Improving Search Space Splitting for Parallel SAT Solving

Ruben Martins Vasco Manquinho Inês Lynce

INESC-ID/IST, Technical University of Lisbon, Portugal

28 October 2010

Outline

Preliminaries

- Propositional Satisfiability (SAT):
  - A literal l_i is either a Boolean variable x_i or $\overline{x}_i$;
  - A clause $\omega = \bigvee_i l_i$:

    e.g. $\omega_1 = (x_1 \vee \overline{x}_2); \omega_2 = (x_2 \vee x_3); \omega_3 = (\overline{x}_2 \vee \overline{x}_3)$.
  - CNF formula $\varphi = \bigwedge_j \omega_j$:

    e.g. $\varphi = (\omega_1 \wedge \omega_2 \wedge \omega_3)$.
  - SAT problem is to decide if φ is satisfiable:

    e.g. φ is satisfied when $x_1 = 1$, $x_2 = 1$ and $x_3 = 0$.

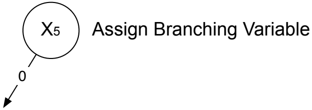
Preliminaries

- Since the mid 90s SAT solvers have shown remarkable improvements;
- Due to these improvements SAT solvers have been successfully applied to many practical applications:
  - Hardware and Software model checking;
  - Planning;
  - Cryptanalysis;
  - Computational Biology;
  - etc.

Sequential SAT Solving

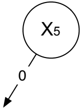
INPUT: CNF formula φ;

OUTPUT: SAT if an assignment is found; UNSAT otherwise.



Sequential SAT Solving

INPUT: CNF formula φ;
OUTPUT: SAT if an assignment is found; UNSAT otherwise.



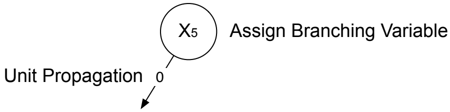
Assign Branching Variable

- VSIDS (Variable State Independent Decaying Sum) heuristic:
  - Each literal has an activity counter;
  - Each literal that occurs in a no-good has its activity increased;
  - At each call, the highest-value unassigned literal is chosen.

Sequential SAT Solving

INPUT: CNF formula φ;

OUTPUT: SAT if an assignment is found; UNSAT otherwise.

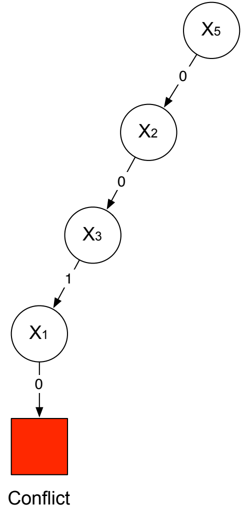


- Unit Clause Rule:
    - Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied.
- Unit Propagation:
    - Iterated application of the unit clause rule;
    - If an unsatisfied clause is identified it returns “Conflict”.

Sequential SAT Solving

INPUT: CNF formula φ;

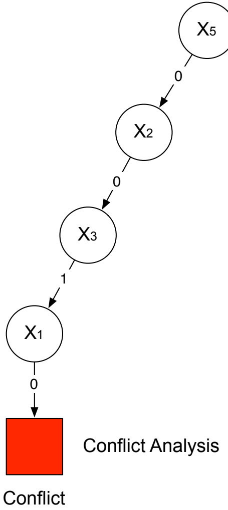
OUTPUT: SAT if an assignment is found; UNSAT otherwise.



Sequential SAT Solving

INPUT: CNF formula φ;

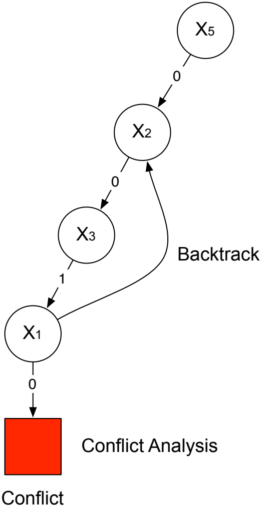
OUTPUT: SAT if an assignment is found; UNSAT otherwise.



