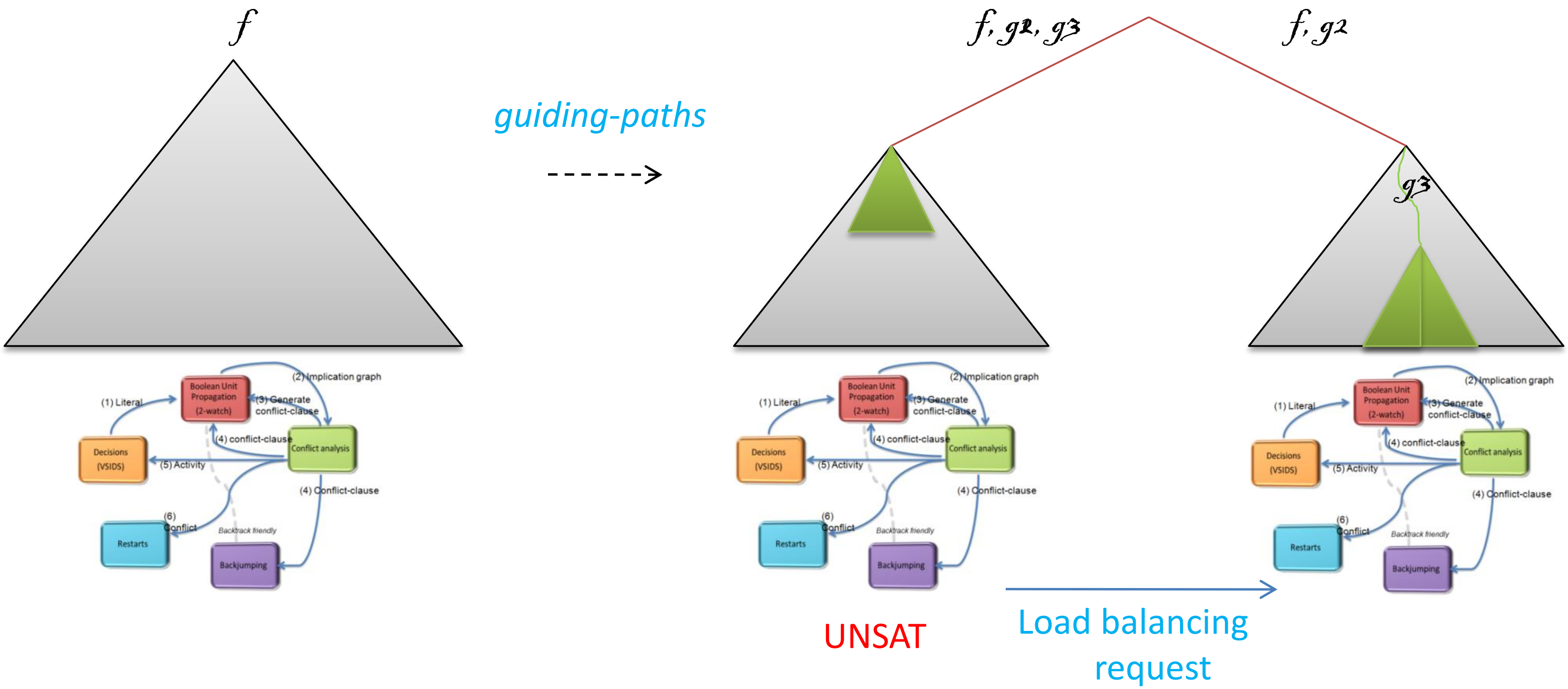
- Parallel algorithm (polynomial number of resources) is unlikely to improve the sequential algorithm by much

PARALLEL SEARCH

Divide and conquer

Principles:

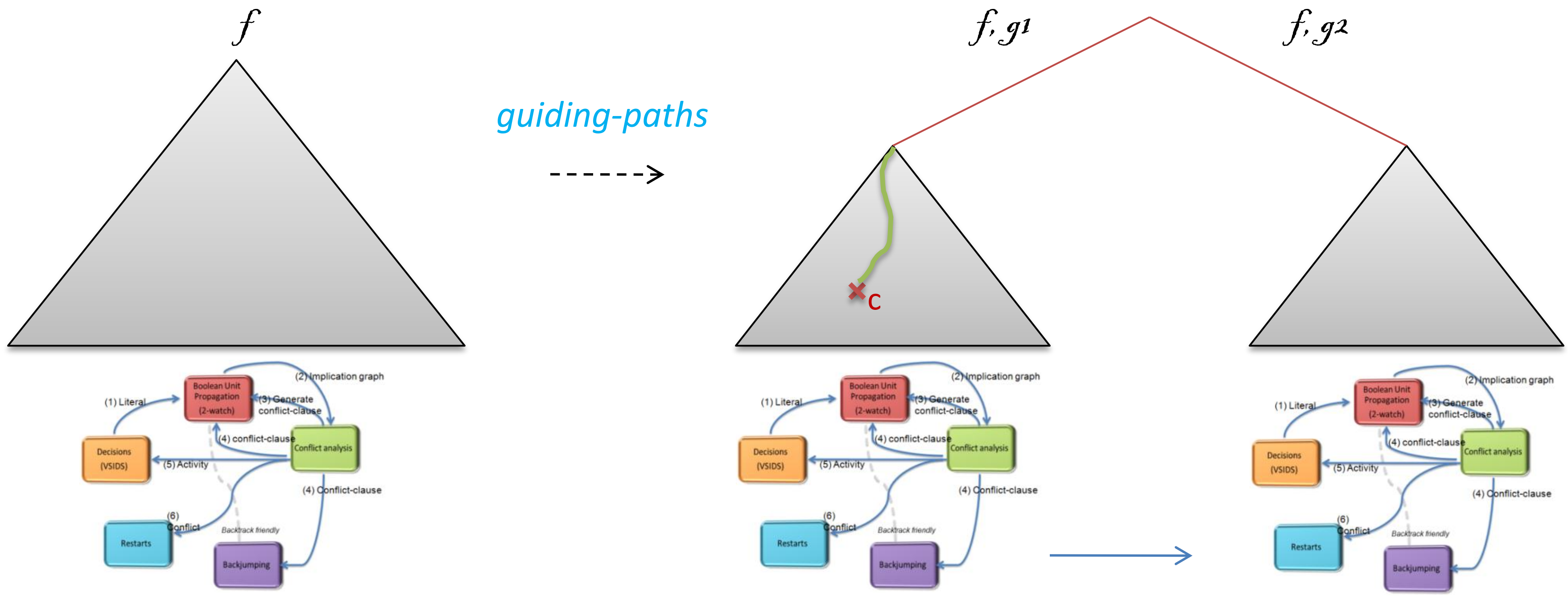
1. Allocate independent subspaces to different resources, organize load-balancing



Divide and conquer

Principles:

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



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

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:
false,
unit,
sat,
other

end, SAT: shared memory variables

An historical approach..

