Parallel Search for Boolean Optimization

Ruben Martins Vasco Manquinho Inês Lynce

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

July 17, 2011

Motivation

- Multicore processors are now predominant;
- In the last years, several parallel SAT solvers have emerged;
- Parallel approaches boost the performance of sequential solvers;
- However, parallel approaches are scarce for Boolean optimization;
- Therefore, we propose new parallel algorithms for Boolean optimization.

Outline

Boolean Satisfiability

Boolean 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. $\omega_1 = (x_1); \omega_2 = (\overline{x}_1 \vee x_2 \vee x_3); \omega_3 = (\overline{x}_2 \vee \overline{x}_3)$.
- 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 $x_3 = 0$.

Boolean Optimization

Maximum Satisfiability (MaxSAT) Problem

Given a CNF formula φ, find an assignment to problem variables that ***maximizes the number of satisfied clauses*** in φ (or minimizes the number of unsatisfied clauses).

Partial MaxSAT Problem

Given a conjunction of two CNF formulas φ_h and φ_s, find an assignment to problem variables that ***satisfies all hard clauses*** (φ_h) and ***maximizes the number of satisfied soft clauses*** (φ_s).

Boolean Optimization

Pseudo-Boolean Optimization (PBO)

$$
\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{j=1}^{n} c_j \cdot x_j \\
\text{subject to} & \displaystyle\sum_{j=1}^{n} a_{ij} \cdot l_j \geq b_i, \\
& l_j \in \{x_j, \overline{x}_j\}, x_j \in \{0, 1\}, \\
& a_{ij}, b_i, c_j \in \mathbb{N}_0^+
\end{array}
$$

Encode MaxSAT to PBO

- For each ***hard*** clause $(l_1 \vee l_2 \vee \cdots \vee l_k)$
  - define a pseudo-Boolean constraint as $l_1 + l_2 + \cdots + l_k \geq 1$
- For each ***weighted soft clause*** (ω, c) where $\omega = (l_1 \vee l_2 \vee \cdots \vee l_k)$
  - define a PB constraint with a new relaxation variable r_i $r_i + l_1 + l_2 + \cdots + l_k \geq 1$
  - add $c \cdot r_i$ to the objective function

Encode MaxSAT to PBO

Weighted Partial MaxSAT instance

$$\begin{aligned} \varphi_h &= \{(x_1 \vee x_2 \vee x_3), (\overline{x}_2 \vee \overline{x}_3)\} \\ \varphi_s &= \{(\overline{x}_1, 2), (x_1 \vee \overline{x}_2, 3), (\overline{x}_1 \vee x_3, 2)\} \end{aligned}$$

Corresponding PBO instance

$$\begin{array}{rr} \text{minimize} & 2r_1 + 3r_2 + 2r_3 \\ \text{subject to} & x_1 + x_2 + x_3 \geq 1 \\ & \overline{x}_2 + \overline{x}_3 \geq 1 \\ & r_1 + \overline{x}_1 \geq 1 \\ & r_2 + x_1 + \overline{x}_2 \geq 1 \\ & r_3 + \overline{x}_1 + x_3 \geq 1 \end{array}$$

Algorithms for Boolean Optimization

Unsatisfiability-based algorithm for MaxSAT (lower bound value search):

1. Identify unsatisfiable sub-formula of an UNSAT formula:
   - SAT (PB) solver able to generate an UNSAT core.
2. For each unsatisfiable sub-formula φ_C:
   - Relax all (soft) clauses in φ_C by adding a new relaxation variable to each clause
   - Add a new constraint such that at most 1 relaxation variable is assigned value 1
3. When the resulting CNF formula is SAT, the solver terminates;
4. Otherwise, go back to 1.

Algorithms for Boolean Optimization

Linear search for PBO on the upper bound values of the objective function:

1. Search for a solution to the set of constraints;
2. Whenever a solution is found:
   - Update the upper bound value;
   - Add a PB constraint such that all solutions with a higher value of the objective function are discarded;
   - Go back to 1;
3. Otherwise, the resulting PBO formula is UNSAT and the solver terminates:
   - The optimum value is given by the last recorded solution.

Parallel Search (2 Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

T_0											T_1
LB											UB
0	1	2	3	4	5	6	7	8	9	10	11

Parallel Search (2 Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

	T_0										T_1
	LB										UB
0	1	2	3	4	5	6	7	8	9	10	11

T_0 returns **UNSAT**, a new lower bound has been found.

