Aalto University
School of Science
and Technology

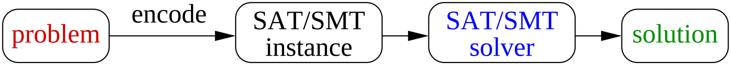
Parallel SAT Solving in a Grid

Tommi Junttila
Joint work with Antti Hyvärinen and Ilkka Niemelä

Department of Information and Computer Science
Aalto University, School of Science
Tommi.Junttila@tkk.fi

Deduction at Scale seminar, Ringberg Castle, Germany, March 7–11, 2011

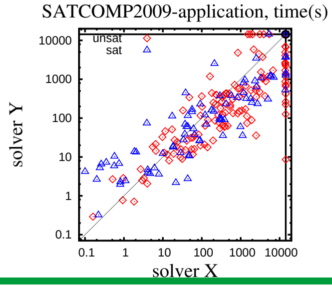
SAT Solvers

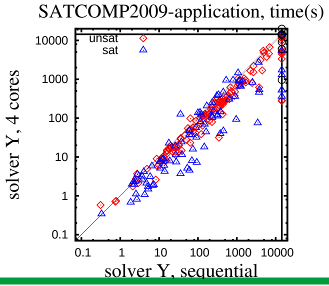


SAT/SMT solvers used when solving other computationally hard problems (verification, planning, etc)

Making SAT solvers run faster:

- Improve deductive power, algorithms, or data structures of solvers
- Use faster running processors (MHz rates not increasing as in past)
- Parallelize to exploit multi-core processors, clusters, grids





Context and Goals

Parallel satisfiability solving

- of hard SAT instances
- in a loosely-coupled computational Grid
- by using randomization, clause learning, and partitioning

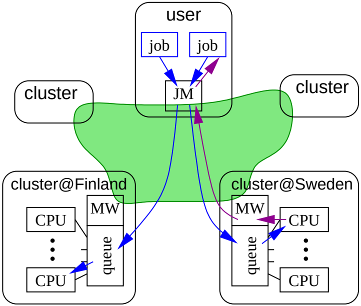
Some goals:

- to be able to exploit existing sequential SAT solvers with as small changes as possible
- to better understand the roles of and interactions between randomization, partitioning, and learning
- to solve previously unsolvable SAT instances

Outline

- Computing environment: a Grid
- Parallelizing SAT solvers:
  1. Framework I: portfolios with clause sharing
  2. Framework II: search space partitioning
- Conclusions

Computing Environment: a Grid



- NorduGrid: a set of clusters of CPUs
- Hundreds of CPUs available via a common interface
- Jobs (SAT solver+instance) submitted to job manager (JM), results from JM
- No communication to/from running jobs due to cost, sandboxing etc
- Resource limitations (time, mem) [de-facto] imposed on jobs
- Substantial delays on jobs: queueing, network connection (a SAT instance can be tens of megabytes large), other users
  $\Rightarrow$ typical submission-to-start delay 2–20 minutes!
  $\Rightarrow$ submit 64 jobs and have 10–64 run in parallel, others wait
  $\Rightarrow$ repeatability, measuring scalability etc difficult
- Jobs can fail (a cluster is reset etc)
- Compare this to multi-core environments with short delays, shared memory/MPI communication, indefinitely running threads, ...

Parallelizing SAT solvers
Framework I: portfolios

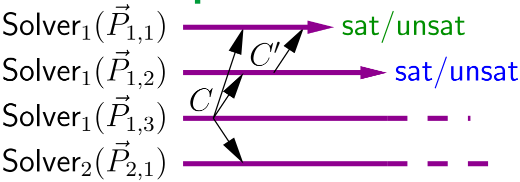
Framework I: portfolios

SAT-Race 2010: framework used in best multi-core SAT solvers

Idea:

- run *n* solvers in parallel ...
  - different solvers or
  - same solver with different parameters
  - solvers *compete*: who solves the problem first?

