Clause Sharing in Parallel MaxSAT

Ruben Martins, Vasco Manquinho, and Inês Lynce

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

Abstract. In parallel MaxSAT solving, sharing learned clauses is expected to help to further prune the search space and boost the performance of a parallel solver. However, not all learned clauses should be shared since it could lead to an exponential blow up in memory and to sharing many irrelevant clauses. The main question is which learned clauses should be shared among the different threads. This paper reviews the existing heuristics for sharing learned clauses, namely, static and dynamic heuristics. Moreover, a new heuristic for clause sharing is presented based on *freezing* shared clauses. Shared clauses are only incorporated into the solver when they are expected to be useful in the near future. Experimental results show the importance of clause sharing and that the freezing heuristic outperforms other clause sharing heuristics.

1 Introduction

Nowadays multicore processors are becoming the dominant platform. As a result, parallel Maximum Satisfiability (MaxSAT) solvers have been recently presented to exploit this new architecture [10, 9]. These parallel solvers simultaneously search on the lower and upper bound values of the optimal solution. Searching in both directions and sharing learned clauses between these two orthogonal approaches makes the search more efficient. However, it is not clear which clauses should be shared among the different threads. The problem of determining if a shared clause will be useful in the future remains challenging, and in practice heuristics are used to select which learned clauses should be shared. This paper sheds some light on the impact of different clause sharing heuristics in parallel MaxSAT solving. The main contribution of this paper is twofold: (1) a new heuristic for clause sharing that freezes shared clauses until they are expected to be useful and (2) an empirical evaluation of static, dynamic and freezing heuristics for clause sharing.

The paper is organized as follows. In the next section the MaxSAT problem is defined and MaxSAT solvers are briefly referred. Section 3 describes different clause sharing heuristics that will be analyzed in the paper. Afterwards, section 4 presents an experimental evaluation of the different clause sharing heuristics. Finally, the paper concludes and suggests future work.

2 Preliminaries

A Boolean formula in conjunctive normal form (CNF) is a conjunction of clauses, where a clause is a disjunction of literals and a literal is a Boolean variable x or its

negation $\neg x$. A Boolean variable may be assigned truth values true or false. A positive (negative) literal x ($\neg x$) is said to be satisfied if the respective variable is assigned value true (false). A positive (negative) literal x ($\neg x$) is said to be unsatisfied if the respective variable is assigned value false (true). A variable (and respective literals) not assigned is said to be unassigned. A clause is said to be satisfied if at least one of its literals is satisfied. A clause is said to be unsatisfied if all of its literals are unsatisfied. A clause is said to be unit if all literals but one are unsatisfied and the remaining literal is unassigned. Otherwise, a clause is said to be unresolved. A formula is satisfied is all of its clauses are satisfied. The Boolean satisfiability (SAT) problem is to decide whether there exists an assignment that makes the formula satisfied. Such assignment is called a solution.

Maximum Satisfiability (MaxSAT) is an optimization version of Boolean Satisfiability (SAT) which consists in finding an assignment that minimizes (maximizes) the number of unsatisfied (satisfied) clauses. MaxSAT has several variants such as partial MaxSAT, weighted MaxSAT and weighted partial MaxSAT. In the partial MaxSAT problem, some clauses are declared as hard, while the rest are declared as soft. The objective in partial MaxSAT is to find an assignment to problem variables 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.

The parallel MaxSAT solver PWBO [9] used in this paper is based on having several threads running a portfolio of two orthogonal algorithms: (i) an unsatisfiability-based algorithm that searches on the lower bound of the optimal solution [8] and (ii) a classical linear search algorithm that searches on the upper bound [7]. Notice that PWBO is not limited to the best performing algorithm in the portfolio, since threads can cooperate by exchanging information on the lower and upper bounds found during the search, as well as exchanging learned clauses that can prune the search on the other threads. In this paper we focus on strategies for the sharing of learned clauses between threads that can be used for improving parallel MaxSAT solvers. It is assumed that the reader is familiar with algorithms for MaxSAT, and we refer to the literature [4, 7, 8, 1, 9] for details.

3 Clause Sharing Heuristics

Clause sharing heuristics can be divided into three categories: (1) static, (2) dynamic and (3) freezing. The static heuristics share clauses within a given cutoff, whereas the dynamic heuristics adjust this cutoff during the search. Alternatively, the freezing heuristics temporarily freeze shared clauses until they are expected to be useful.

3.1 Static

The static heuristics are the most used for clause sharing since they are simple but still efficient in practice. The following measures are used in these heuristics:

- *Size*: the clause size is given by the number of literals. Small clauses are expected to be more useful than larger clauses.

– *Literal Block Distance* (LBD) [3]: the literal block distance corresponds to the number of different decision levels involved in a clause. The decision level of a literal denotes the depth of the decision tree at which the corresponding variable was assigned a value. Clauses with small LBD are considered as more relevant.
– *Random*: randomly decide whether to share each learned clause with a given probability. This heuristic was designed to evaluate the other heuristics which are expected to be more effective than a random one.

