Parallel Search for Boolean Optimization

Ruben Martins, Vasco Manquinho, and Inês Lynce

IST/INESC-ID, Technical University of Lisbon, Portugal
{ruben,vmm,ines}@sat.inesc-id.pt

Abstract. The predominance of multicore processors has increased the interest in developing parallel Boolean Satisfiability (SAT) solvers. As a result, more parallel SAT solvers are emerging. Even though parallel approaches are known to boost performance, parallel approaches developed for Boolean optimization are scarce. This paper proposes parallel search algorithms for Boolean optimization and introduces a new parallel solver for Boolean optimization problem instances. Using two threads, an unsatisfiability-based algorithm is used to search on the lower bound value of the objective function, while at the same time a linear search is performed on the upper bound value of the objective function. Searching in both directions and exchanging learned clauses between these two orthogonal approaches makes the search more efficient. This idea is further extended for a larger number of threads by dividing the search space considering different local upper values of the objective function. The parallel search on different local upper values leads to constant updates on the lower and upper bound values, which result in reducing the search space. Moreover, different search strategies are performed on the upper bound value, increasing the diversification of the search.

1 Introduction

An increasing number of parallel Boolean Satisfiability (SAT) solvers have come to light in the recent past as a result of multicore processors having become the dominant platform. The use of SAT is widespread with many practical application and it is clear that the optimization version of SAT, i.e. Boolean optimization, can be applied to solve many practical optimization problems. The competitive performance and robustness of Boolean optimization solvers is certainly required to achieve this goal.

When compared with SAT instances, Boolean optimization instances tend to be more intricate as it is not sufficient to find an assignment that satisfies all the constraints, but rather an optimization function has to be taken into account. Hence, it comes as a natural step to develop parallel algorithms to Boolean optimization, following the recent success in the SAT field.

Although this reasoning comes as natural, there are only a few parallel implementation for solving Boolean Optimization. `SAT4J PB RES//CP` [1] implements a resolution based algorithm that competes with a cutting plane based algorithm to find a new upper

Proceedings of the 18th RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2011).
In conjunction with IJCAI 2011, Barcelona, Spain, July 17-18, 2011.

[1] `http://www.satcompetition.org/PoS/presentations-pos/leberre.pdf`

bound or to prove optimality. When one of the algorithms finds a new upper bound, it terminates the search of the other algorithm and both restart their search within the new upper bound. If one of the algorithms proves optimality then the problem is solved and the search is stopped. Clause sharing is not performed between these two algorithms. In the context of Integer Linear Programming (ILP), the commercial solver `CPLEX` is known to have the option of performing parallel search [2] but no detailed description is available.

Parallel algorithms have the advantage of allowing to implement orthogonal approaches that complement each other. That is the case in `SAT4J PB RES//CP` where cutting planes are run against resolution. Another alternative, which will be explored in this paper, is to run an algorithm that searches to increase the lower bound value against an algorithm that searches to decrease the upper bound value. Furthermore, one may have more than one algorithm searching on the upper bound value.

The main contribution of this paper is two-fold. First, we introduce a parallel search algorithm for Boolean optimization that uses two threads: one thread searches to reduce the upper bound value and the other thread searches to increase the lower bound value. Second, a more complex parallel algorithm is introduced, which extends the previous algorithm with additional threads searching to reduce the upper bound value.

The paper is organized as follows. The next section describes the preliminaries, namely Maximum Satisfiability (MaxSAT) and Pseudo-Boolean Optimization (PBO). Section 3 describes a parallel two-thread search algorithm for Boolean optimization, which is extended to a multithread algorithm in section 4. Afterwards, an experimental evaluation of the new algorithms is presented and the paper concludes.

2 Preliminaries

In this section we briefly describe the Boolean optimization formalisms to be used in the remainder of the paper, namely Maximum Satisfiability (MaxSAT) and Pseudo-Boolean Optimization (PBO). Moreover, we also review the encoding from MaxSAT to PBO.

The MaxSAT problem can be defined as finding an assignment to problem variables such that it minimizes (maximizes) the number of unsatisfied (satisfied) clauses in a CNF formula φ. However, MaxSAT has several variants such as partial MaxSAT, weighted MaxSAT and weighted partial MaxSAT. In the partial MaxSAT problem some clauses in φ are declared as hard, while the reminder are declared as soft. The objective in partial MaxSAT is to find an assignment such that all hard clauses are satisfied while minimizing the number of unsatisfied soft clauses. Finally, in the weighted versions of MaxSAT, soft clauses can have weights greater than 1 and the objective is to satisfy all hard clauses while minimizing the total weight of unsatisfied soft clauses.

A related Boolean optimization formalism is Pseudo-Boolean Optimization (PBO). PBO is defined as finding an assignment to problem variables such that all pseudo-Boolean constraints are satisfied and the value of a linear cost function is minimized. Unlike MaxSAT, constraints in PBO are more general and there are no soft constraints.