Parallel Search (2 Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

	T_0						T_1				
	LB						UB				
0	1	2	3	4	5	6	7	8	9	10	11

T_1 returns **SAT**, a new upper bound has been found.

Parallel Search (2 Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

		T_0					T_1				
		LB					UB				
0	1	2	3	4	5	6	7	8	9	10	11

T_0 returns **UNSAT**, a new lower bound has been found.

Parallel Search (2 Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

		T_0 ; T_1									
		LB ; UB									
0	1	**2**	3	4	5	6	7	8	9	10	11

T_1 returns **SAT**, a new upper bound has been found.

Parallel Search (2 Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

		T_0 ; T_1									
		LB ; UB									
0	1	**2**	3	4	5	6	7	8	9	10	11

LB value = UB value, hence the search terminates.

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

LB											UB
T_0			T_2				T_3				T_1
0	1	2	**3**	4	5	6	7	8	9	10	11

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

	LB										UB
	T_0		T_2				T_3				T_1
0	1	2	**3**	4	5	6	7	8	9	10	11

T_0 returns **UNSAT**, a new lower bound has been found.

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

	LB					UB					
	T_0		T_2			T_3					T_1
0	1	2	3	**4**	5	6	7	8	9	10	11

T_3 returns **SAT**, a new upper bound value has been found.

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

	LB					UB					
	T_0		T_2	T_3							T_1
0	1	2	**3**	4	5	6	7	8	9	10	11

T_3 gets a new local upper bound value.

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

	LB					UB					
	T_0		T_2	T_3		T_1					
0	1	2	**3**	4	5	6	7	8	9	10	11

T_1 updates its upper bound value.

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

			LB			UB					
	T_0		T_2	T_3		T_1					
0	1	2	**3**	4	5	6	7	8	9	10	11

T_2 returns **UNSAT**, a new lower bound value has been found.

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

			LB			UB					
	T_0			T_3	T_2	T_1					
0	1	2	**3**	4	5	6	7	8	9	10	11

T_2 gets a new local upper bound value.

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

			LB ; UB								
	T_0		T_1	T_3	T_2						
0	1	2	**3**	4	5	6	7	8	9	10	11

T_1 returns **SAT**, a new upper bound value has been found.

Parallel Search (n Threads)

- Parallel Search:
  - 1 thread searches on the LB (T_0);
  - 1 thread searches on the UB (T_1);
  - $(n-2)$ threads search on local UB ($T_2, \ldots, T_n$);
  - The optimum value is found when:
    - LB or UB thread terminates with a solution;
    - or when LB value = UB value.

			LB ; UB								
	T_0		T_1	T_3	T_2						
0	1	2	**3**	4	5	6	7	8	9	10	11

LB value = UB value, hence the search terminates.

Diversification of the Search

- Use two threads to search on the upper bound (T_1, T_2);
- Different strategies for each thread when updating the upper bound:
  - T_1 adds a PB constraint to limit the value of the objective function;
  - T_2 uses the ***sequential encoding*** to translate the PB constraint into clauses.
- The approaches are equivalent, but the search space is searched differently.

Clause Sharing

Soft and Hard Learned Clauses

If the conflict which gave origin into a new clause only involves hard clauses, then the learned clause is said to be a ***hard learned clause***. Otherwise, it is said to be a ***soft learned clause***.

Clause Sharing

Thread Bound Constraint

The PB constraint that is iteratively added to limit the value of the objective function is named ***thread bound constraint***.

Example

- Local UB value: 6
- Thread Bound Constraint: $\sum_{j=1}^{n} c_j \cdot x_j < 6$

Clause Sharing

Local Constraint

A thread bound constraint is a ***local constraint***. If a conflict which gave origin into a new clause involves a local constraint, then the learned clause is also a ***local constraint***.