Framework I: portfolios

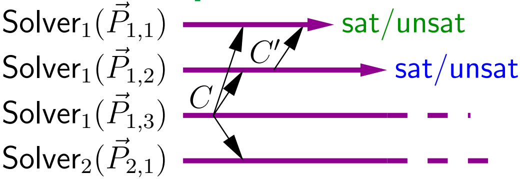


SAT-Race 2010: framework used in best multi-core SAT solvers

Idea:

- run *n* solvers in parallel ...
  - different solvers or
  - same solver with different parameters
  - solvers *compete*: who solves the problem first?
- and share learnt clauses between solvers
  - learnt clauses $\approx$ lemmas found during search
  - current best solvers: conflict-driven clause learning (CDCL) Davis-Putnam-Logemann-Loveland algorithm
  - solvers *co-operate*: avoid mistakes made by others
    $\Rightarrow$ better than the best

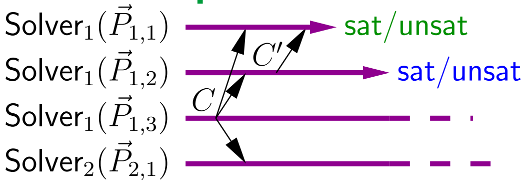
Framework I: portfolios



SAT-Race 2010: framework used in best multi-core SAT solvers

- Plingeling [Biere 2010]:
  n thread copies of lingeling, different random seeds and deduction component scheduling in threads, share unit clauses
- ManySAT [Hamadi, Jabbour & Sais, J.Sat 2009]:
  n threads, differentiate search strategies, share clauses of length at most 8
- SArTagnan [Kottler, Sat-Race 2010] and antom [Schubert, Lewis & Becker, Sat-Race 2010]:
  run different search strategies, clause sharing

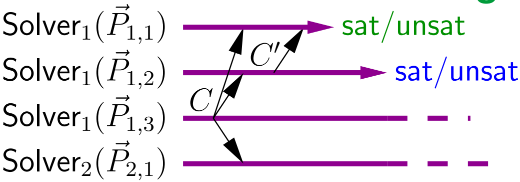
Framework I: portfolios



Some other references

- //Z3 [Wintersteiger, Hamadi & de Moura, CAV 2009]: *n* threads, differentiate SAT search strategies and run theory solvers in parallel, share clauses of length at most 8
- SATzilla2009 [Xu, Hutter, Hoos & Leyton-Brown, SatComp 2009]: Real algorithm portfolio, select and run different SAT solvers in parallel
- [Hamadi, Jabbour & Sais, IJCAI 2009]: how to share clauses between solvers

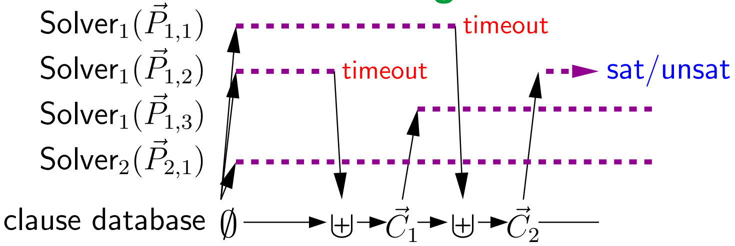
Portfolios with Clause Sharing in a Grid



Problems when applied in our computational environment:

- No communication to/from running jobs
- Resource limits imposed on jobs: jobs must terminate within a predefined time limit (e.g. 1–4 hours)

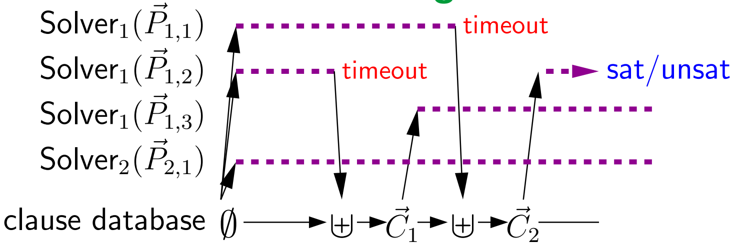
Portfolios with Clause Sharing in a Grid



An approach: [Hyvärinen, Junttila & Niemelä, J.Sat 2009]

- Maintain a master database $\vec{C}$ of learnt clauses
- Clause sharing only when a solver starts or timeouts
  - Start: import a part $\vec{D}$ of the database permanently into solver's instance, i.e. solve $\phi \wedge \vec{D}$ instead of ϕ
  - Timeout: merge (a subset of) current learnt clauses into the database [and simplify with unit propagation etc]
- "Cumulative parallel learning with hard restarting solvers"

Portfolios with Clause Sharing in a Grid



Some design issues:

- How large should the master clause database be?