[2] `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`

Formally, one can define PBO as follows:

$$\begin{array}{ll} \text{minimize} & \sum\limits_{j\in N} c_j\, x_j \\ \text{subject to} & \sum\limits_{j\in N} a_{ij}\, l_j \geq b_i, \\ & l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0,1\}, a_{ij}, b_i, c_j \in \mathbf{N}_0^+. \end{array} \tag{1}$$

Notice that propositional clauses are a particular case of pseudo-Boolean constraints where all coefficients a_{ij} and the right-hand side b_i are equal to 1. Moreover, despite the differences one can easily encode MaxSAT instances into PBO. Consider the encoding of weighted partial MaxSAT to PBO. Since this is the most general variant of MaxSAT, the other MaxSAT variants trivially follow. The encoding can be as follows:

- Each hard clause in weighted partial MaxSAT formula φ is mapped directly as a pseudo-Boolean constraint.
- For each soft clause $\omega_i = \sum_{j=1}^{k} l_j$ in φ with positive weight c_i, create a new relaxation variable r_i. Next, add the term $c_i r_i$ to the objective function to minimize and add the constraint $r_i + \sum_{j=1}^{k} l_j \geq 1$ to the pseudo-Boolean formula.

Example 1. Consider the weighted partial MaxSAT instance where the formula is composed by a set of hard clauses φ_h and a set of weighted soft clauses φ_s such that:

$$\begin{array}{l} \varphi_h = \{\ (x_1 \vee x_2 \vee x_3), (\bar{x}_2 \vee \bar{x}_3)\}, \\ \varphi_s = \{\ (\bar{x}_1, 2), (x_1 \vee \bar{x}_2, 3), (\bar{x}_1 \vee x_3, 2)\}. \end{array} \tag{2}$$

According to the described encoding, the corresponding PBO instance would be:

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

Notice that some optimizations can be applied to the described encoding [18]. For instance, when encoding a soft clause with only one literal, the relaxation variable does not need to be generated. In our example, when encoding the weighted soft clause $(\bar{x}_1, 2)$, one can simply add the term $2x_1$ to the objective function instead of $2r_1$ and constraint $r_1 + \bar{x}_1 \geq 1$ does not need to be generated. When using such simplifications to the encoding, fewer relaxation variables are used, resulting in smaller PBO formulations. In practice this technique is very useful, since several industrial MaxSAT formulations have a large number of soft clauses with just one literal. As a result, for MaxSAT instances where all soft clauses have only one literal, no relaxation variables are introduced, and the set of variables in the PBO formulation is the same as in the original MaxSAT instance. In this section we do not review the encoding from PBO to MaxSAT, since it is not necessary to the reminder of the paper. However, these encodings have already been described in the literature [18].

3 Parallel Search on the Lower and Upper Bound Values

Unsatisfiability-based algorithms are very effective for several Boolean optimization problems [10, 18, 2]. These algorithms work by iteratively identifying unsatisfiable sub-formulas φ_U from the original formula φ. At each step, a SAT (or pseudo-Boolean) solver is used to check if the formula is unsatisfiable. If that is the case, for each soft constraint[3] in the identified unsatisfiable sub-formula φ_U, a new relaxation variable is added such that when assigned to 1, the soft constraint becomes satisfiable [18]. Moreover, additional constraints are also added to φ such that only one of the newly created relaxation variables can be assigned value 1. Next, the solver checks if the formula remains unsatisfiable. The procedure ends when the working formula becomes satisfiable and the solver returns a solution (i.e. the optimum value was found), or if φ_U only contains hard constraints (i.e. the original problem instance is unsatisfiable) [10].

The original procedure proposed by Fu and Malik [10] was improved, namely by using more effective encodings [21, 20] for the constraints on the relaxation variables, as well as different strategies to minimize the overall number of relaxation variables needed [20, 2]. Moreover, generalizations for the weighted MaxSAT variants have also been proposed [18, 3].

The most classical approach for Boolean optimization is the use of branch and bound algorithms where an upper bound on the value of the objective function is updated when a new solution is found. In these algorithms, lower bounds are estimated and whenever the lower bound is higher or equal to the upper bound, the search procedure can safely backtrack since extending the current set of variables assignments will surely not result in a better solution. Several MaxSAT and PBO algorithms follow this approach using different lower bounding procedures [15, 16, 4, 12, 17].

Another classical approach is to perform a linear search on the value of the objective function [8]. In this case, whenever a new solution is found, the upper bound value is updated and a new constraint is added such that all solutions with a higher value are excluded. Several PBO solvers use this approach [23, 9, 14, 1]. Moreover, by using an encoding to PBO, MaxSAT instances can also be solved using this approach [14].

Notice that the unsatisfiability-based procedures correspond to searching on the lower bound of the value of the optimal solution. At each iteration the working formula is unsatisfiable, and the algorithm terminates when the working formula becomes satisfiable. On the other hand, linear search on the values of the objective function corresponds to searching on the upper bound. In this case, the working formula is satisfiable at each iteration. The algorithm terminates when the problem instance becomes unsatisfiable and the optimum value is given by the last recorded solution.