3.2 Dynamic

The size of learned clauses tends to increase over time. Consequently, in parallel solving, any static limit may lead to halting the clause sharing process. Therefore, to continue sharing learned clauses it is necessary to dynamically increase the limit during search. Hamadi et al. [6] proposed the following dynamic heuristic. At every k conflicts (corresponding to a period α) the throughput of shared clauses is evaluated between each pair of threads $(t_i \rightarrow t_j)$ according to the following heuristic:

$$\text{limit}^{\alpha+1}_{t_i \rightarrow t_j} = \begin{cases} \text{limit}^{\alpha}_{t_i \rightarrow t_j} + \text{quality}^{\alpha}_{t_i \rightarrow t_j} \times \frac{b}{\text{limit}^{\alpha}_{t_i \rightarrow t_j}} & \text{if sharing is small} \\ \text{limit}^{\alpha}_{t_i \rightarrow t_j} - (1 - \text{quality}^{\alpha}_{t_i \rightarrow t_j}) \times a \times \text{limit}^{\alpha}_{t_i \rightarrow t_j} & \text{if sharing is large} \end{cases},$$

where *a* and *b* are positive constants and the value of $\text{quality}^{\alpha}_{t_i \rightarrow t_j}$ corresponds to the quality of shared clauses that were send from t_i to t_j.

A shared clause is said to have *quality* [6] if at least half of its literals are active. A literal is *active* if its VSIDS heuristic [11] score is high, i.e. it is likely to be chosen as a decision variable in the near future. Hence, $\text{quality}^{\alpha}_{t_i \rightarrow t_j}$ gives the ratio between quality shared clauses and the total number of shared clauses in the period α. If the quality is high then the increase (decrease) in the size limit of shared clauses will be larger (smaller). The reasoning behind this heuristic is that the information recently received from a thread t_i is qualitatively linked to the information which could be received from the same thread t_i in the near future. In our experimental setting, we have selected $a = 0.125, b = 8$ and $\alpha = 3000$ conflicts. The throughput at each period is set to 750, i.e. if a thread t_j receives less than 750 shared learned clauses in the period α, it increases the limit of the size of shared clauses. Otherwise, this limit is decreased. These parameters are similar to the ones used by Hamadi et al. [6].

3.3 Freezing

Shared learned clauses may not be useful when they are imported and can actually deviate the search from the correct path. Our motivation for the freezing heuristic is to only import shared clauses when they are expected to be useful in the near future. Figure 1 illustrates the freezing procedure. Each shared clause ω is evaluated to determine if it will be frozen or imported. If ω is frozen then it will be reevaluated later. However, if ω is assigned the frozen state more than k times it is permanently deleted. When evaluating ω, our goal is to import clauses that are unsatisfied or that will become unit clauses in the near future. Next, the freezing heuristic is presented. According to the *status* of ω (satisfied, unsatisfied, unit or unresolved), whether ω should be frozen is decided:

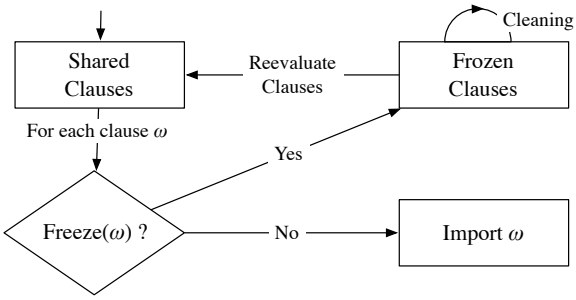

Fig. 1. Freezing procedure for sharing learned clauses

- ω is *satisfied*: Let *level* denote the current decision level, $level_h(\omega)$ the highest decision level of the satisfied literals in ω, $unassignedLits(\omega)$ the number of unassigned literals in ω and $activeLits(\omega)$ the number of active literals in ω. If $(level - level_h(\omega) \leq a)$ and $(unassignedLits(\omega) - activeLits(\omega) \leq b)$ then ω is imported, otherwise it is frozen. A satisfied clause is expected to be useful in the near future if it is not necessary to backtrack significantly to make the clause unit. It is also important that the number of unassigned literals is small or else the clause may not become unit in the near future. Active literals are also taken into consideration since they will be assigned in the near future.
- ω is *unsatisfied* or *unit*: ω is always imported;
- ω is *unresolved*: if $(\text{unassigned}(\omega) - \text{active}(\omega) \leq b)$ then the clause is imported. Otherwise, it is frozen. Similarly to the satisfied case, if the number of unassigned literals is small then ω is likely to be unit in the near future.

In our experimental setting, we have selected $a = 31, b = 5$ and $k = 7$. In addition, the frozen clauses are reevaluated every 300 conflicts. These parameters were experimentally tuned. Freezing learned clauses has been recently proposed in the context of deletion strategies for learned clauses in SAT solving [2]. However, to the best of our knowledge, freezing shared clauses in a parallel solving context is a novel approach.