  We allowed at most 1M literals, should expand gradually

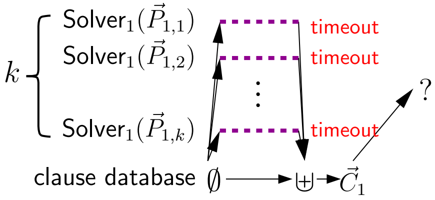
- Which clauses should be imported/merged?

  We evaluated random, length-based (keep shortest clauses), and frequency-based (keep most frequent) filtering; should use frequency-based but length-based easier to implement

  Imported/merged at most 100k literals

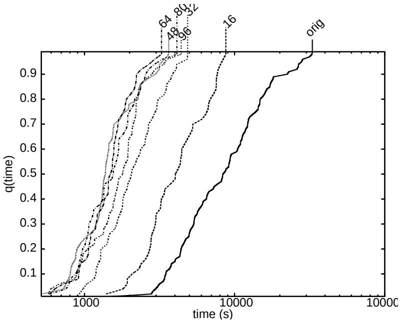
  See [Hyvärinen, Junttila & Niemelä, J.Sat 2009] for further analysis

Portfolios with Clause Sharing in a Grid

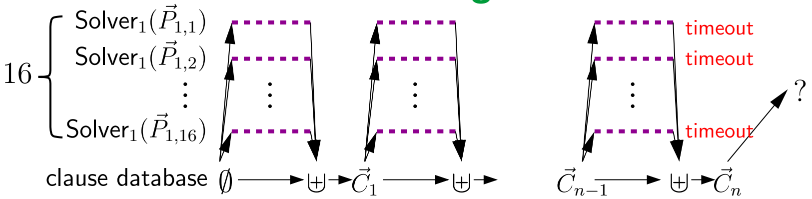


Controlled experiment: number of solvers run in parallel

- One “round” of parallel learning
- Instance manol-pipe-f9b solver Minisat 1.14
- Each solver run 25% of the minimum run time, with different seed
- Length-based filtering
- Plot shows cumulative run-time distributions: instance solved 50 times with different prng seeds

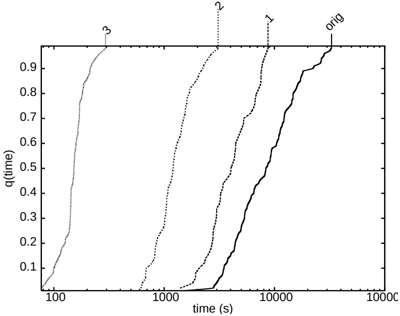


Portfolios with Clause Sharing in a Grid



Controlled experiment: number of rounds

- Cumulative effect of parallel learning
- Instance manol-pipe-f9b, solver Minisat 1.14
- 16 solvers in each round
- Each solver run 25% of the minimum run time
- Length-based filtering



Portfolios with Clause Sharing in a Grid

Wall clock times for some difficult instances from SAT-Comp 2007

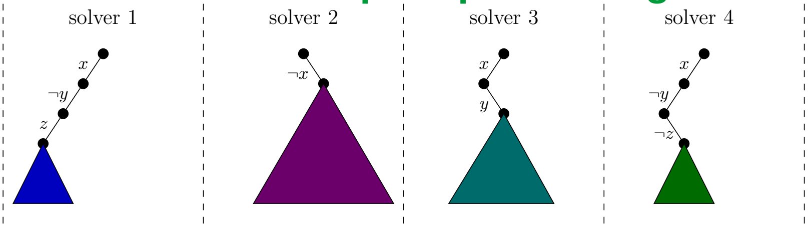
- Grid: at most 64 Minisat 1.14 solvers in parallel, 1 hour time limit per solver, 3 days time limit in total
- Sequential: sequential Minisat 1.14, no time limit, mem limit 2GB

Solved by some solver in SAT 2007 but not by Minisat 1.14			
Name	Type	Grid (s)	sequential (s)
ezfact64_5.sat05-452.reshuffled-07	SAT	**4,826**	65,739
vmpc_33	SAT	**669**	184,928
safe-50-h50-sat	SAT	**12,070**	m.o.
connm-ue-csp-sat-n800-d-0.02-s1542454144.sat05-533.reshuffled-07	SAT	**5,974**	119,724