	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

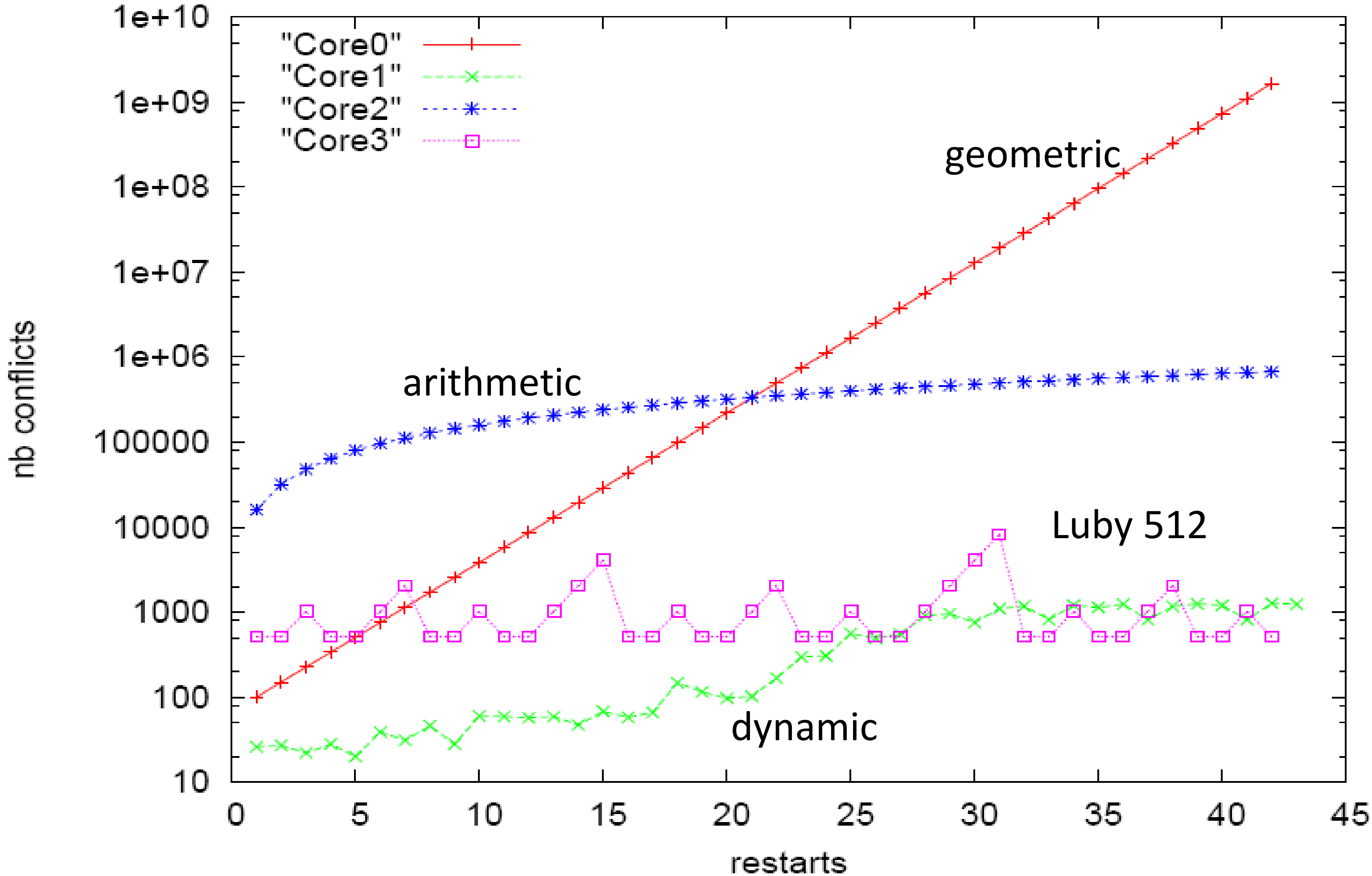
- Portfolio approach: let several *differentiated* but related DPLLs compete and cooperate to be the first to solve a given instance

- Tradeoff:
  - Cover the space of search strategies, i.e., as good as the best
  - Exchange useful information, i.e., better than the best

- State-of-the-art:

  Plingeling [Biere 2010], Antom [Schubert et al. 2010], SArTagnan [Kottler 2010], //z3 [Wintersteiger et al. 2009], ManySAT [Hamadi et al. 2008]

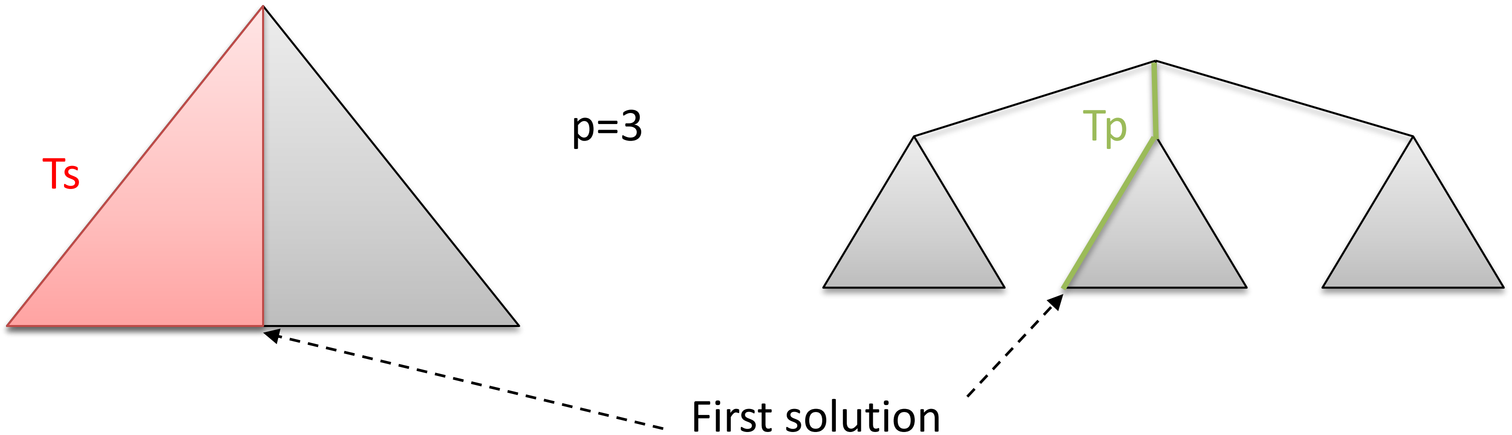
ManySAT detail: restart policies



ManySAT: covering the space of search strategies..

