since each application may change the resulting CNF formula. This is not the case for backbone identification, and this explains why it is performed at start, only. Observe that each of the remaining techniques generates a CNF formula which is a logical consequence of its input. As a consequence, if a literal belongs to the backbone of a CNF formula which results from the composition of such elementary preprocessings, then it belongs as well to the backbone of the CNF formula considered initially. Any further call to backboneSimpl would just be a waste of time. Within pmc the other elementary preprocessings can be performed several times. Iteration stops when a fixpoint is reached (i.e., the output of the preprocessing is equal to its input) or when a preset (maximal) number $numTries$ of iterations is reached. In our experiments $numTries$ was set to 10.

Algorithm 7: pmc

input : a CNF formula Σ
output: a CNF formula Φ such that $\|\Phi\| = \|\Sigma\|$

$\Phi \leftarrow \Sigma$;
if $optB$ **then** $\Phi \leftarrow$ backboneSimpl(Φ);
$i \leftarrow 0$;
while $i < numTries$ **do**
  $i \leftarrow i + 1$;
  if $optO$ **then** $\Phi \leftarrow$ occurrenceSimpl(Φ);
  if $optG$ **then** $\Phi \leftarrow$ gatesSimpl(Φ);
  if $optV$ **then** $\Phi \leftarrow$ vivificationSimpl(Φ);
  if *fixpoint* **then** $i \leftarrow numTries$;
return Φ

Empirical Evaluation

Setup. In our experiments, we have considered two combinations of the elementary preprocessings described in the previous section:

- eq corresponds to the parameter assignment of pmc where $optV = optB = optO = 1$ and $optG = 0$. It is equivalence-preserving and can thus be used for weighted model counting.
- $\#eq$ corresponds to the parameter assignment of pmc where $optV = optB = optO = 1$ and $optG = 1$. This combination is guaranteed only to preserve the number of models of the input.

As to model counting, we have considered both a "direct" approach, based on the state-of-the-art model counter Cachet (www.cs.rochester.edu/~kautz/Cachet/index.htm) (Sang et al. 2004), as well as a compilation-based approach, based on the C2D compiler (reasoning.cs.ucla.edu/c2d/) which generates (smooth) d−DNNF compilations (Darwiche 2001). As explained previously, it does not make sense to use the $\#eq$ preprocessing upstream to C2D.