Not solved by any solver in SAT 2007			
Name	Type	Grid (s)	sequential (s)
AProVE07-01	UNSAT	**13,780**	39,627
AProVE07-25	UNSAT	**94,974**	306,634
QG7a-gensys-ukn002.sat05-3842.reshuffled-07	UNSAT	**8,260**	127,801
vmpc_34	SAT	**3,925**	90,827
safe-50-h49-unsat		t.o.	m.o.
partial-10-13-s.cnf	SAT	**7,960**	m.o.
sortnet-8-ipc5-h19-sat		t.o.	m.o.
dated-10-17-u	UNSAT	**11,747**	105,821
eq.atree.braun.12.unsat	UNSAT	**9,072**	59,229

Parallelizing SAT solvers
Framework II: search space partitioning

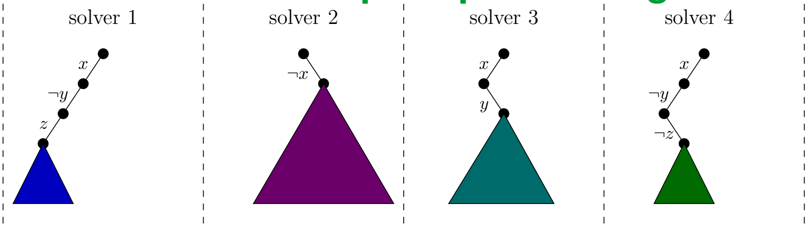
Framework II: search space partitioning



In multi-core, p2p network environments:

- Guiding paths ($\approx$ first search tree decisions) to make solvers explore different parts of the search space
- Dynamic load balancing by splitting guiding paths
- Search ends when a solution is found or the whole search space is covered
- On-the-fly clause sharing also possible

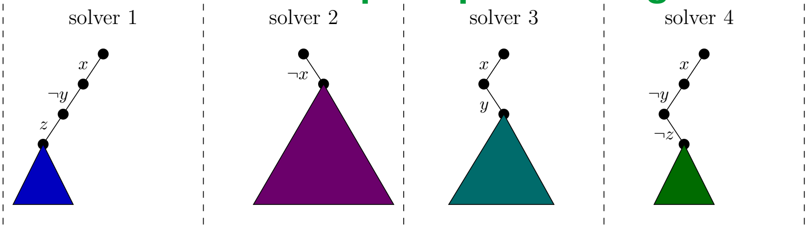
Framework II: search space partitioning



In multi-core, p2p network environments:

- [Blochinger, Sinz & Küchlin, Par.Comp. 2003]
- ZetaSAT [Blochinger, Westje, Küchlin & Wedeniwski, IEEE CCGrid 2005]
- Satciety [Schulz & Blochinger, WPSS 2010]
- GridSAT [Chrabakh & Wolski, Par.Comp 2006]
- MiraXT [Lewis, Schubert & Becker, ASP-DAC 2007]
- PaMiraXT [Lewis, Schubert & Becker, J.SAT 2009]
- pMinisat [Chu & Stuckey, Sat-Race 2008]

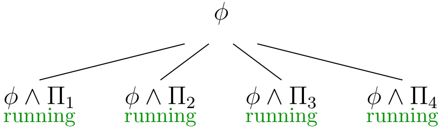
Framework II: search space partitioning



A problem in our environment:

- Dynamic load balancing by splitting guiding paths not possible

Partitioning Functions



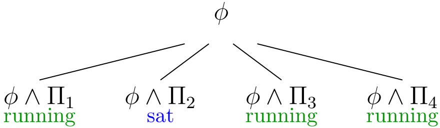
- Static partitioning to avoid communication to/from running solvers
- Partition the instance ϕ into n model-disjoint derived instances $\phi \wedge \Pi_1, ..., \phi \wedge \Pi_n$ and solve them in parallel in Grid
- A partitioning function $\mathcal{P}$ maps a formula ϕ to a set

$$\mathcal{P}(\phi) = \{\Pi_1, ..., \Pi_k\}$$

of generic partitioning constraints such that

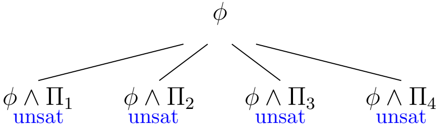
1. $\phi \equiv (\phi \wedge \Pi_1) \vee ... \vee (\phi \wedge \Pi_k)$ (equivalence)
2. $\phi \wedge \Pi_i \wedge \Pi_j$ is unsat if $i \neq j$ (disjoint models)