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

Theoretical Performance



Ts >> Tp S >> p E >> 1

- "Speed-up anomalies in parallel tree search", first reported identification circa 1975 [Pruul 88]
- [Rao et al. 93]: "… sequential DFS is sub-optimal…"

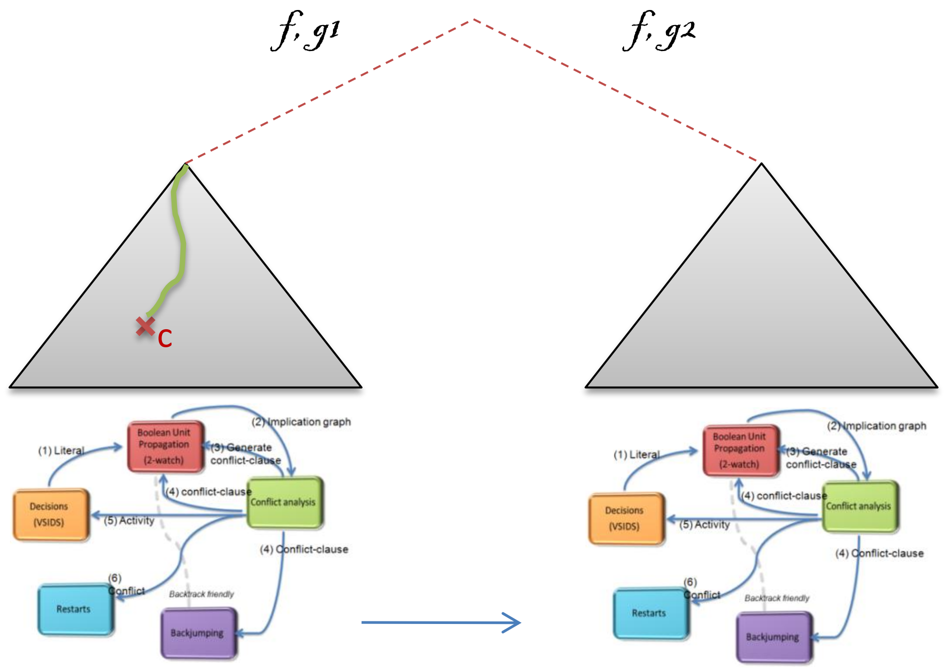
Practical Performance

- 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
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
Runtime variation	**13.7%**	14.7%	15.2%

KNOWLEDGE SHARING

Clause-sharing: classical policy



If $|c| <= e$, export c
(prunes $2^{(n-|c|)}$ tuples)

Clause-sharing: offline tuning

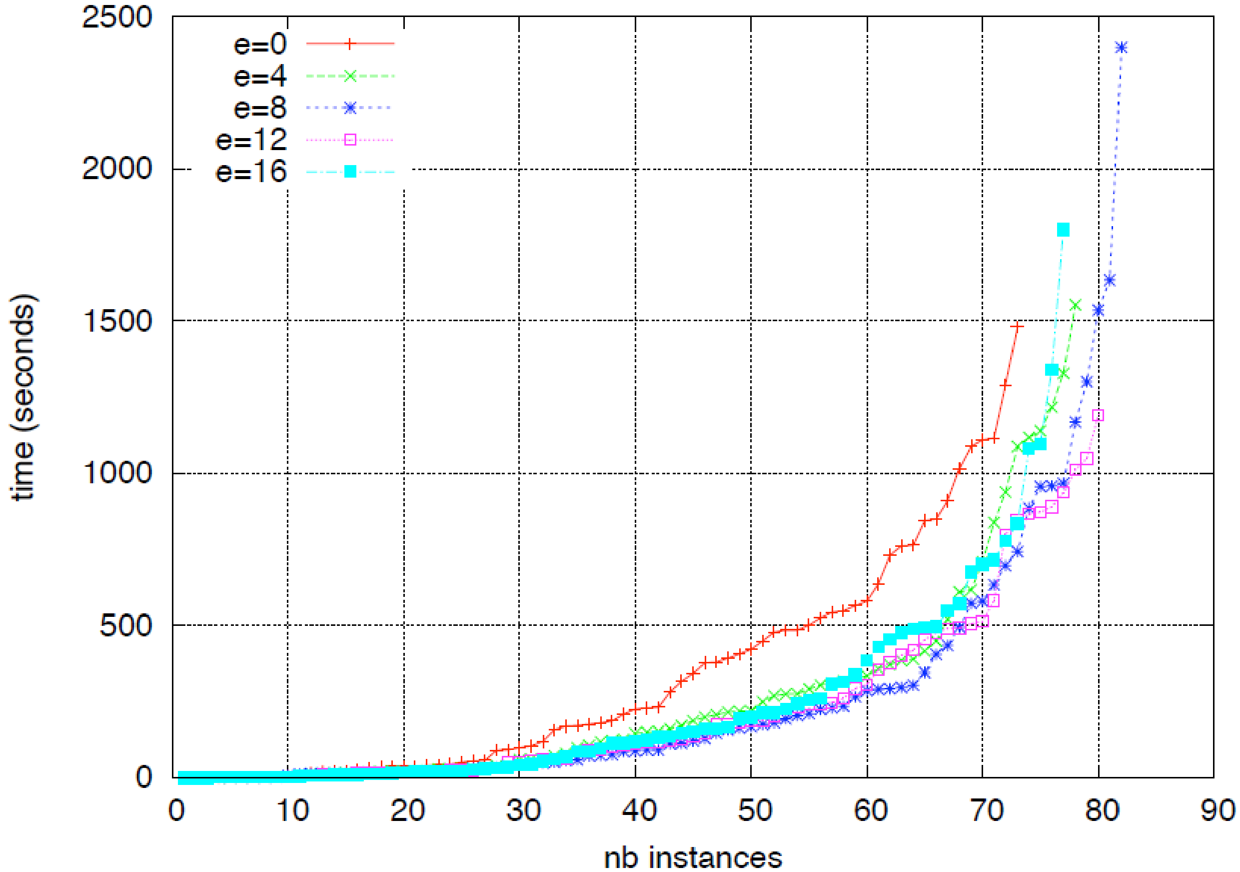
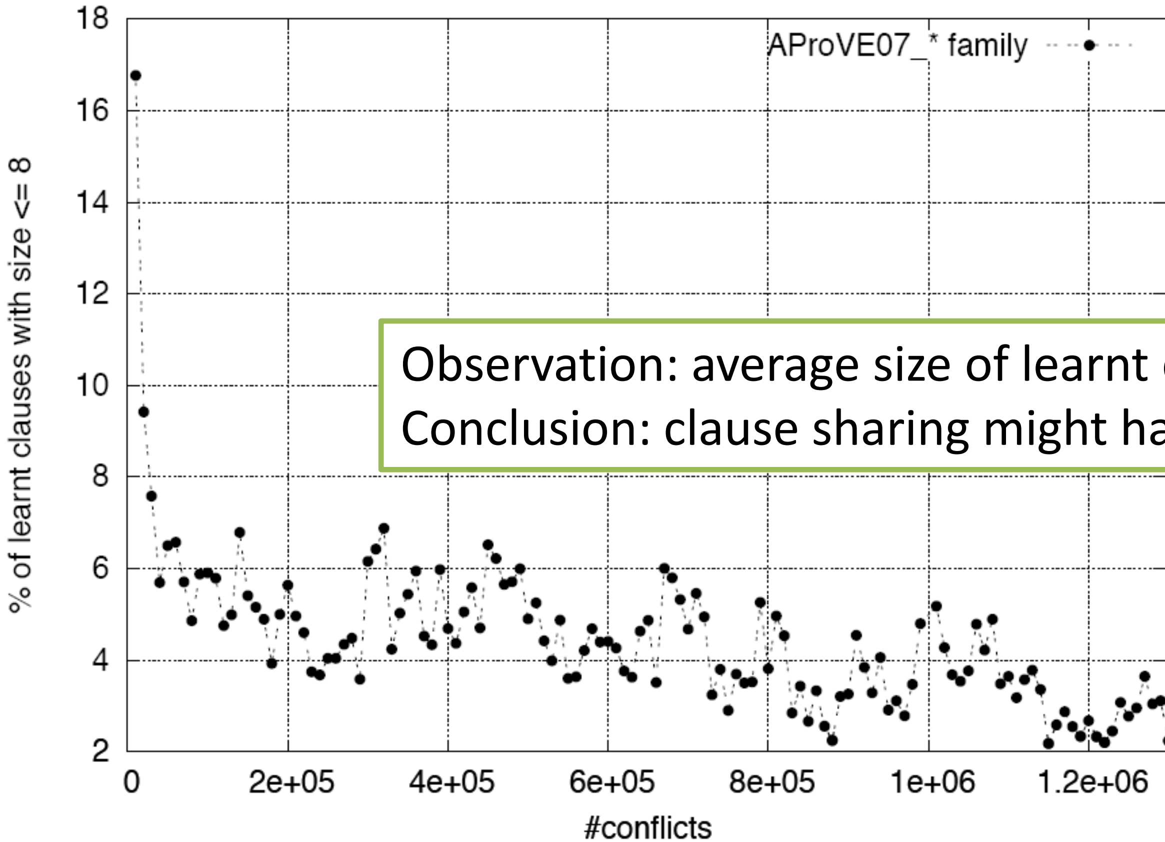