An algorithm that searches on both the lower and upper bounds of the objective function has already been proposed [19]. The search is initially done by a pseudo-Boolean solver that performs a search on the upper bound value of the objective function. However, the use of the pseudo-Boolean solver is limited to 10% of the time limit given to solve the formula. If the PBO solver proves optimality within this time limit, the optimal solution has been found without having to search on the lower bound side.

[3] In MaxSAT a soft constraint is a clause, but for more general formulations, it can be any linear pseudo-Boolean constraint.

On the other hand, if the PBO solver was not able to prove optimality within the time limit, an unsatisfiability-based algorithm is used to search on the lower bound value of the objective function. `wbo` [18, 19] is a weighted Boolean optimization solver that uses this approach. Experimental results show that searching on the upper and lower bound values leads to solving more instances. Since these approaches are orthogonal, they complement each other on several classes of problem instances. In this paper, for simplicity of the algorithmic description, it is assumed that the Boolean optimization problem to be solved is weighted partial MaxSAT. However, algorithms described next can be easily generalized to other Boolean formulations.

3.1 Parallel Search

Nowadays, extra computing power is not coming anymore from higher processor frequencies but rather from a growing number of cores and processors. Exploiting this new architecture will allow Boolean Optimization solvers to become more effective and to be able to solve more problem instances. In this section we propose to perform a parallel search on the upper and lower bound values of the objective function. Even though searching on both the upper and the lower bound is not new [21, 19], searching on both of them in parallel is novel to the best of our knowledge. In this paper we propose the parallelization of the `wbo` solver, and the new solver is named `pwbo`. `pwbo` uses a linear search algorithm to search on the upper bound side and an unsatisfiability-based algorithm for searching on the lower bound side.

A parallel search with these two orthogonal strategies results in a performance as good as the best strategy for each problem instance. However, if both threads cooperate through clause sharing, it is possible to perform better than the best strategy. Additionally, both strategies can also cooperate in finding the optimum value. If during the search the lower bound value provided by the unsatisfiability-based algorithm and the upper bound value provided by the other thread become the same, it means that the optimum solution has been found. Therefore, it is not necessary for any of the threads to continue the search to prove optimality since their combined information already proves it.

3.2 Clause Sharing

It is commonly known that conflict-driven clause learning is crucial for the efficiency of modern Boolean optimization solvers. The description of conflict-driven clause learning procedures is out of the scope of the paper and will be assumed. We refer to the literature for detailed explanations on these procedures [22, 25]. In the context of parallel solving, it is expected that sharing learned clauses can help to further prune the search space and boost the performance of the parallel solver.

In parallel SAT solving, learned clauses that have less than a given number of literals are shared among the different threads. More advanced heuristics can be used for controlling the throughput and quality of the shared clauses [11]. Moreover, the literal block distance [5] can also be used for sharing clauses in a parallel context [13]. In our approach, we start by sharing clauses that have 5 or fewer literals. This cutoff is dynamically changed using the throughput and quality heuristic proposed by Hamadi et al. [11]. Additionally, all clauses that have literal block distance 2 are also shared.

It should be noted that in the `pwbo` solver not all conflict-driven learned clauses can be shared between both threads. This is due to the fact that the working formulas are different. On the unsatisfiability-based algorithm, the input formula φ_{MS} is a weighted partial MaxSAT formula with soft and hard constraints.

However, on the thread that makes the linear search on the upper bound value of the objective function, we encode the input formula φ_{MS} into a PBO formulation φ_{PBO}. As a result of that encoding (see example 1), the set of variables in φ_{PBO} might have been extended by additional relaxation variables necessary to encode the soft clauses in the original formula φ_{MS}. In order to define the conditions for safe clause sharing, we start by defining soft and hard learned clauses.

Definition 1 (Soft and Hard Learned Clauses). *If in the conflict analysis procedure used in the unsatisfiability-based algorithm, at least one soft clause is used in the clause learning process, then the generated learned clause is labeled as soft. On the other hand, if only hard clauses are used, then the generated learned clause is labeled as hard.*

Since φ_{MS} contains both soft and hard clauses, it will also have soft and hard learned clauses. On the other hand, φ_{PBO} only has hard clauses, and as a result, will only have hard learned clauses. Nevertheless, as mentioned previously, φ_{PBO} may contain additional variables not present in φ_{MS}. As a result, the safe sharing procedure between the two threads is as follows:

- A hard learned clause from the unsatisfiability-based algorithm can be safely shared to the other thread. This is due to the fact that the resolution operations used in φ_{MS} can also be reproduced in φ_{PBO}, since all original hard clauses in φ_{MS} are also present in φ_{PBO}.
- A soft learned clause from the unsatisfiability-based algorithm is not shared since it may not be valid for formula φ_{PBO}.
- A hard learned clause generated when solving φ_{PBO} can be shared with the unsatisfiability-based algorithm if the learned clause does not contain relaxation variables. This is safe since one can reproduce the generation of the hard learned clause by resolution steps using just hard clauses also present in φ_{MS}.