Partitioning Functions



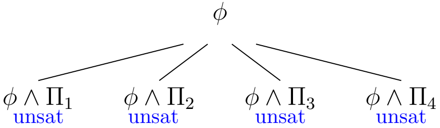
- If ϕ is satisfiable, it is enough to find a solution in one of the derived instances

Partitioning Functions



- If ϕ is unsatisfiable, must show all derived instances unsatisfiable

Partitioning Functions



- If ϕ is unsatisfiable, must show all derived instances unsatisfiable
- Assume a void partitioning function $\mathcal{P}$ that produces derived instances as hard as the original one
- For instance, Π_is constrain an easy part of ϕ or variables not in an unsat core
- When the number n of derived instances and parallel solvers is increased, the expected run-time of the parallel approach tends to the maximum run-time of the original instance ("increasing bad luck")
  That is, more parallelism $\Rightarrow$ run times can get worse

Implementing Partitioning Functions

- VSIDS Scattering [Hyvärinen, Junttila & Niemelä, SAT 2006]: run minisat, restart, select best branching literals as unit constraints, repeat with negated constraint included

Implementing Partitioning Functions

- VSIDS Scattering [Hyvärinen, Junttila & Niemelä, SAT 2006]: run minisat, restart, select best branching literals as unit constraints, repeat with negated constraint included
  1. run Minisat for x seconds on ϕ
  2. output $\Pi_1 = (x_1) \wedge (\neg x_{17})$
  3. run Minisat for x seconds on $\phi \wedge (\neg x_1 \vee x_{17})$
  4. output $\Pi_2 = (\neg x_1 \vee x_{17}) \wedge (x_3) \wedge (\neg x_{90})$
  5. run Minisat for x seconds on $\phi \wedge (\neg x_1 \vee x_{17}) \wedge (\neg x_3 \vee x_{90})$
  6. output $\Pi_3 = (\neg x_1 \vee x_{17}) \wedge (\neg x_3 \vee x_{90}) \wedge (x_{150})$
     output $\Pi_4 = (\neg x_1 \vee x_{17}) \wedge (\neg x_3 \vee x_{90}) \wedge (\neg x_{150})$

Implementing Partitioning Functions

- VSIDS Scattering [Hyvärinen, Junttila & Niemelä, SAT 2006]: run minisat, restart, select best branching literals as unit constraints, repeat with negated constraint included
- Lookahead DPLL partitioning function [Hyvärinen, Junttila & Niemelä, LPAR-17 2010]
  - Non-learning lookahead DPLL (e.g. satz, march)
  - The partial truth assignments at log n level nodes are the partitioning constraints
  - A new method for speeding up failed literal rule detection

Implementing Partitioning Functions

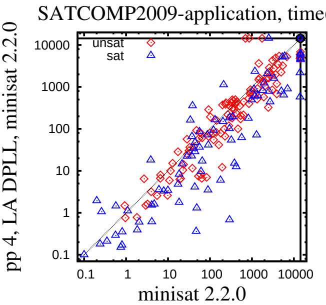
- VSIDS Scattering [Hyvärinen, Junttila & Niemelä, SAT 2006]: run minisat, restart, select best branching literals as unit constraints, repeat with negated constraint included
- Lookahead DPLL partitioning function [Hyvärinen, Junttila & Niemelä, LPAR-17 2010]
  - Non-learning lookahead DPLL (e.g. satz, march)
  - The partial truth assignments at log n level nodes are the partitioning constraints
  - A new method for speeding up failed literal rule detection
- Lookahead Scattering [Hyvärinen, Junttila & Niemelä, LPAR-17 2010]: same as VSIDS scattering but lookahead-based branching heuristics in Minisat