4 Experimental Results and Discussion

All experiments were run on the partial MaxSAT instances from the industrial category of the MaxSAT Evaluation 2011 [1]. Instances that are easily solved will have similar solving times with and without sharing learned clauses. Hence, if an instance takes less than 60 seconds to be solved it is not considered. The evaluation was performed on two AMD Opteron 6172 processors (2.1 GHz with 64 GB of RAM) running Fedora Core 13 with a timeout of 1,800 seconds (wall clock time). The different clause sharing heuristics were implemented on top of the portfolio version of PWBO [9] and were run with 4 threads. To have a better understanding of the impact of each heuristic, we have built a deterministic version of PWBO that is based on exchanging only information

[1] `http://www.maxsat.udl.cat/11/`

Table 1. Comparison of the different heuristics for sharing learned clauses

	Heuristic	#Solved	Avg. #Clauses	Avg. Size	Time	Speedup
	No sharing	137	–	–	32,188.57	1.00
Static	Random 30	134	10,140.22	128.21	27,394.46	1.18
Static	LBD 5	137	8,947.36	9.94	25,346.69	1.27
Static	Size 8	137	7,529.18	5.30	25,098.85	1.28
Static	Size 32	138	18,027.48	11.76	25,174.29	1.28
	Dynamic	138	13,296.28	7.33	24,218.84	1.33
	Freezing	140	16,228.53	11.01	21,611.21	1.49

between the different threads at synchronization points (at every 100 conflicts). This is similar to what has been done in the deterministic version of MANYSAT [5].

Table 1 compares the different heuristics regarding the number of solved instances, average number of imported clauses by each thread, average size of imported clauses, solving time and speedup. Despite the number of solved instances not changing significantly, randomly sharing clauses can deteriorate the performance of the solver. Note that the solving time presented in table 1 only considers instances that were solved by all heuristics. LBD and size heuristics have similar speedups. Other sizes were also evaluated. It was observed that if the limit is too small then the speedup is reduced since not many clauses are shared. On the other hand, if the limit is too large then the speedup is also reduced since many irrelevant clauses are shared. Nevertheless, a size limit of 32 is comparable to a limit of 8 since there are some instances where learning larger clauses can be more useful than just learning smaller clauses. The dynamic heuristic outperforms the static heuristics but is outperformed by the freezing heuristic.

To summarize, although sharing learned clauses does not improve the number of solved instances significantly, it does reduce the solving time considerably. The freezing heuristic clearly outperforms all other heuristics in terms of solving time and number of instances solved and provides a strong stimulus for further research.

5 Conclusions

Recently, new parallel algorithms have been proposed for SAT as well as for MaxSAT. One of the main goals of parallel algorithms is to be able to take advantage of new multicore computer architectures by running several threads at the same time. One way to speedup these algorithms is to be able to share learned clauses between the different threads, thus allowing the pruning of the search space already explored in other threads.

In this paper different sharing heuristic procedures already proposed for SAT are described and used in a MaxSAT parallel solver. Moreover, a new heuristic based on the notion of freezing is proposed. This heuristic delays the import of shared clauses by a given thread until it is considered relevant in the context of its own search.

Experimental results show that sharing learned clauses in a portfolio-based parallel MaxSAT solver does not increase significantly the number of solved instances. However, it does allow a considerable reduction of the solving time. Finally, our proposed

freezing heuristic outperforms all other heuristics both in solving time and number of solved instances.

As future work one might consider the aggregation of several heuristic criteria. Variations of the freezing heuristic can also be devised that take into consideration other information from the context of the search space being explored in the importing thread.

Acknowledgements

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

References

1. 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.
2. G. Audemard, J.-M. Lagniez, B. Mazure, and L. Sais. On Freezing and Reactivating Learnt Clauses. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 188–200, 2011.
3. G. Audemard and L. Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *International Joint Conferences on Artificial Intelligence*, pages 399–404, 2009.
4. 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.
5. Y. Hamadi, S. Jabbour, C. Piette, and L. Sais. Deterministic Parallel DPLL: System Description. In *Pragmatics of SAT Workshop*, 2011.
6. Y. Hamadi, S. Jabbour, and L. Sais. Control-Based Clause Sharing in Parallel SAT Solving. In *International Joint Conferences on Artificial Intelligence*, pages 499–504, 2009.
7. C. M. Li and F. Manyà. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, pages 613–631. IOS Press, 2009.
8. 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.
9. R. Martins, V. Manquinho, and I. Lynce. Exploiting Cardinality Encodings in Parallel Maximum Satisfiability. In *International Conference on Tools with Artificial Intelligence*, pages 313–320, 2011.
10. R. Martins, V. Manquinho, and I. Lynce. Parallel Search for Boolean Optimization. In *RCRA International Workshop on Experimental Evaluation of Algorithms for solving problems with combinatorial explosion*, 2011.
11. 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.