Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions, volume B-2014-2 of Department of Computer Science Series of Publications B, University of Helsinki 2014. ISBN 978-951-51-0043-6.

PeneLoPe in SAT Competition 2014

Gilles Audemard, Benoît Hoessen, Saïd Jabbour, Jean-Marie Lagniez, Cédric Piette
Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens
{audemard,hoessen,jabbour,lagniez,piette}@cril.fr

***Abstract*—This paper provides a short system description of our updated portfolio-based solver called `PeneLoPe`, based on `ManySat`. Particularly, this solver focuses on collaboration between threads, providing different policies for exporting and importing learnt clauses between CDCL searches. Moreover, different restart strategies are also available, together with a deterministic mode.**

I. Overview

PeneLoPe [2] is a portfolio parallel SAT solver that uses the most effective techniques proposed in the sequential framework: unit propagation, lazy data structures, activity-based heuristics, progress saving for polarities, clause learning, etc. As for most of existing solvers, a first preprocessing step is achieved. For this step -which is typically sequential- we have chosen to make use of SatElite [6].

In addition, PeneLoPe includes a recent technique for its learnt clause database management. Roughly, this approach follows this schema: each learnt clause c is periodically evaluated with a so-called *psm* measure [3], which is equal to the size of the set-theoretical intersection of the current interpretation and c. Clauses that exhibit a low *psm* are considered relevant. Indeed, the lower is a *psm* value, the more likely the related clause is about to unit-propagate some literal, or to be falsified. On the opposite, a clause with a large *psm* value has a lot of chance to be satisfied by many literals, making it irrelevant for the search in progress.

Thus, only clauses that exhibit a low *psm* are selected and currently used by the solver, the other clauses being *frozen*. When a clause is frozen, it is removed from the list of the watched literals of the solver, in order to avoid the computational over-cost of maintaining the data structure of the solver for this useless clause. Nevertheless, a frozen clause is not erased but it is kept in memory, since this clause may be useful in the next future of the search. As the current interpretation evolves, the set of learnt clauses actually used by the solver evolves, too. In this respect, the *psm* value is computed periodically, and sets of clauses are frozen or unfrozen with respect to their freshly computed new value.

Let P_k be a sequence where $P_0 = 500$ and $P_{i+1} = P_i + 500 + 100 \times i$. A function "*updateDB*" is called each time the number of conflict reaches P_i conflicts (where $i \in [0..\infty[$). This function computes new *psm* values for every learnt clauses (frozen or activated). A clause that has a *psm* value less than a given limit l is activated in the next part of the search. If its *psm* does not hold this condition, then it is frozen. Moreover, a clause that is not activated after k (equal to 7 by default) time steps is deleted. Similarly, a clause remaining active more than k steps without participating to the search is also permanently deleted (see [3] for more details).

Besides the *psm* technique, PeneLoPe also makes use of the *lbd* value defined in [4]. *lbd* is used to estimate the quality of a learnt clause. This new measure is based on the number of different decision levels appearing in a learnt clause and is computed when the clause is generated. Extensive experiments demonstrates that clauses with small *lbd* values are used more often than those with higher *lbd* ones. Note also that *lbd* of clauses can be recomputed when they are used for unit propagations, and updated if it becomes smaller. This update process is important to get many good clauses.

Given these recently defined heuristic values, we present in the next Section several strategies implemented in PeneLoPe.

II. Detailled Features

PeneLoPe proposes a certain number of strategies regarding importation and exportation of learnt clauses, restarts, and the possibility of activating a deterministic mode.

Importing clause policy: When a clause is imported, we can consider different cases, depending on the moment the clause is attached for participating to the search.

- *no-freeze*: each imported clause is actually stored with the current learnt database of the thread, and will be evaluated (and possibly frozen) during the next call to *updateDB*
- *freeze-all*: each imported clause is *frozen* by default, and is only used later by the solver if it is evaluated relevant w.r.t. unfreezing conditions.
- *freeze*: each imported clause is evaluated as it would have been if locally generated. If the clause is considered relevant, it is added to the learnt clauses, otherwise it is frozen

Exporting clause policy: Since PeneLoPe can freeze clauses, each thread can import more clauses than it would with a classical management of clauses, where all of them are attached. Then, we propose different strategies, more or less restrictive, to select which clauses have to be shared:

- *unlimited*: any generated clause is exported towards the different threads.
- *size limit*: only clauses whose size is less than a given value (8 in our experiments) are exported [8].
- *lbd limit*: a given clause c is exported to other threads if its *lbd* value $lbd(c)$ is less than a given limit value d (8

by default). Let us also note that the *lbd* value can vary over time, since it is computed with respect to the current interpretation. Therefore, as soon as $lbd(c)$ is less than d, the clause is exported.