Example

- T_2 local UB value: 3
  - Thread Bound Constraint: $\sum_{j=1}^{n} c_j \cdot x_j < 3$
  - T_2 learns a local constraint ω_2
- T_3 local UB value: 6
  - Thread Bound Constraint: $\sum_{j=1}^{n} c_j \cdot x_j < 6$
  - T_3 learns a local constraint ω_3
- ω_3 is always valid in T_2, however ω_2 may not be valid in T_3

Clause Sharing

Learned clauses created by the different algorithms

Learned Clause	Algorithms		
	LB	Local UB	UB
Soft	✓		
Hard	✓	✓	✓
Local		✓	
w/ encoding vars			✓

Clause Sharing

Learned clauses ***not shared*** between the different algorithms

- Soft Learned Clauses
- Learned Clauses with Encoding Variables

Learned clauses ***shared*** between the different algorithms

- Hard Learned Clauses
- Local Constraints
  - Shared only between UB algorithms and if: the upper bound of the importing thread is ***smaller or equal*** than the upper bound of the exporting thread.

Parallel Optimization Solvers

Solvers	# Threads		
	LB	local UB	UB
pwbo 2T	1	0	1
pwbo 4T	1	2	1
pwbo 4T-CNF	1	1	2

- Clause sharing is implemented on all solvers;
- Clauses that have 5 or less literals are shared:
  - this cutoff is dynamically changed during search (e.g. ManySAT);
  - clauses with literal block distance 2 are also shared (e.g. SArTagnan).
- Learned clauses are exported at each conflict;
- Shared clauses are imported at each restart.

Experimental Results

- Benchmarks: 497 partial MaxSAT instances from the industrial category of the MaxSAT Evaluation 2010;
- AMD Opteron 6172 processors (2.1 GHz with 64 GB of RAM) running Fedora Core 13;
- Timeout: 1,800 seconds (wall clock time);
- pwbo is a non-deterministic parallel solver:
  - Each version of pwbo was run 3 times on each instance;
  - The runtimes are the median of the runs;
  - An instance is solved if it can be solved in at least 2 runs.

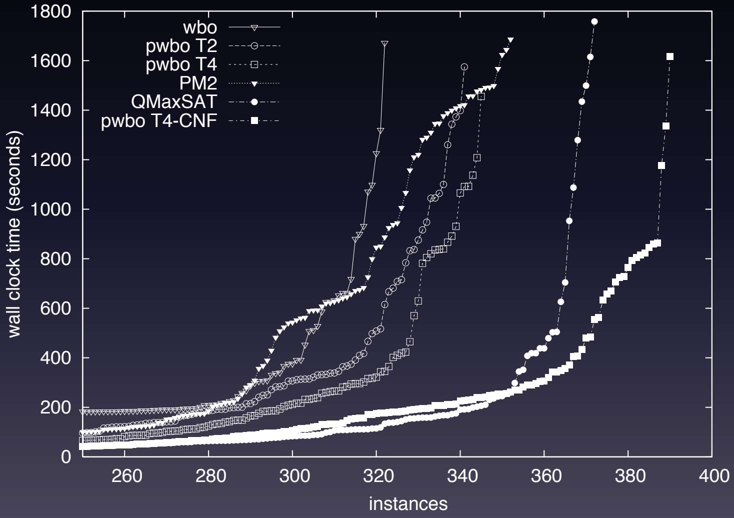
Experimental Results

Number of industrial partial MaxSAT instances solved by sequential and parallel solvers

#I	QMaxSAT	pm2	wbo	pwbo		
				2T	4T	4T-CNF
497	372	352	317	341	345	390

Experimental Results

Cactus plot with running times of solvers



Experimental Results

Speedup on the instances solved by wbo and all pwbo solvers

Solver	Time (s)	Speedup
wbo	36,208.33	1.00
pwbo 2T	22,798.28	1.59
pwbo 4T	18,203.79	1.99
pwbo 4T-CNF	13,236.87	2.74

Conclusions

- Parallel algorithms for Boolean optimization are scarce;
- New algorithms for parallel Boolean optimization have been proposed:
  - pwbo 2T: searches on the lower and upper bound values
    – searching in both directions increase the efficiency of the solver
  - pwbo 4T: also searches on local upper bound values
    – constant updates on the bound values reduce the search space
  - pwbo 4T-CNF: two threads search on the upper bound value
    – different strategies increase the diversification of the search
  - Clause sharing is implemented on all parallel solvers
    – clause sharing improve the performance of the solver

Research Directions

- Implement a portfolio of complementary algorithms:
  - Increase diversification of the lower and upper bound search;
  - Improve the effectiveness of the local upper bound search;
  - Change to a portfolio approach when the interval between the lower and upper bound becomes small.
- Study the scalability of our approach;
- On-the-fly clause sharing.