Figure 3. SAT-Race 2008: different limits for clause sharing

Clause-sharing: saturation

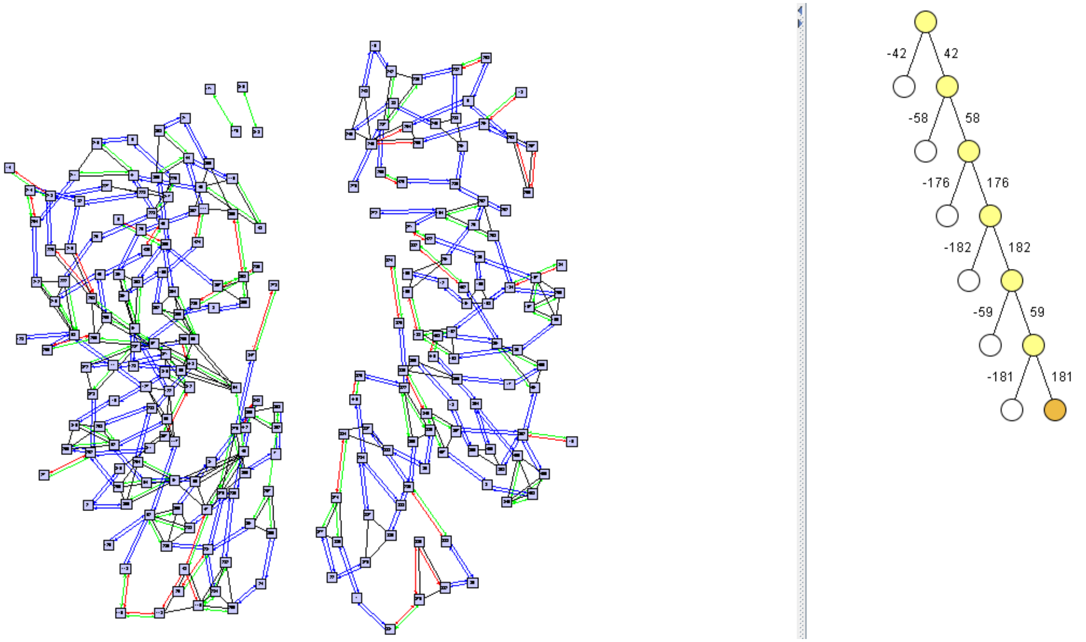
Simple experiment with Minisat 2.0 (sequential):



Observation: average size of learnt clauses is raising
Conclusion: clause sharing might halt

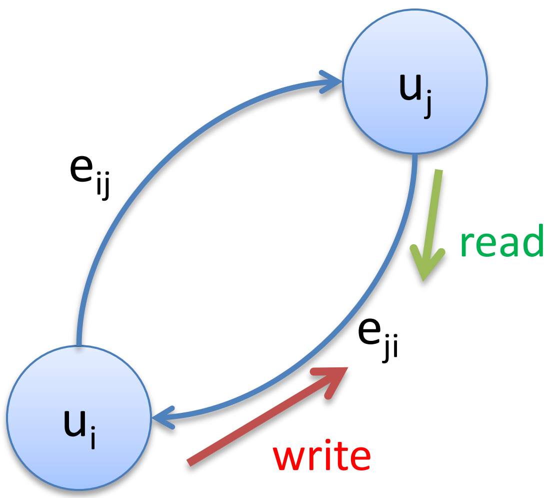
Clause-sharing: relevance

Exchange between unrelated search efforts:



[DPVis, Sinz 05]

Control-based clause-sharing

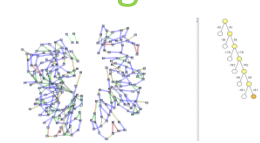


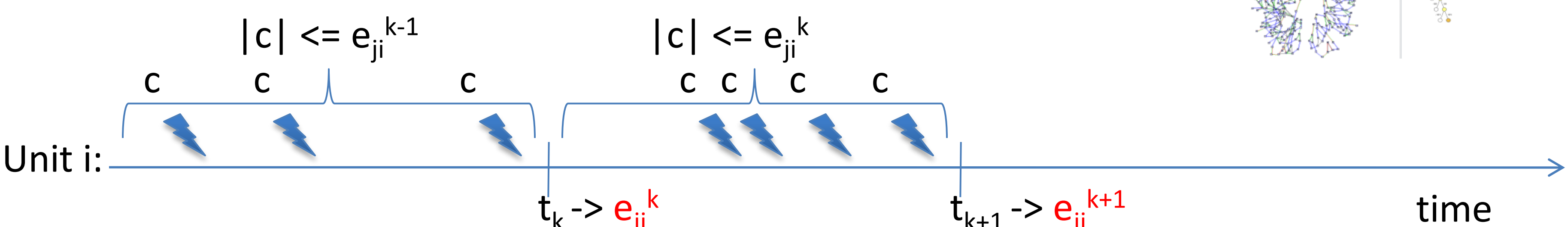
1. Pairwise size limits e_{ij} to control clause sharing from *i* to *j*
2. Each unit performs (lock-free) periodic revisions of incoming limits

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

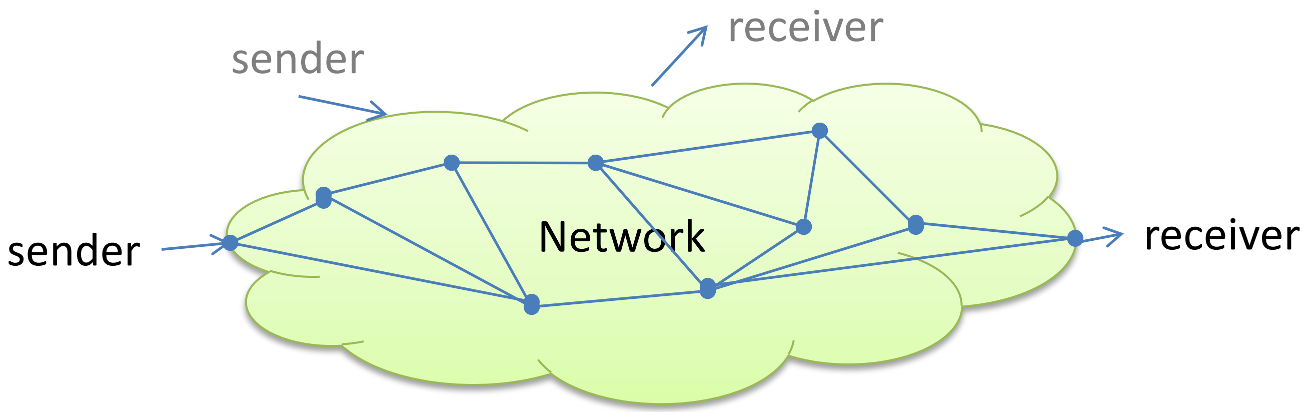
- Throughput T is a number of foreign clauses received in each time interval
  - Time interval = α conflicts
  - Typically, $T = \alpha/c$

- 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^k_{ji} limits
  - If $|R_k| > T$, uniform decrease of e^k_{ji} limits

- How do we update the limits?