Finally, between iterations of the unsatisfiability-based algorithm, working formula φ_{MS} is also extended with additional relaxation variables. However, since these variables are added to soft clauses, if a conflict-based learned clause contains any relaxation variable, then it will necessarily be considered a soft clause. This is due to the fact that at least one soft clause would have been used in the learning procedure.

4 Parallel Search on the Upper Bound Value

The previous section presented a parallel search solver for Boolean optimization based on two orthogonal strategies. In the proposed approach, one thread is used for each strategy. For computer architectures with more than two cores, we can extend the previous idea by performing a parallel search on the upper bound value of the objective function. Therefore, if n cores are available, we can use one thread to search on the lower bound value of the objective function, while at the same time k threads search on different local upper bound values of the objective function and $n - k - 1$ threads search on the upper bound value of the objective function. Local bound threads have a local upper bound value that is enforced in their search. The iterative search on different local upper bound values leads to constant updates on the lower and upper bound values that will reduce the search space. Next, an example of this approach is described. Afterwards, a more detailed description of the algorithm is provided.

Example 2. Consider a weighted partial MaxSAT formula φ_{MS} as input. For the input formula, one can easily find initial lower and upper bounds. Suppose the initial lower and upper bound values are 0 and 11, respectively. Moreover, consider also that the optimal solution is 3 and our goal is to find it using four threads, t_0, t_1, t_2 and t_3. Thread t_0 applies an unsatisfiability-based algorithm (i.e., searches on the lower bound of the optimum value of the objective). This thread starts with a lower bound of 0 and will iteratively increase the lower bound until the optimum value is found.

Thread t_1 searches on the upper bound value of the objective function, while threads t_2 and t_3 search on different local upper bound values of the objective function. The initial input formula φ_{MS} is encoded into the pseudo-Boolean formalism (see section 2) and an additional constraint is added to limit the value of the objective function in each thread. For example, thread t_1 starts its search with upper bound value of 11 and threads t_2 and t_3 can start their search with respective local upper bound values of 3 and 7.

Suppose that thread t_2 finishes its computation and finds that the formula is unsatisfiable for an upper bound of 3. This means that there is no solution with values 0, 1 and 2 for the objective function. Therefore, the global lower bound value can be updated to 3. Thread t_2 is now free to search on a different local upper bound value, for example 5. In the meantime, thread t_3 found a solution with objective value 6. Hence, the global upper bound value can be updated to 6. Thread t_1 updates its upper bound value to 6 and thread t_3 is now free to search on a different local upper bound value, for example 4. Afterwards, consider that thread t_1 found a solution with objective value 3. Again, the global upper bound value can be updated to 3. Since the global lower bound value is the same as the global upper bound value, the optimum has been found and the search terminates.

4.1 Algorithmic Description

In what follows it is shown how the parallel search on the values of the objective function can be implemented in `pwbo`. Algorithm 1 describes `pwbo`. It receives a weighted partial MaxSAT formula (φ_{MS}) and the number of available threads (n). The thread with index 0 is referred to as the lower bound thread and applies an unsatisfiability-based algorithm to φ_{MS}. The thread with index 1 is referred to as the upper bound thread and searches on the upper bound value of the objective function. The threads indexed 2 to $n - 1$ are referred as local upper bound threads and search on different local upper bound values of the objective function. For the sake of simplicity, it is considered that there is only one thread that searches on the upper bound value of the objective function. However, this algorithm can be easily generalized for k local upper

Algorithm 1 PWBO Algorithm

```
INITSHAREDDATA($\varphi_{MS}, n$)
1  $globalLB \leftarrow 0$
2  $globalUB \leftarrow 1$
3  for each soft clause $(\omega, c) \in \varphi_{MS}$
4      do $globalUB \leftarrow globalUB + c$
5  $threadUB[1] \leftarrow globalUB$
6  $localThread[1] \leftarrow$ false
7  for $(t = 2; t < n; t$++)
8      do $threadUB[t] \leftarrow \lfloor \frac{(t-1) \times globalUB}{n-1} \rfloor$
9         $localThread[t] \leftarrow$ true
10 $search \leftarrow$ true
11 $globalModel \leftarrow \emptyset$

PWBO($\varphi_{MS}, n$)
1  INITSHAREDDATA($\varphi_{MS}, n$)
2  $\varphi_{PBO} \leftarrow$ ENCODEPBO($\varphi_{MS}$)
3  PARALLELLOWERALG($\varphi_{MS}$)
4  for $(t = 1; t < n; t$++)
5      do PARALLELUPPERALG($\varphi_{PBO}, t$)
6  ▷ wait until all threads terminate
7  return $globalModel$
```

bound threads and $n - k - 1$ threads that search on the global upper bound value of the objective function.

Algorithm 1 starts by initializing all data structures shared among all threads. First, the global lower bound is set to 0 since the weights of each soft clause are non-negative, and the global upper bound is set to the sum of the weights of all soft clauses plus 1. Next, each local upper thread is bounded by a local upper bound value in order to reduce its search space. The array $threadUB$ stores the upper bound values that limit the search for each thread. This array is continuously updated during the search and will always contain the upper bound limit for each thread.