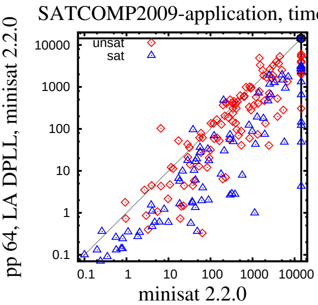
Implementing Partitioning Functions

- VSIDS Scattering [Hyvärinen, Junttila & Niemelä, SAT 2006]: run minisat, restart, select best branching literals as unit constraints, repeat with negated constraint included
- Lookahead DPLL partitioning function [Hyvärinen, Junttila & Niemelä, LPAR-17 2010]
  - Non-learning lookahead DPLL (e.g. satz, march)
  - The partial truth assignments at log *n* level nodes are the partitioning constraints
  - A new method for speeding up failed literal rule detection
- Lookahead Scattering [Hyvärinen, Junttila & Niemelä, LPAR-17 2010]: same as VSIDS scattering but lookahead-based branching heuristics in Minisat
- See also [Bordeaux, Hamadi & Samulowitz, IJCAI 2009]: Partition with parity constraints over randomly selected variables

Experiments with a Partitioning Function

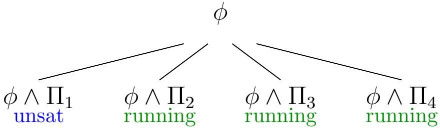
- Controlled experiments with LA DPLL partitioning function
- At most 300s spent in partitioning ϕ into 4 or 64 derived instances
- Run time of the derived instance set $\{\phi \wedge \Pi_1, \ldots, \phi \wedge \Pi_n\}$:
  - SAT: minimum run time of any satisfiable derived instance
  - UNSAT: maximum run time of (unsat) derived instances





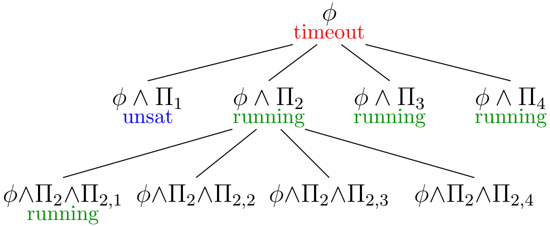
- Theory meets practise: UNSAT harder than SAT
- More results: [Hyvärinen, Junttila & Niemelä, LPAR-17 2010]

Iterative Partitioning



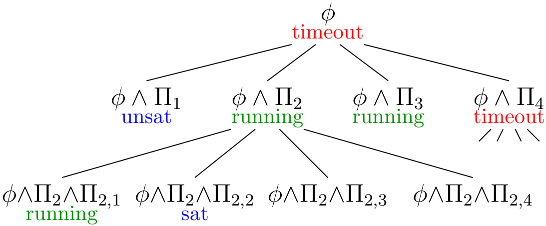
- Into how many derived instances should one partition ϕ?
- Many derived instances in a plain partitioning can be easy

Iterative Partitioning



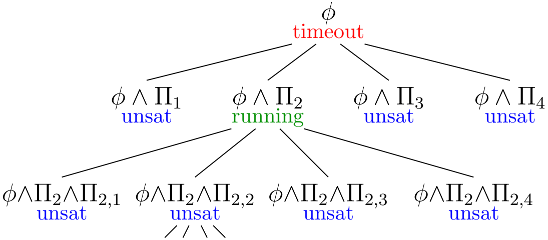
- Use smallish partitioning factor (e.g. 8), further partition hard derived instances, and use the free resources to solve these
- BFS/DFS construction of a "partitioning tree"
- Theoretical hazard of "void partitioning" an unsat instance is avoided when the *[derived] instance is also attempted to be solved*
- Fault tolerant

Iterative Partitioning



- Use smallish partitioning factor (e.g. 8), further partition hard derived instances, and use the free resources to solve these
- BFS/DFS construction of a "partitioning tree"
- Theoretical hazard of "void partitioning" an unsat instance is avoided when the *[derived] instance is also attempted to be solved*
- Fault tolerant

Iterative Partitioning



- Use smallish partitioning factor (e.g. 8), further partition hard derived instances, and use the free resources to solve these
- BFS/DFS construction of a "partitioning tree"
- Theoretical hazard of "void partitioning" an unsat instance is avoided when the *[derived] instance is also attempted to be solved*
- Fault tolerant

Iterative Partitioning: Some Experimental Results

- LA DPLL, LA scatter, VSIDS scatter: iterative partitioning
  NorduGrid, at most 64 jobs, all delays included
  Solver at jobs: Minisat 1.14, 1GB mem limit, 60–90min time limit
- SD 64: best of 64 runs of sequential Minisat 1.14, different prng seeds, 1GB mem limit
- ManySAT 1.0 and Plingeling 276: 12 cores, 32GB mem limit

6 hour wall-clock time limit for all approaches
SAT-Comp 2009 applications category, 63 insts not solved in the comp.