TCP Congestion Avoidance

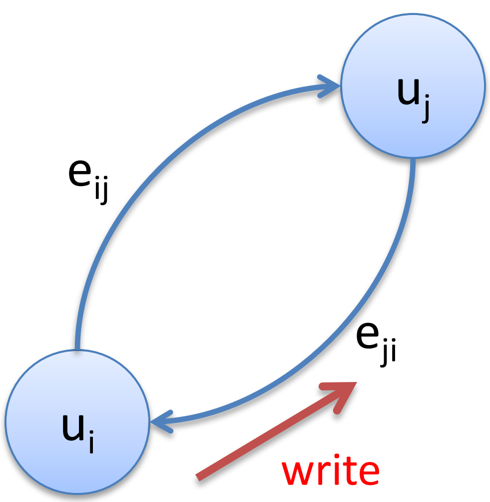
- Problem: guess the available bandwidth, i.e., find the correct communication rate w



- 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)

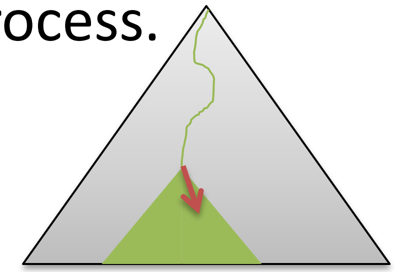
- Clause sharing: an increase of the limits can generate a very large number of incoming clauses.
  - Slow increase, as long as T not met
  - Exponential decrease, if T is met



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

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: $\mathcal{A}_i^{max}$
  - Set of active literals of a foreign clause c:

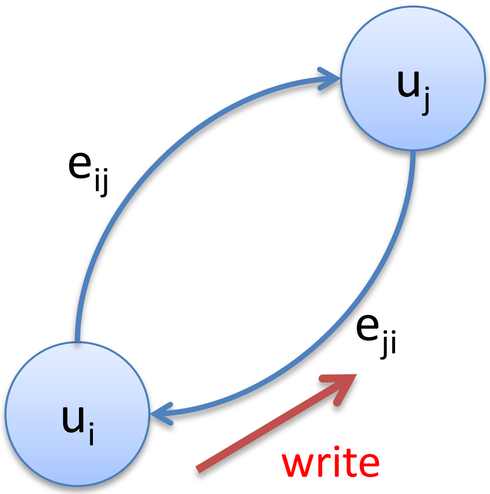
$$\mathcal{L}_{\mathcal{A}_i}(c) = \{x/x \in c \; s.t. \; \mathcal{A}_i(x) \geq \frac{\mathcal{A}_i^{max}}{2}\}$$