Additionally, variable *search* is set to true. This Boolean flag is used to control the parallel search. When one thread finds the optimum value, it stops the search of the remaining threads by setting *search* to false. Finally, the threads are launched. The first thread (t_0) uses an unsatisfiability-based algorithm, while the remaining threads ($t_1, \ldots, t_{n-1}$) perform a search on different upper bound values of the objective function. The procedures PARALLELLOWERALG and PARALLELUPPERALG are further described in algorithm 2. When the optimum value is found, the search terminates and returns the model for the optimal solution.

Algorithm 2 show the behavior of the lower and upper bound threads. The PARALLELLOWERALG procedure describes the search using an unsatisfiability-based algorithm. First, the value of the lower bound is initialized to 0. At each iteration, a PB solver is used (line 2) and its output is a tuple (*st,* φ_U*, model*), where *st* denotes the resulting status of the solver (satisfiable, unsatisfiable or forced_abort), φ_U contains the

Algorithm 2 Parallel Algorithms for Boolean Optimization

```
ParallelLowerAlg($\varphi$)
 1  $localLB \leftarrow 0$
 2  while ($search$)
 3      do (st, $\varphi_U$, $model$) $\leftarrow$ PBSolver($\varphi$)
 4         if st = UNSAT
 5           then $localLB \leftarrow localLB +$ coreWeight($\varphi_U$)
 6                relaxCore($\varphi, \varphi_U$)
 7                updateLowerBound($localLB$, 0)
 8                if $globalUB = globalLB$
 9                  then $search \leftarrow$ false
10           else if st = SAT
11                  then updateUpperBound($localLB$, 0)
12                       $globalModel \leftarrow model$
13                       $search \leftarrow$ false

ParallelUpperAlg($\varphi, id$)
 1  while ($search$)
 2      do (st, $\varphi_U$, $model$) $\leftarrow$ PBSolver($\varphi \cup \{\sum c_j l_j \leq threadUB[id] - 1\}$)
 3         if st = UNSAT
 4           then updateLowerBound($threadUB[id]$, $id$)
 5                clearLocalConstraints($\varphi$)
 6           else if st = FORCED_ABORT
 7                  then clearLocalConstraints($\varphi$)
 8                  else if st = SAT
 9                         then updateUpperBound(value(model), $id$)
10                              $globalModel \leftarrow model$
11         if $globalUB = globalLB$
12           then $search \leftarrow$ false
```

unsatisfiable sub-formula provided by the PB solver if φ is unsatisfiable, and *model* contains an assignment to the variables of φ when the formula is satisfiable. In this thread, if the outcome of the PB solver is forced_abort, it means an optimal solution has been found by another thread (*search* was set to false) and the procedure terminates.

When the status of the PB solver is unsatisfiable (line 3), the unsatisfiable sub-formula φ_U is relaxed in the procedure relaxCore. We refer to the literature for the details of this procedure [2, 18]. Next, if *localLB* is greater than the current global lower bound, the global lower bound is updated in updateLowerBound (line 7). Notice that this may result in forcing one or more upper bound threads to abort and updating their upper bound limits. Otherwise, it means that an upper thread has already proved a better lower bound, and the search proceeds. If the status of the PB solver is satisfiable (line 10) it means that the unsatisfiability-based algorithm has found an optimal solution. As a result, the upper bound is updated (line 11), the solution is stored (line 12) and the flag *search* is set to false so that the remaining threads terminate.

The PARALLELUPPERALG procedure takes as input a PBO formula φ (section 2) and a thread identifier. At each iteration, a PB solver is used to solve φ (line 2), with an additional constraint that limits the value of the objective function. Let this constraint be named the *thread bound constraint*.

Notice that the thread bound constraint cannot be shared among all threads, since it is only valid if the optimum value is lower than the thread upper bound. The same sharing rules must apply to conflict-driven learned clauses that depend on the thread bound constraint. Therefore, it is necessary to define what is a local constraint and in what conditions it can be shared with other threads.

Definition 2 (Local Constraint). *The thread bound constraint is labeled a local constraint. Let ω be a conflict-driven learned clause and let φ_ω be the set of constraints used in the implication graph to learn ω. The new clause ω is defined as a local constraint if at least one constraint in φ_ω is a local constraint.*

After the call to the PB solver (line 3), if it returns *unsatisfiable*, it means that a new lower bound has been found. The lower bound is updated (line 4) and if the thread is searching on a local upper bound then it gets a new local upper bound value. Since the formula given to the PB solver was unsatisfiable, it is necessary to remove the thread bound constraint (line 5). Additionally, all local clauses are also removed since they may not be valid with the new local upper bound.