Sequential SAT Solving

INPUT: CNF formula φ;

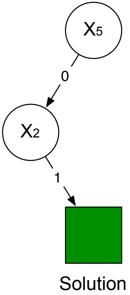
OUTPUT: SAT if an assignment is found; UNSAT otherwise.



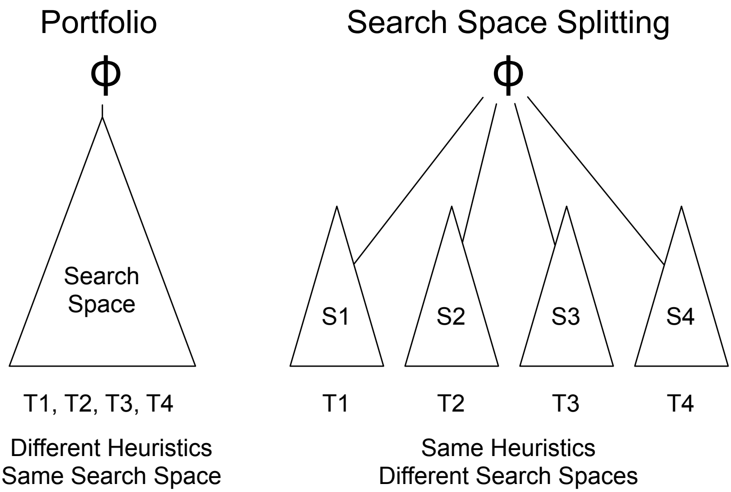
Sequential SAT Solving

INPUT: CNF formula φ;

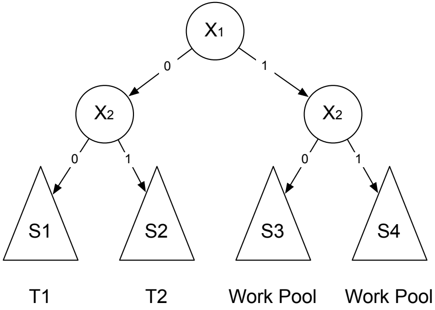
OUTPUT: SAT if an assignment is found; UNSAT otherwise.



Parallel SAT Solving: Portfolio vs Search Space Splitting

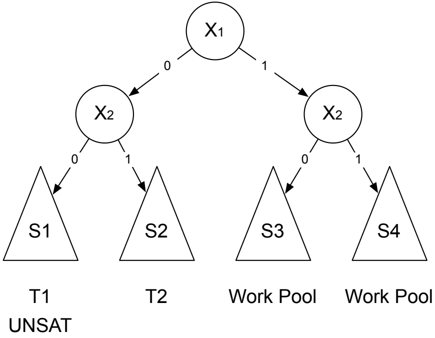


Search Space Splitting



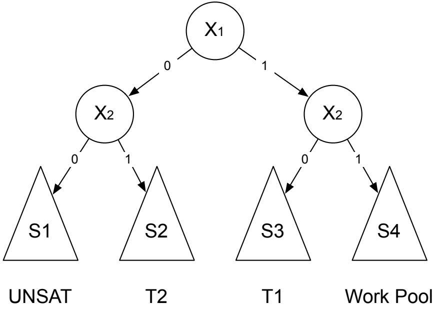
- The guiding paths describe the current state of the search process;
- The unused guiding paths are stored in the work queue;
- If a thread proves that its current subspace is unsatisfiable, it gets a new subspace from the work queue and continues searching;
- A dynamic work stealing procedure guarantees that work is available for all threads.

Search Space Splitting



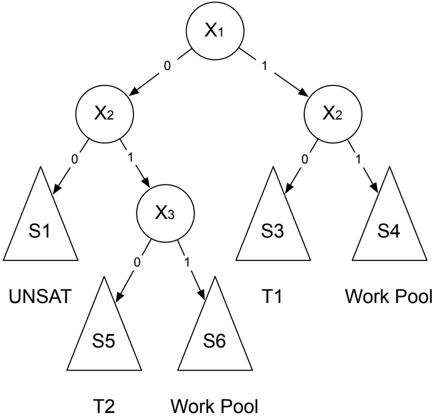
- The guiding paths describe the current state of the search process;
- The unused guiding paths are stored in the work queue;
- If a thread proves that its current subspace is unsatisfiable, it gets a new subspace from the work queue and continues searching;
- A dynamic work stealing procedure guarantees that work is available for all threads.