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

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

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

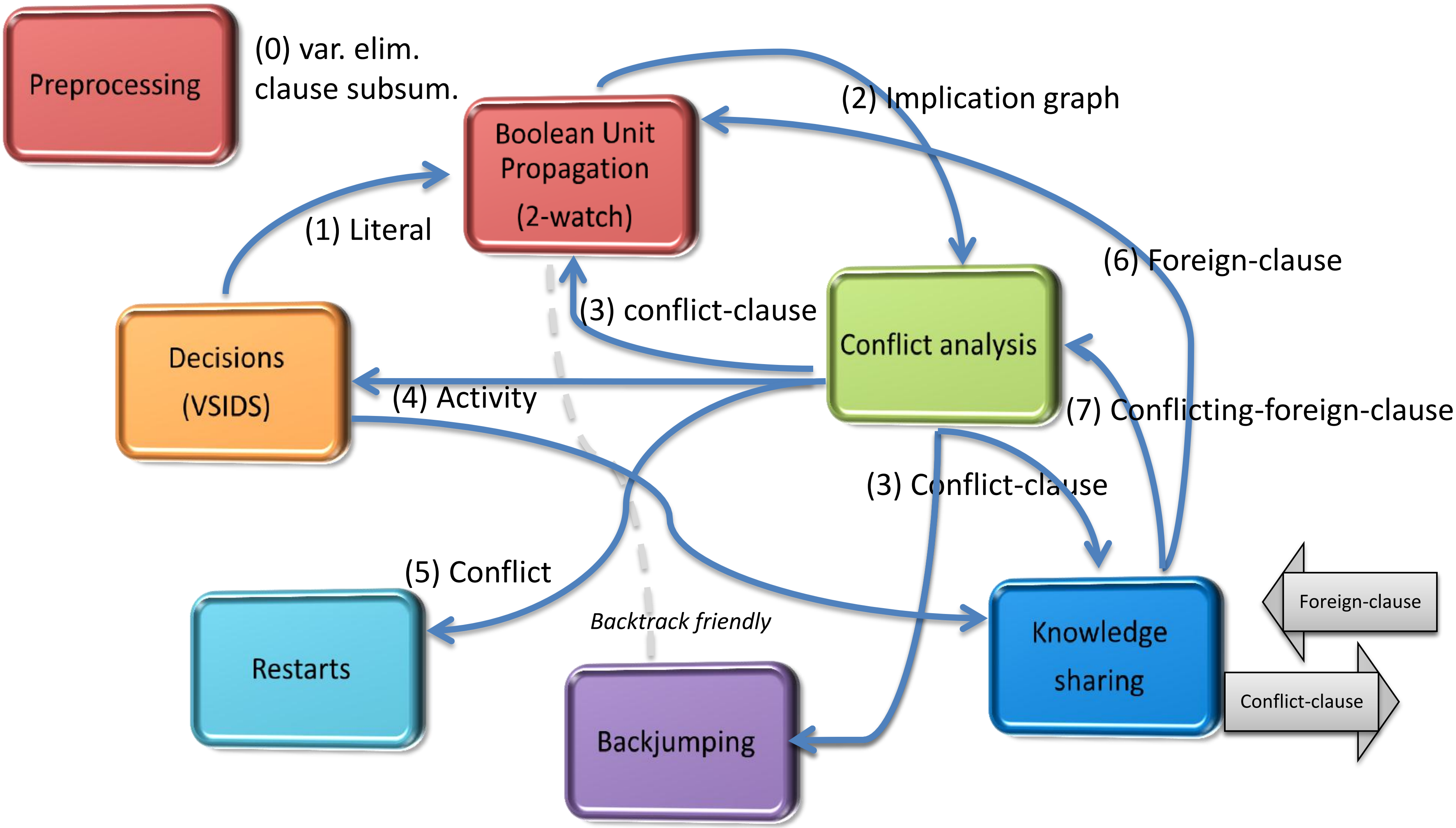
Maintain a Throughput T of Quality Q



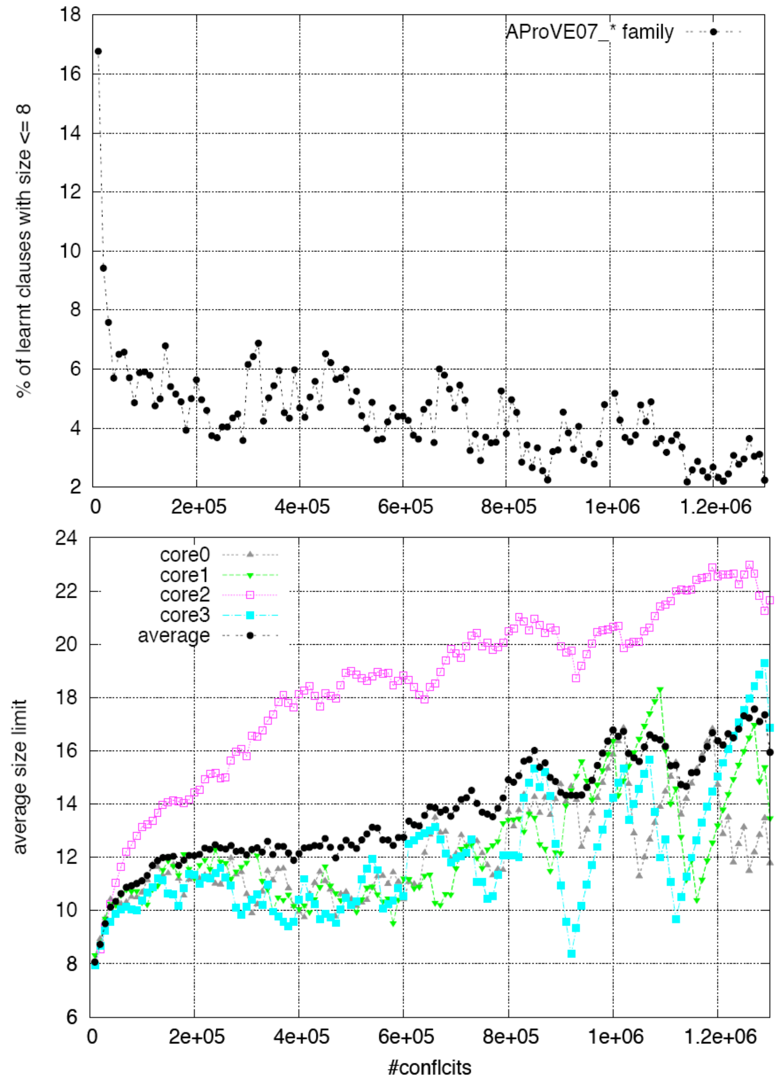
$$aimdTQ(R_i^k)\{$$
$$\forall j | 0 \leq j < n, j \neq i$$
$$e_{j\to i}^{k+1} = \begin{cases} e_{j\to i}^k + (\frac{Q_{j\to i}^k}{100}) \times \frac{b}{e_{j\to i}^k}, if(|R_i^k| < T) \\ e_{j\to i}^k - (1 - \frac{Q_{j\to i}^k}{100}) \times a \times e_{j\to i}^k, if(|R_i^k| > T) \end{cases}$$

- Non uniform increase/decrease:
  - Favour units which give *related* clauses

Parallel SAT Solving



Evaluation: saturation



Evaluation: 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

DETERMINISTIC PARALLEL DPLL $(DP)^2LL$

Motivation

Instance	**nbVars**	**nbModels (diff)**	**n$\bar{H}$**	**avgTime (σ)**
12pipe_bug8	117526	10 (1)	0	2.63 (53.32)
ACG-20-10p1	381708	10 (10)	1.42	1452.24 (40.61)
AProVE09-20	33054	10 (10)	33.84	19.5 (9.03)
dated-10-13-s	181082	10 (10)	0.67	6.25 (9.30)
gss-16-s100	31248	10 (1)	0	38.77 (18.75)
gss-19-s100	31435	10 (1)	0	441.75 (35.78)
gss-20-s100	31503	10 (1)	0	681 (58.27)
itox_vc1138	150680	10 (10)	26.62	0.65 (22.99)
md5_47_4	65604	10 (10)	34.8	173.9 (31.03)
md5_48_1	66892	10 (10)	34.76	704.74 (74.65)
md5_48_3	66892	10 (10)	34.16	489.02 (68.96)
safe-30-h30-sat	135786	10 (10)	22.32	0.37 (0.79)
sha0_35_1	48689	10 (10)	33.18	45.4 (21.88)
sha0_35_2	48689	10 (10)	33.25	61.65 (29.93)
sha0_35_3	48689	10 (10)	32.76	72.21 (21.93)
sha0_35_4	48689	10 (10)	33.2	105.8 (35.22)
sha0_36_5	50073	10 (10)	34.19	488.16 (58.58)
sortnet-8-ipc5-h19-sat	361125	4 (4)	15.86	2058.39 (47.5)
total-10-19-s	331631	10 (10)	0.5	5.31 (6.75)
UCG-20-10p1	259258	10 (10)	2.12	768.17 (31.63)
vmpc_27	729	10 (2)	2.53	11.95 (32.62)
vmpc_28	784	10 (2)	3.67	34.61 (25.92)
vmpc_31	961	8 (1)	0	583.36 (88.65)

- Satisfiable instances, SAT Race 2010
- ManySAT 1.1, 10 runs
  - Nb of different solutions
  - Normalized Hamming distance between solutions
  - Avg. time, std-dev
- Sources of non determinism:
  1. Integration of foreign clauses
  2. Report of termination

Deterministic Parallel DPLL

Algorithm 1: Deterministic Parallel DPLL

```
Data: A CNF formula F;
Result: true if F is satisfiable; false otherwise
begin
    <inParallel, 0 ≤ i < nbCores>
        answer[i] = search(core_i) ;
    for (i = 0; i < nbCores; i++) do
        if (answer[i]! = unknown) then
            return answer[i];
end
```

Algorithm 2: search(core$_i$)

```
    nbConflicts=0;
    while (true) do
        if (!propagate()) then
            nbConflicts++;
            if (topLevel) then
                answer[i]= false;
                goto barrier_1;
            learntClause=analyze();
            exportExtraClause(learntClause);
            backtrack();
            if (nbConflicts % period == 0) then
                barrier_1: <barrier>
                if (∃j|answer[j]! = unknown) then
                    return answer[i];
                updatePeriod();
                importExtraClauses();
                <barrier>
        else
            if (!decide()) then
                answer[i]= true;
                goto barrier_1;
```