If the status of the solver is *forced_abort*, it means that some other thread already proved that the current search space is redundant. This can happen if the thread local upper bound is smaller than the global lower bound, or if the thread local upper bound is greater than the global upper bound. Local constraints are therefore removed (line 7). In fact, the local constraints are only removed when the forced abort is caused by an update on the global lower bound value. Otherwise, local constraints remain valid. If the PB solver returns *satisfiable*, a new upper bound has been found. Therefore, the global upper bound is updated (line 9) and the model is stored (line 10). If the thread is searching on a local upper bound then it gets a new local upper bound value, since the upper bound thread will continue the search on the new upper bound that has been found. If the thread is searching on the global upper bound the search then proceeds as usual. Finally, after the necessary updates depending on the PB solver status, it is checked wether the global upper bound is equal to the global lower bound. If this occurs, optimality is proved and the search terminates (lines 11-12).

We should note that some details are not fully described in this algorithmic description due to lack of space. In particular, updates to global data structures are inside critical regions and locks are used to avoid two or more threads to be updating these data structures at the same time. Moreover, updates to global lower and upper bounds only take place when the new values improve the current ones. Additionally, the update on the saved model is also inside a critical region and is only done when the global upper bound is updated.

Finally, when the global bounds are updated at UPDATELOWERBOUND and UPDATEUPPERBOUND, that may result in forcing the PB solver in other threads to stop (resulting in a *forced_abort* status). As a result, new thread local upper bounds must be defined for the aborted threads. Hence, each aborted thread is assigned a new local upper bound that covers the broadest range of yet untested bounds. More formally, the new local upper bounds are chosen as follows. Let $B =< b_0, b_1 \dots b_{k-1}, b_k >$ be a sorted list where b_0 equals the global lower bound and b_k equals the global upper bound, while the remaining b_i are the non-aborted thread upper bounds. Let $[b_{m-1}, b_m]$, where $1 \leq m \leq k$, define an interval such that for all $1 \leq i \leq k$ we have $b_m - b_{m-1} \geq b_i - b_{i-1}$. In this case, the new upper bound of the aborted thread is $\lfloor (b_m + b_{m-1})/2 \rfloor$. The sorted list B is updated with the new value and this process is repeated for each aborted thread.

4.2 Diversification of the Search

Until this point, all upper bound threads are searching using the same algorithm. To increase diversification of the search we propose to use two threads to search on the global upper bound value. When updating the upper bound, a PB constraint is added to limit the value of the objective function. However, instead of simply adding a PB constraint, we can encode the PB constraint into clauses and add these clauses to the PB solver. Even though the approaches are equivalent, the search space will be searched differently. Similar approaches of encoding PB constraints to CNF have been successfully used in the past [9, 6, 7].

4.3 Clause Sharing

In this section the clause sharing procedure between threads is presented. We start by describing the clause sharing between the thread searching on the lower bound and the other threads searching on the upper bound. First, rules for sharing clauses described in section 3 can be safely applied. The difference is that local constraints cannot be safely shared with the lower bound thread. Hence, only clauses that are not local and do not have relaxation variables, are shared with the lower bound thread.

Although local constraints are not shared with the lower bound algorithm, these constraints can be shared between upper bound threads. However, sharing local constraints depends on the thread upper bound. If an importing thread has an upper bound smaller or equal to the upper bound of the exporting thread, then the import is safe. Otherwise, the import may be unsafe and the sharing is not done. Note that, when using diversification of the search, the clauses that translate the PB constraint cannot be shared since they have auxiliary variables that do not exist in the remaining upper bound threads.

5 Experimental Results

The parallel algorithms for Boolean optimization were implemented on the top of `wbo` and evaluated against state-of-the-art solvers for Boolean Optimization. The solvers were run on all the partial MaxSAT instances from the industrial category of the MaxSAT Evaluation 2010 [4], which correspond to a set of 497 instances. Results for other categories could be presented, but the number of industrial instances on the remaining categories at the MaxSAT evaluation 2010 is low and `wbo` was already the best solver for those categories. The evaluation was performed on two AMD Opteron 6172 processors

[4] `http://www.maxsat.udl.cat/10/`

Table 1. Number of industrial partial MaxSAT instances solved by sequential and parallel solvers

Benchmark set	#I	QMaxSAT	pm2	wbo	pwbo		
					2T	4T	4T-CNF
bcp-fir	59	50	58	42	44	44	56
bcp-hipp-yRa1	55	46	45	22	22	24	40
bcp-msp	64	26	14	16	15	15	20
bcp-mtg	40	40	40	31	32	33	40
bcp-syn	74	32	39	34	36	36	40
CircuitTraceCompaction	4	4	4	4	4	4	4
HaplotypeAssembly	6	0	5	5	5	5	5
pbo-mqc	168	153	129	147	167	168	168
pbo-routing	15	15	15	15	15	15	15
PROTEIN_INS	12	6	3	1	1	1	2
Total	497	372	352	317	341	345	390

(2.1 GHz with 64 GB of RAM) running Fedora Core 13 with a timeout of 1,800 seconds (wall clock time).