Search Space Splitting



- The guiding paths describe the current state of the search process;
- The unused guiding paths are stored in the work queue;
- If a thread proves that its current subspace is unsatisfiable, it gets a new subspace from the work queue and continues searching;
- A dynamic work stealing procedure guarantees that work is available for all threads.

Search Space Splitting



- The guiding paths describe the current state of the search process;
- The unused guiding paths are stored in the work queue;
- If a thread proves that its current subspace is unsatisfiable, it gets a new subspace from the work queue and continues searching;
- A dynamic work stealing procedure guarantees that work is available for all threads.

Overview: Portfolio vs Search Space Splitting

Portfolio:

- Different strategies for each SAT solver cooperate when solving the same search space:
  - Covers the space of search strategies;
  - Each solver has a complete view of the formula;
  - For a large number of cores it can be hard to find diverse viewpoints that provide orthogonal performance;
  - State-of-the-art multicore SAT solvers use this approach.

Overview: Portfolio vs Search Space Splitting

Search Space Splitting:

- Each SAT solver solves a different search subspace until no more subspaces exist or satisfiability has been proved:
  - Each solver has only a partial view of the formula;
  - It is not clear how to effectively choose the partition variables;
  - Load balancing requires an overhead on the dynamic work stealing procedure;

Improving Search Space Splitting

Choosing the Partition Variables:

1. Collecting VSIDS information:
    - Each thread runs a sequential SAT algorithm until *k* conflicts are reached;
    - After *k* conflicts the VSIDS heuristic of each thread can be analyzed in order to determine the partition variables;

- *Weak* Portfolio:
    - The VSIDS information is increased by running each thread with a different initial order of the VSIDS heuristic.

- Advantages of the *weak* portfolio:
    - Some instances can be solved during this stage;
    - More information is collected about the variables.

Improving Search Space Splitting

Choosing the Partition Variables:

2. Using the VSIDS information for choosing the partition variables:

VSIDS(T_1):	x_1	x_3	x_2	x_5	x_4
VSIDS(T_2):	x_3	x_2	x_1	x_5	x_4

Improving Search Space Splitting

Choosing the Partition Variables:

2. Using the VSIDS information for choosing the partition variables:

VSIDS(T_1):	x_1	x_3	x_2	x_5	x_4
Score:	1	2	3	4	5

VSIDS(T_2):	x_3	x_2	x_1	x_5	x_4
Score:	1	2	3	4	5

- For each variable is given a score from 1 to n according to its position;

Improving Search Space Splitting

Choosing the Partition Variables:

2. Using the VSIDS information for choosing the partition variables:

VSIDS(T_1):	x_1	x_3	x_2	x_5	x_4
Score:	1	2	3	4	5

VSIDS(T_2):	x_3	x_2	x_1	x_5	x_4
Score:	1	2	3	4	5

VSIDS($T_1 + T_2$):	x_3	x_1	x_2	x_5	x_4
Total Score:	3	4	5	8	10

- The final score of each variable is the sum of its score in each thread;
- The first n variables with lowest score are chosen as partition variables and are used to create the initial 2^n guiding paths.

Improving Search Space Splitting

Hybrid Heuristic:

1. Preventing load balance issues:
   a) If a thread t is searching for more than k conflicts;
   b) And at least half of the work pool has guiding paths that were created by t;

   - Then this means that the subspace of the thread t is dominating the search and can cause load balance issues.

2. Increase diversification of the search:
   a) If a thread is searching for more than z conflicts ($z >> k$);

   - Then this means that the search is going on for some time, and at this point a diversification of the search tends to lead to better results.

SAT4J//

- SAT4J is implemented in Java:
  - Even though SAT4J is not as efficient as other SAT solvers it is one of the most popular SAT solvers;
  - A parallel version of SAT4J can therefore be useful for many users.
- SAT4J// is a parallelization of SAT4J 2.1(sequential version):
  - Each thread maintains its own local clause database;
  - Clauses are not shared between threads.
- Although clause sharing increases the performance of a solver, without it we can have a better understanding of the impact of our heuristics.