1. Controlled termination

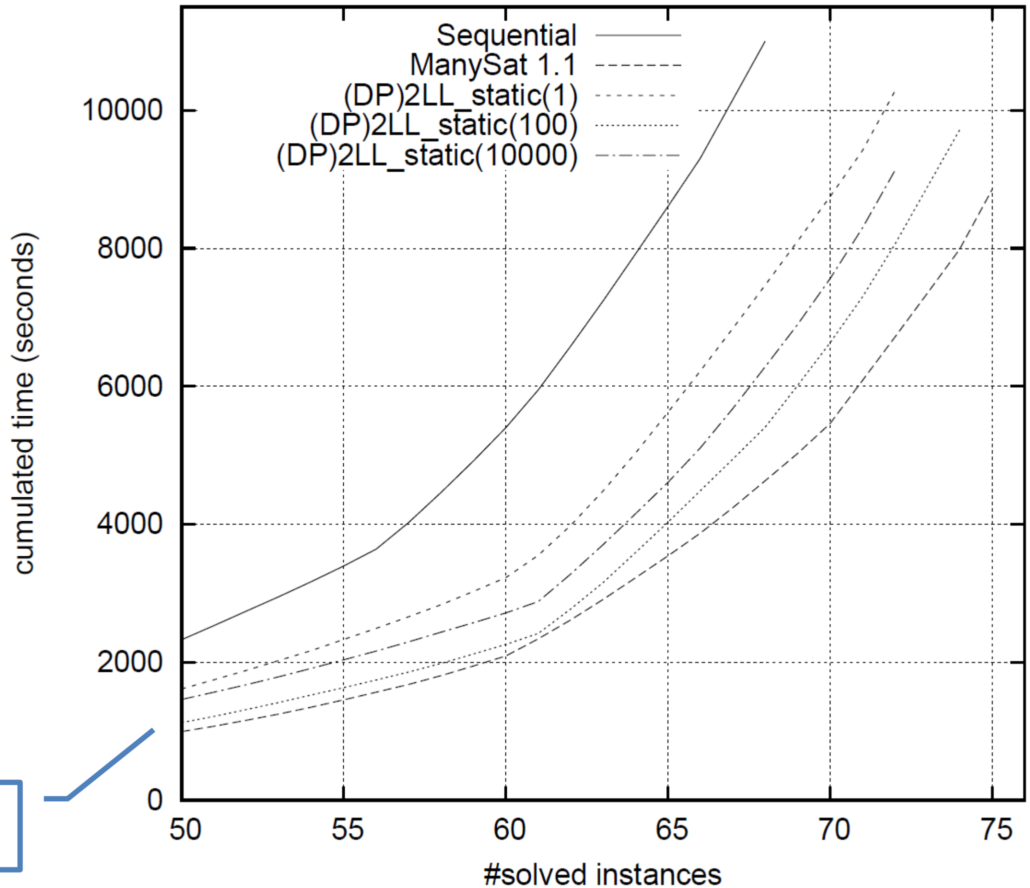
2. Controlled integration of foreign clauses

Deterministic Parallel DPLL

Trade off small/large period:

- Early/late integration of foreign clauses
- Large/small cumulated waiting time at the barriers

Real time!

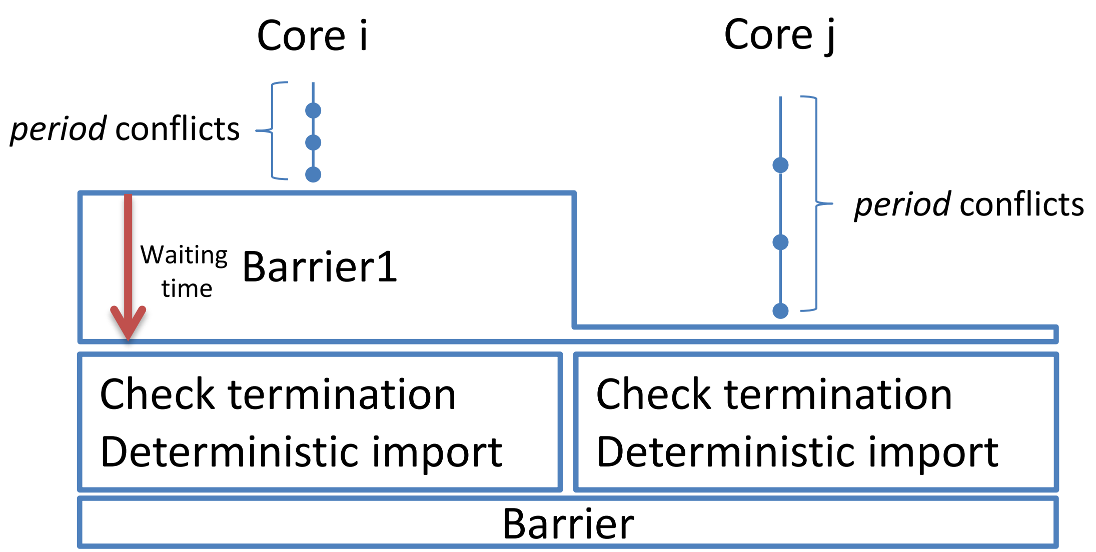


Understanding the waiting time

Observation: Cores run at different *speed*

Explanation:

- They develop different trees, i.e., reach conflicts at different rates
- Develop different learnt-bases, and therefore use more or less time to reach conflicts



Reducing the waiting time

- Idea: arrive at the same time at the barrier
- Each core has its own dynamically adjusted *period:*
  - *Slow* cores can use a small period (less conflicts)
  - *Fast* cores can use a large period (more conflicts)
- How can we estimate their relative speeds?
- Observation: Large learnt-clause db -> slow unit propagation -> slow conflict generation
- Proposal: use the size of learnt base to estimate the relative speed of the cores.

Reducing the waiting time

Synchro step k,

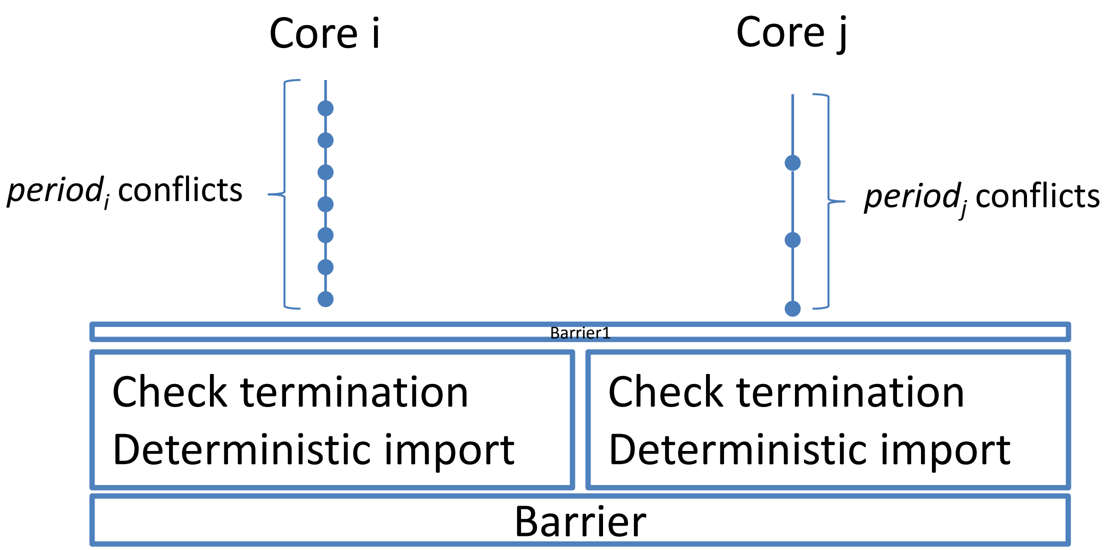
Maximum db size, $m = \max\left(|\Delta_j^k|\right) \forall 0 \leq j < nbCores$