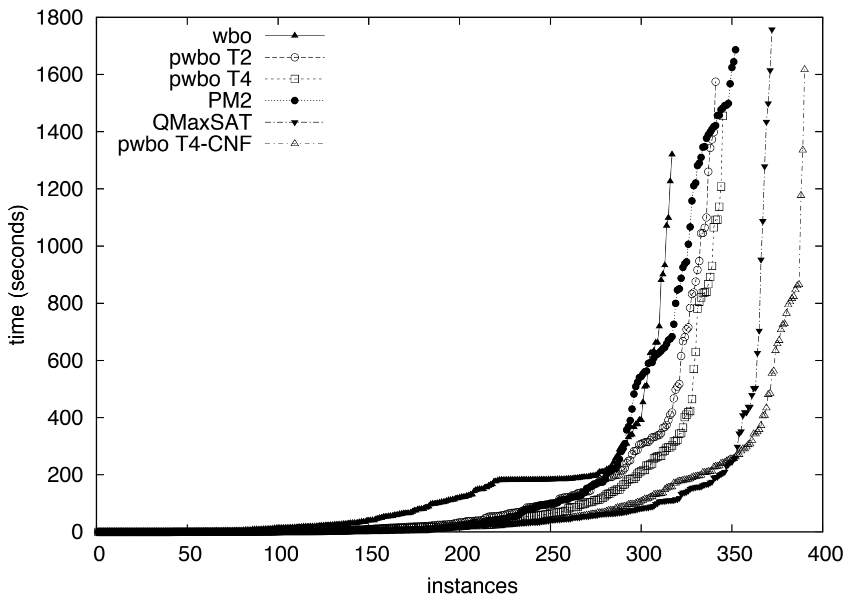
The results were obtained by running each parallel solver on each instance for three times. Similarly to what is done when analyzing randomized solvers, the median time was taken into account. This means that an instance must be solved by at least two of the three runs to be considered solved. We should note, however, that this measure is more conservative than the one used in the SAT Race 2008 [5] which is commonly used by parallel SAT solvers [11].

Table 1 gives the number of partial MaxSAT instances from the industrial category that were solved by sequential and parallel solvers. The sequential solvers considered were `QMaxSAT` [6] (ranked 1st in the MaxSAT Evaluation 2010), `pm2` [2] (ranked 2nd) and `wbo` [18, 19] (ranked 3rd). Note that `wbo` is also our reference solver as the new parallel algorithms were implemented on the top of `wbo`. `SAT4J MAXSAT` [14] and `SAT4J MAXSAT RES//CP` were not evaluated since their performance is not comparable to the remaining state-of-the-art partial MaxSAT solvers. For the 497 instances tested, `SAT4J MAXSAT 2.2.3` and `SAT4J MAXSAT RES//CP` can only solve 277 and 290 instances, respectively.

The parallel solvers evaluated correspond to the different versions of `pwbo`. `pwbo` is a parallel solver implemented on the top of `wbo`. `pwbo 2T` uses two threads according to what is described in section 3, thus having one thread searching on the lower bound value and another thread searching on the upper bound value. `pwbo 4T` and `pwbo 4T-CNF` use four threads according to what is described in section 4, thus having one thread searching on the lower bound value and three threads searching on the upper bound value. The difference between `pwbo 4T` and `pwbo 4T-CNF` is on the number of threads that search on local and global upper bound values. Increasing the number of threads that search on local upper bound values allows to reduce the search space by finding new lower and upper bounds. On the other hand, increasing the number of

[5] `http://baldur.iti.uka.de/sat-race-2008/`
[6] `http://www.maxsat.udl.cat/10/solvers/QMaxSat.pdf`

Fig. 1. Cactus plot with running times of solvers



threads that search on the global upper bound increases the diversification of the search, since those threads are searching using different strategies. `pwbo 4T` uses two threads to search on local upper bound values and one thread to search on the global upper bound value. On the other hand, `pwbo 4T-CNF` uses one thread to search on local upper bound values and two threads to search on the global upper bound value with the different strategies described in section 4.2. The objective function for partial MaxSAT instances corresponds to a cardinality constraint, since all coefficients are 1. Therefore, `pwbo 4T-CNF` uses Sinz's encoding [24] to translate the cardinality constraint into clauses.

Clearly, all versions of `pwbo` perform better than the sequential solver `wbo`. When analyzing each benchmark family, one can conclude that the benefits obtained from parallel solvers are not the same for all benchmarks families, although in general the number of solved instances tends to increase for all families. There is a significant boost when using two threads (`pwbo T2`), showing that a parallel search on the lower and upper bounds makes the search mode efficient and solves more instances. When using four threads the number of solved instances still increases. `pwbo T4` shows that reducing the search space by doing a local upper bound search allows solving more instances. Another significant boost is given by the diversification of the search. Indeed, `pwbo T4-CNF` with its combination of search diversification and search space reduction is able to solve more instances than the best sequential solver (`QMaxSAT`), thus improving the current state of the art.