SAT4J//

Four versions of SAT4J// were implemented:

- *no-Info*:
  - Search space splitting approach based on guiding paths;
  - Dynamic work stealing with a central queue of work which is topped up by the longer running thread;
  - The partition variables are chosen randomly.
- *Info*:
  - Uses a short initial stage of weak portfolio;
  - After this stage, the partition variables are chosen using the VSIDS information of all threads.

SAT4J//

Four versions of SAT4J// were implemented:

- *Pfolio*:
  - Uses a portfolio of SAT algorithms;
  - Each thread has a different combination of the following strategies: (1) restart, (2) polarity and (3) learning simplification.
- *Hybrid:*
  - Starts by using the search space splitting approach present in *Info*;
  - The hybrid heuristics switches from *Info* into *Pfolio*;
  - When switching to a portfolio mode:
    - All guiding paths are merged into a unique guiding path that has the literals that were common to all guiding paths;
    - The learnt clauses are kept from the search space splitting stage.

Experimental Results

- Benchmarks: 82 instances from the applications category of the SAT competition 2009 such that:
  - SAT4J 2.1 was able to solve in more than 180 seconds;
  - SAT4J 2.1 was unable to solve but MiniSAT 2.1 was able to solve within 1,200 seconds.
- The set of benchmarks is challenging for SAT4J and interesting for parallel testing;

Experimental Results

- Intel Core i7 CPU 930 (2.80 Ghz, 6GB) running Ubuntu 10.04 LTS;
- Timeout: 3,600 seconds (wall clock time);
- All versions of SAT4J// were run with 4 threads:
  - Each version of SAT4J// was run 3 times on each instance;
  - The runtimes shown in this section are the median of the successful runs for each instance;
  - An instance was considered solved if it could be solved in at least one run.

Experimental Results

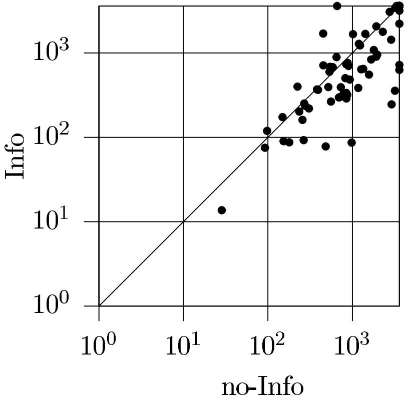
- Number of instances solved by each approach:

	# Inst	Seq	no-Info	Info	Pfolio	Hybrid
SAT	25	16	17	19	19	20
UNSAT	57	43	42	42	45	45
Total	82	59	59	61	64	65

- *Info* can solve more 2 instances than *no-Info*, showing the importance of the partition variables;
- *Hybrid* can solve more 1 instance than *Pfolio*, suggesting that a hybrid approach can outperform a pure portfolio approach.

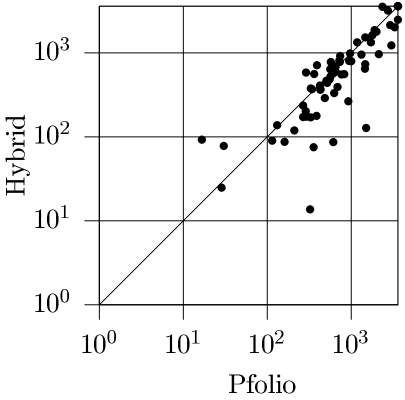
Experimental Results

- Runtimes for *no-Info* and *Info*:



Experimental Results

- Runtimes for *Pfolio* and *Hybrid*:



Conclusions

- Portfolio approaches currently dominate multicore SAT solvers;
- Experimental results show that:
  - Heuristically choosing the partition variables leads to clear improvements in search space splitting;
  - A hybrid approach between search space splitting and portfolio can lead to better results than a pure portfolio approach.
- This provides a strong stimulus for further exploration of hybrid solutions.
- As future work, we propose to:
  - Extend the use of the VSIDS heuristic of all threads to guide the search during runtime;
  - Improve the hybrid heuristic.