Name	Type	LA DPLL	LA scatter	VSIDS scatter	SD 64	ManySAT	Plingeling
9dlx_vliw_at_b_iq8	UNSAT	—	—	—	—	—	**3256.41**
9dlx_vliw_at_b_iq9	UNSAT	—	—	—	—	—	**5164.00**
AProVE07-25	UNSAT	**8992.60**	9176.91	11347.42	—	—	—
dated-5-19-u	UNSAT	16557.82	20155.96	**4124.62**	—	—	4465.00
eq.atree.braun.12.unsat	UNSAT	3157.19	**2357.55**	3006.19	20797.60	15338.00	—
eq.atree.braun.13.unsat	UNSAT	**7117.39**	8504.50	8158.85	—	—	—
gss-24-s100	SAT	1977.19	3449.55	2271.24	**968.23**	13190.00	2929.92
gss-26-s100	SAT	10844.22	—	**6057.80**	—	—	18173.00
gss-32-s100	SAT	—	**16412.40**	—	—	—	—
gus-md5-14	UNSAT	**14779.03**	16264.37	16098.04	—	—	—
ndhf_xits_09_UNSAT	UNSAT	—	—	**14793.78**	—	—	—
rpoc_xits_09_UNSAT	UNSAT	—	—	**12388.32**	—	—	—
sortnet-8-ipc5-h19-sat	SAT	—	—	—	—	—	**2699.62**
total-10-17-u	UNSAT	4431.21	7198.23	5099.73	—	10216.00	**3672.00**

Iterative Partitioning: Some Experimental Results

Same setting and solvers

"medium hard" instances, application and crafted categories

Name	Type	LA DPLL	LA scatter	VSIDS scatter	SD 64	COMP	ManySAT	Plingeling
Solved in SAT-COMP 2009 with best time at least 1 hour								
9dlx_vliw_at_b_iq7	UNSAT	—	—	—	—	6836.20	7665.00	**1576.08**
AProVE07-01	UNSAT	1465.22	**1322.04**	2451.36	20230.30	6816.94	13219.00	21144.00
dated-5-13-u	UNSAT	3881.60	4745.52	4563.15	—	8005.27	15818.00	**2524.05**
gss-22-s100	SAT	**830.77**	1151.13	4246.25	2280.82	4326.83	—	1136.39
gss-27-s100	SAT	—	—	9156.71	—	**7132.69**	—	18013.00
gus-md5-11	UNSAT	**1190.28**	2077.99	2092.54	5057.39	4518.06	20184.00	—
maxor128	UNSAT	—	—	—	—	7131.52	—	**2227.07**
maxxor064	UNSAT	—	—	—	—	5162.75	**2837.28**	9346.00
minandmaxor128	UNSAT	—	—	—	—	5143.44	4228.00	**3737.00**
mod4block_3vars_7gates	UNSAT	**1740.17**	1755.47	2326.02	—	4109.89	—	5048.00
new-difficult-26-243-24-70	SAT	3260.86	8887.61	5087.98	3311.62	4440.72	13343.00	**0.17**
rbcl_xits_08_UNSAT	UNSAT	4557.86	**2390.50**	3695.97	—	3892.92	10136.00	4783.00
sgen1-unsat-109-100	UNSAT	**1363.14**	3000.48	4196.36	14675.60	4045.49	—	—
UR-20-10p1	SAT	4463.24	—	—	—	8766.23	8164.00	**3598.17**
UTI-20-10p1	SAT	—	7097.74	—	—	6289.06	**750.76**	892.84
Challenge instances for Minisat								
countbitsarray02_32	UNSAT	1746.29	3003.50	997.84	2504.93	834.519	969.67	**258.60**
simon-s02b-k2f-gr-rcs-w8	UNSAT	3816.20	3106.70	14756.10	—	6.40	153.59	**5.01**
vange-col-abb313GPIA-9-c	SAT	—	—	—	—	**445.09**	—	520.95
velev-pipe-uns-1.0-8	UNSAT	—	—	—	—	307.48	337.94	**202.54**
vmpc_34	SAT	12452.59	1350.17	1479.62	2796.19	**35.347**	490.71	4064.00