Core$_i$, relative speed, $S_i^k = \frac{|\Delta_i^k|}{\mathrm{m}}$

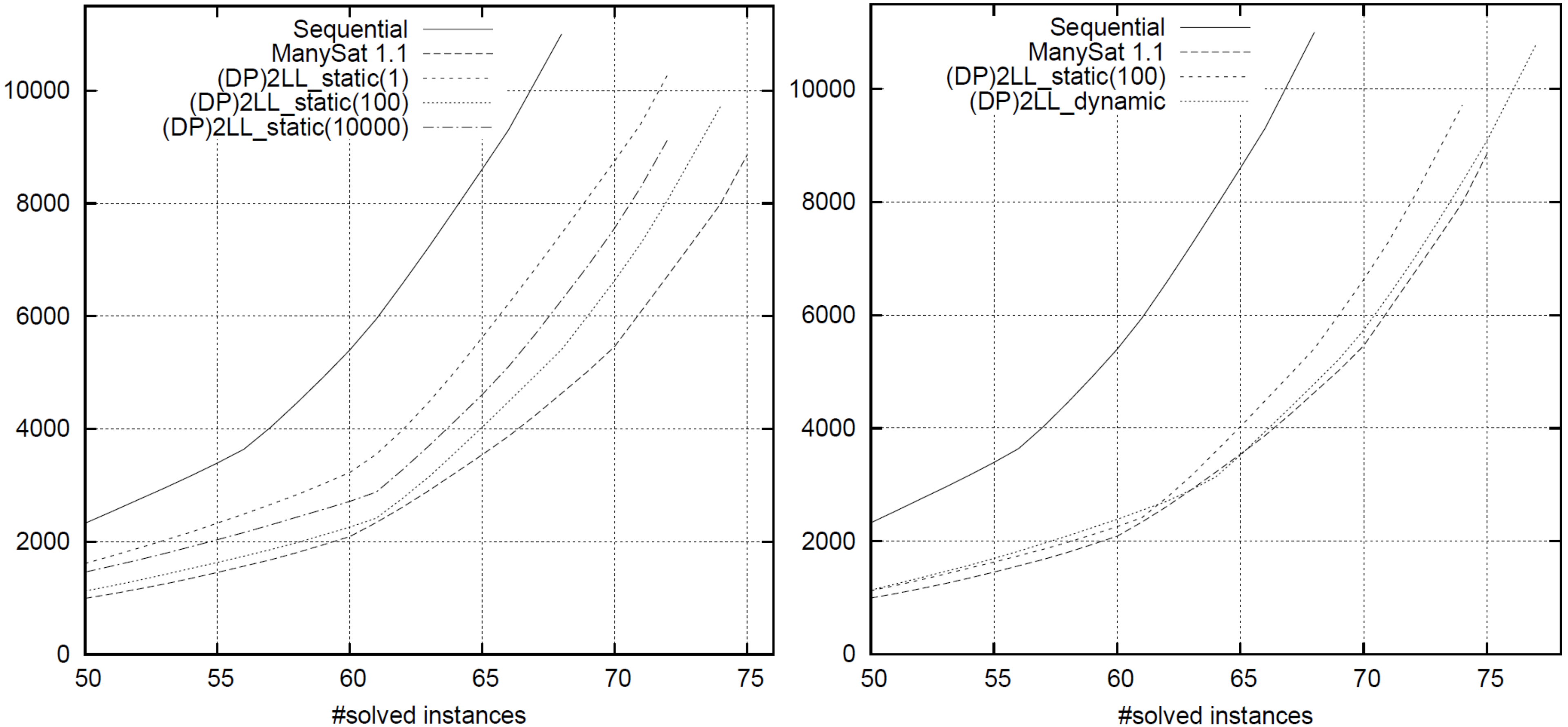
Period for next step, $period_i^{k+1} = \alpha + \left(1 - S_i^k\right) \times \alpha$

- *relatively slow,* S_i^k -> 1, $period_i^{k+1}$ -> α
- *relatively fast,* S_i^k -> 0, $period_i^{k+1}$ > α

Reducing the waiting time



Static v Dynamic periods



Summary

- Divide-and-conquer: an historical approach..
  - Works very well for deterministic tasks
  - Standpoint: in worst-case exhaust the space
- Portfolios: the current approach
  - Made by people with a Search background
  - Standpoint: let’s try to avoid being wrong by multiplying strategies
- Knowledge sharing
  - Portfolio becomes better than individual strategies
  - Difficulty: orthogonal strategies v sharing
  - Can be dynamically adjusted
- Deterministic Parallel Search
  - DP2LL: can be done efficiently

Perspectives

- V

10

MSR/INRIA Paris joint-lab

Some references

- On the parallel complexity of discrete relaxation in constraint satisfaction networks, S. Kasif, AIJ Volume 45, Issue 3, 1990.
- Optimal Distributed Arc-Consistency, Y. Hamadi, Fifth International Conference on Principles and Practice of Constraint Programming (**CP'99**), p219-233, Springer, October 1999.
- On the Efficiency of Parallel Backtracking, V. Rao, and V. Kumar, IEEE Transactions on Parallel and Distributed Systems, Volume 4, Issue, 4, 1993.
- A fast parallel sat-solver with efficient workload balancing, M. Bohm and E. Speckenmeyer, Ann. Math. Artif. Intell., 17(3-4):381-400, 1996.
- GrADSAT: A parallel sat solver for the grid, W. Chrabakh and R.Wolski, Technical report, UCSB Computer Science Technical Report Number 2003-05, 2003.
- Parallel propositional satisfiability checking with distributed dynamic learning, W. Blochinger, C. Sinz, and W. Kuchlin. Parallel Computing, 29(7):969-994, 2003.
- A universal parallel SAT checking kernel, W. Blochinger, C. Sinz, and W. Küchlin, *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications PDPTA 03*, volume 4, pages 1720-1725, 2003.
- ManySAT: a Parallel SAT Solver, Y. Hamadi, S. Jabbour, and L. Sais, Int. Journal on Satisfiability, Boolean Modeling and Computation (**JSAT**), Volume 6, Special Issue on Parallel SAT, Ed. Y. Hamadi, IOS Press, 2009.
- Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010, A. Biere, Technical Report **10/1**, August 2010, FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria.
- 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.
- A Concurrent Portfolio Approach to SMT Solving, C. Wintersteiger, Y. Hamadi, and L. de Moura, Twenty-one International Conference on Computer Verification (**CAV'09**), June 2009, Grenoble, France.
- 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**).
- Deterministic Parallel DPLL (DP2LL), Y. Hamadi, S. Jabbour, C. Piette, and L. Sais, MSR-TR-2011-47.
- Improving Parallel Local Search for SAT, A. Arbelaez, Y. Hamadi, Learning and Intelligent Optimization (**LION'11**), Roma, Italy.