Figure 1 contains a cactus plot with the running times of all the solvers for which data was given in Table 1. With no doubt, the parallel versions of `pwbo` perform better than `wbo`. Moreover, the best performing solver is `pwbo T4-CNF` that clearly outperforms all other solvers, including the best sequential solver `QMaxSAT`. Finally, Table 2

Table 2. Speedup on the 312 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

contains the speedup resulting from using `pwbo`, the parallel version of `wbo`. `wbo` is compared against `pwbo 2T`, `pwbo 4T` and `pwbo 4T-CNF`. The results are conclusive. The speedup increases as the number of threads increases, being almost 2 in `pwbo 4T` when local upper bound search is used and close to 3 in `pwbo 4T-CNF` when diversification of the search is combined with reduction of the search space.

6 Conclusions

This paper introduces new parallel algorithms for Boolean optimization. This work was in part motivated by the recent success of parallel SAT algorithms, also taking into account that parallel algorithms for Boolean optimization are scarce. Two new algorithms were proposed. The first algorithm uses two threads, one searching on the lower bound value and the other one searching on the upper bound value of the objective function. The second algorithm uses an additional number of threads to search on local upper bound values. Moreover, this algorithm is further improved by increasing the diversification of the search through different search strategies on the global upper bound. Experimental results, obtained on a significant number of problem instances, clearly show the efficiency of the new proposed algorithms.

Due to the success of our approach in partial MaxSAT, we plan to further extend our evaluation to weighted Boolean optimization, as future work. Moreover, we propose to further increase the diversification of the search by implementing a portfolio of complementary algorithms. The portfolio of algorithms can then be used to search on local and global upper bounds thus increasing the efficiency of the solver. Finally, an experimental study of the scalability of our approach should also be performed.

Acknowledgement. This work was partially supported by FCT under research projects BSOLO (PTDC/EIA/76572/2006) and iExplain (PTDC/EIA-CCO/102077/2008), and INESC-ID multiannual funding through the PIDDAC program funds.

References

1. F. Aloul, A. Ramani, I. Markov, and K. A. Sakallah. Generic ILP versus specialized 0-1 ILP: An update. In *International Conference on Computer-Aided Design*, pages 450–457, 2002.
2. C. Ansótegui, M. Bonet, and J. Levy. Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 427–440, 2009.
3. C. Ansótegui, M. Bonet, and J. Levy. A New Algorithm for Weighted Partial MaxSAT. In *AAAI Conference on Artificial Intelligence*, pages 3–8, 2010.

4. J. Argelich, C. M. Li, and F. Manyà. An improved exact solver for partial max-sat. In *International Conference on Nonconvex Programming: Local and Global Approaches*, pages 230–231, 2007.
5. G. Audemard and L. Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *International Joint Conference on Artificial Intelligence*, pages 399–404, 2009.
6. O. Bailleux, Y. Boufkhad, and O. Roussel. A Translation of Pseudo Boolean Constraints to SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–200, 2006.
7. O. Bailleux, Y. Boufkhad, and O. Roussel. New Encodings of Pseudo-Boolean Constraints into CNF. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 181–194, 2009.
8. P. Barth. A Davis-Putnam Enumeration Algorithm for Linear Pseudo-Boolean Optimization. Technical Report MPI-I-95-2-003, Max Plank Institute for Computer Science, 1995.
9. N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
10. Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 252–265, 2006.
11. Y. Hamadi, S. Jabbour, and L. Sais. Control-Based Clause Sharing in Parallel SAT Solving. In *International Joint Conference on Artificial Intelligence*, pages 499–504, 2009.
12. F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSAT: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.
13. S. Kottler. SArTagnan. *SAT Race, Solver Description*, 2010.
14. D. Le Berre and A. Parrain. The sat4j library, release 2.2 system description. *Journal on Satisfiability Boolean Modeling and Computation*, 7:59–64, 2010.
15. C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.
16. H. Lin and K. Su. Exploiting inference rules to compute lower bounds for MAX-SAT solving. In *International Joint Conference on Artificial Intelligence*, pages 2334–2339, 2007.
17. V. Manquinho and J. Marques-Silva. Search pruning techniques in SAT-based branch-and-bound algorithms for the binate covering problem. *IEEE Transactions on Computer-Aided Design*, 21(5):505–516, 2002.
18. V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for Weighted Boolean Optimization. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 495–508, 2009.
19. V. Manquinho, R. Martins, and I. Lynce. Improving Unsatisfiability-Based Algorithms for Boolean Optimization. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 181–193, 2010.
20. J. Marques-Silva and V. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 225–230, 2008.
21. J. Marques-Silva and J. Planes. Algorithms for Maximum Satisfiability using Unsatisfiable Cores. In *Design, Automation and Testing in Europe Conference*, pages 408–413, 2008.
22. J. Marques-Silva and K. Sakallah. GRASP: A new search algorithm for satisfiability. In *International Conference on Computer-Aided Design*, pages 220–227, 1996.
23. H. Sheini and K. Sakallah. Pueblo: A Modern Pseudo-Boolean SAT Solver. In *Design, Automation and Testing in Europe Conference*, pages 684–685, March 2005.
24. C. Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 827–831, 2005.
25. L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *International Conference on Computer-Aided Design*, pages 279–285, 2001.