Analytic studies

- For analytic and experimental run-time distribution based analyses on
  - portfolios without clause sharing,
  - partitioning, and
  - combinations of these

  on instances that are
  - unsatisfiable,
  - satisfiable with many solutions, or
  - satisfiable with few solutions

  see [Hyvärinen, Junttila, Niemelä, AISC 2008], [Hyvärinen, Junttila, Niemelä, AI*IA 2009] or [Hyvärinen, Junttila, Niemelä, Fund.Inf.], and [Hyvärinen, Junttila, Niemelä, LPAR-17]

Conclusions

- For “medium hard” instances multi-core approach with portfolios with clause sharing very competitive
- Portfolios with clause sharing can also work in a Grid environment
- Iterative search space partitioning very promising for “very hard” instances
- Obtaining good partitioning functions is challenging, especially for unsatisfiable instances
  - How to efficiently parallelize a resolution proof?
- Experiments with very hard instances very time consuming
- Possible future challenges: parallel generation of
  1. unsatisfiability cores
  2. proofs of unsatisfiability
  3. interpolants

References

This presentation was based mostly on the following articles by Hyvärinen, Junttila, and Niemelä:

- A Distribution Method for Solving SAT in Grids, Proc. SAT 2006, LNCS 4121, pp. 430–435, 2006
- Strategies for Solving SAT in Grids by Randomized Search, Proc. AISC 2008, LNCS 5144, pp. 125–140, 2008
- Incorporating Learning in Grid-Based Randomized SAT Solving, Proc. AIMSA 2008, LNCS 5253, pp. 247–261, 2008
- Incorporating Clause Learning in Grid-Based Randomized SAT Solving, J. Satisfiability 6:223–244, 2009
- Partitioning Search Spaces of a Randomized Search Proc. AI*IA 2009, LNCS 5883, pp. 243–252, 2009.
- Partitioning Search Spaces of a Randomized Search Fundamenta Informatica, accepted for publication.
- Partitioning SAT Instances for Distributed Solving, Proc. LPAR-17, LNCS 6397, pp. 372–386, 2010.
- IJCAI submission

Some More References

- W. Chrabakh, R. Wolski, GridSAT: a system for solving satisfiability problems using a computational grid, Parallel Computing 32(9):660–687, 2006
- L. Bordeaux, Y. Hamadi, and H. Samulowitz, Experiments with Massively Parallel Constraint Solving, Proc. IJCAI 2009, pp. 443–448, 2009
- Y. Hamadi, S. Jabbour, and L. Sais, Control-based Clause Sharing in Parallel SAT Solving, Proc. IJCAI 2009, pp. 499–504, 2009
- Y. Hamadi, S. Jabbour, and L. Sais, ManySAT: a Parallel SAT Solver, J. Satisfiability, 6(2009) 245–262
- C.M. Wintersteiger, Y. Hamadi, and L. de Moura, A Concurrent Portfolio Approach to SMT Solving, Proc. CAV 2009, LNCS 5643, pp. 715–720, 2009.
- A. Biere, Lingeling, Plingeling, PicoSAT, and PrecoSAT at SAT Race 2010, Tech. Report 10/1, Johannes Kepler University, 2010

Some More References

- W. Blochinger, C. Sinz, and W. Küchlin, Parallel propositional satisfiability checking with distributed dynamic learning, Parallel Computing 29(2003), 969–994
- S. Schulz, and W. Blochinger, Cooperate and Compete! A Hybrid Solving Strategy for Task-Parallel SAT Solving on Peer-to-Peer Desktop Grid, Proc. WPSS 2010, pp. 314–323, 2010.
- M. Lewis, T. Schubert and B. Becker, Multithreaded SAT Solving, Proc. ASP-DAC 2007, pp. 926–931, 2007
- T. Schubert, M. Lewis, and B. Becker, PaMiraXT: Parallel SAT Solving with Threads and Message Passing, J.Satisfiability 9:203–222, 2009
- W. Blochinger, W. Westje, W. Küchlin, and S. Wedeniwski, ZetaSAT — Boolean SATisfiability Solving on Desktop Grids, Proc. IEEE CCGrid 2005, pp. 1079–1086, 2005