Restarts policy: Beside exchange policies, we define two restart strategies.

- *Luby*: Let l_i be the i^{th} term of the Luby serie. The i^{th} restart is achieved after $l_i \times \alpha$ conflicts (α is set to 100 by default).
- *LBD* [4]: Let LBD_g be the average value of the LBD of each learnt clause since the beginning. Let LBD_{100} be the same value computed only for the last 100 generated learnt clause. With this policy, a restart is achieved as soon as $LBD_{100} \times \alpha > LBD_g$ (α is set to 0.7 by default). In addition, the VSIDS score of variables that are unit-propagated thank for a learnt clause whose *lbd* is equal to 2 are increased, as detailled in [4].

Furthermore, we have implemented in `PeneLoPe` a deterministic mode which ensures full reproducibility of the results for both runtime and reported solutions (model or refutation proof). Large experiments show that such mecanism does not affect significantly the solving process of portfolio solvers [7]. Quite obviously, this mode can also be unactivated in `PeneLoPe`.

III. Fine Tuning Parameters of PeneLoPe

`PeneLoPe` is designed to be fine-tuned in an easy way, namely without having to modify its source code. To this end, a configuration file (called `configuration.ini`, an example is provided in Figure 1) is proposed to describe the default behavior of each thread. This file actually contains numerous parameters that can be modified by the user before running the solver. For instance, besides export, import and restart strategies, one can choose the number of threads that the solver uses, the α factor if the Luby techniques is activated for the restart strategy, etc. Each policy and/or value can obviouly differ from one thread to the other, in order to ensure diversification.

```
ncores = 8
deterministic = false
;this is the default behavior of each
;thread, can be modified or specified
;after each [solverX] item
[default]
;if set to true, then psm is used
usePsm = true
;allowed values: avgLBD, luby
restartPolicy = avgLBD
;allowed values: lbd, unlimited, size
exportPolicy = lbd
;allowed values:
;freeze, no-freeze, freeze-all
importPolicy = freeze
;number of freeze before the clause
;is deleted
maxFreeze = 7
;initial #conflict before the first
;updateDB
initialNbConflictBeforeReduce = 500
;incremental factor for updateDB
nbConflictBeforeReduceIncrement = 100
;maximum lbd value for exchanged clauses
maxLBDExchange = 8
[solver0]
importPolicy = no-freeze
[solver1]
initialNbConflictBeforeReduce = 5000
nbConflictBeforeReduceIncrement = 1000
[solver2]
maxFreeze = 8
;solver3 is the default solver
[solver3]
[solver4]
restartPolicy = luby
lubyFactor = 100
[solver5]
exportPolicy = size
[solver6]
maxFreeze = 4
[solver7]
importPolicy = freeze-all
```

Fig. 1. Configuration.ini file

Acknowledgment

`PeneLoPe` has been partially developed thank to the financial support of CNRS and OSEO, under the ISI project "Pajero".

References

[1] Gilles Audemard, Benoît Hoessen, Said Jabbour, Jean-Marie Lagniez, and Cédric Piette. Penelope, a parallel clause-freezer solver. In *proceedings of SAT Challenge 2012: Solver and Benchmarks Descriptions*, pages 43–44, Lens, may 2012.

[2] Gilles Audemard, Benoît Hoessen, Saïd Jabbour, Jean-Marie Lagniez, and Cédric Piette. Revisiting clause exchange in parallel sat solving. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 200–213. Springer Berlin Heidelberg, 2012.

[3] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Saïs. On freezeing and reactivating learnt clauses. In *proceedings of SAT*, pages 147–160, 2011.

[4] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *proceedings of IJCAI*, pages 399–404, 2009.

[5] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT*, page 2008.

[6] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.

[7] Youssef Hamadi, Said Jabbour, Cédric Piette, and Lakhdar Saïs. Deterministic parallel DPLL. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4):127–132, 2011.

[8] Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. Control-based clause sharing in parallel SAT solving. In *proceedings of IJCAI*, pages 499–504, 2009.

[9] Knot Pipatsrisawat and Adnan Darwiche. Width-based restart policies for clause-learning satisfiability solvers. In *SAT*, pages 341–355, 2009.