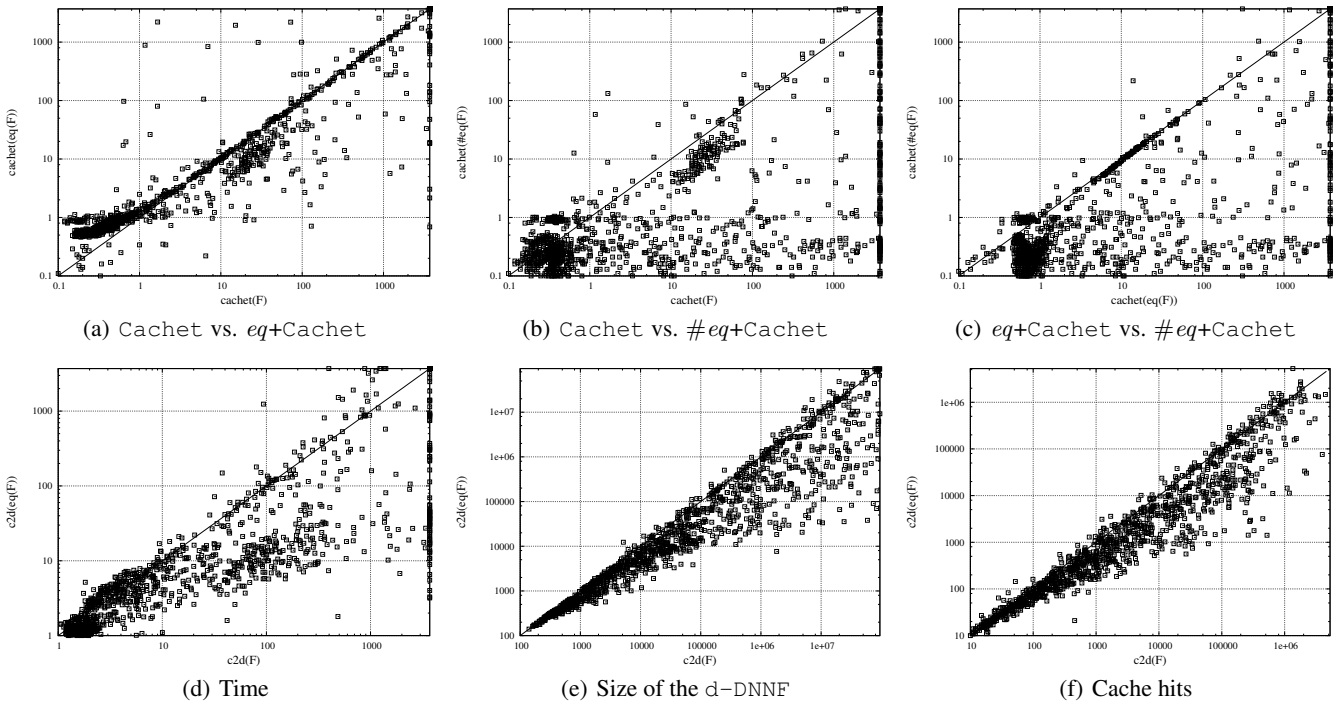
We made quite intensive experiments on a number of CNF instances Σ from different domains, available in the SAT LIBrary (www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html). 1342 instances Σ from 19 families have been used. The aim was to count the number of models of each Σ using pmc for the two combinations of preprocessings listed above, and to determine whether the combination(s) under consideration prove(s) or not useful for solving it. Our experiments have been conducted on a Quad-core Intel XEON X5550 with 32GB of memory. A time-out of 3600 seconds per instance Σ has been considered for Cachet and the same time-out has been given to C2D for achieving the compilation of Σ and computing the number of models of the resulting d−DNNF formula. Our preprocessor pmc and some detailed empirical results, including the benchmarks considered in our experiments, are available at www.cril.fr/PMC/pmc.html.

Impact on Cachet and on C2D Figure 1 (a)(b)(c) illustrates the comparative performances of Cachet, being equipped (or not) with some preprocessings. No specific optimization of the preprocessing achieved depending on the family of the instance under consideration has been performed: we have considered the eq preprocessing and the $\#eq$ preprocessing for every instance. Each dot represents an instance and the time needed to solve it using the approach corresponding to the x-axis (resp. y-axis) is given by its x-coordinate (resp. y-coordinate). In part (a) of the figure, the x-axis corresponds to Cachet (without any preprocessing) while the y-axis corresponds to $\#eq$+Cachet. In part (b), the x-axis corresponds to Cachet (without any preprocessing) and the y-axis corresponds to $\#eq$+Cachet. In part (c), the x-axis corresponds to eq+Cachet and the y-axis corresponds to $\#eq$+Cachet.

In Figure 1 (d)(e)(f), the performance of C2D is compared with the one offered by eq+C2D. As on the previous figure, each dot represents an instance. Part (d) of the figure is about compilation time (plus the time needed to count the models of the compiled form), while part (e) is about the size of the resulting d−DNNF formula. Part (f) of the figure reports the number of cache hits.

Synthesis. Table 1 synthesizes some of the results. Each line compares the performances of two approaches to model counting (say, A and B), based on Cachet or C2D, and using or not some preprocessing techniques. For instance, the first line compares Cachet without any preprocessing (A) with eq+Cachet (B). #s(A or B) indicates how many instances (over 1342) have been solved by A or by B (or by both of them) within the time limit. #s(A) - #s(B) (resp. #s(B) - #s(A)) indicates how many instances have been solved by A but not by B (resp. by B but not by A) within the time limit. #s(A and B) indicates how many instances have been solved by both A and B, and T_A (resp. T_B) is the cumulated time (in seconds) used by A (resp. B) to solve them. The preprocessing time $T_{\texttt{pmc}}$ spent by pmc is included in the times reported in Figure 1 and in Table 1. $T_{\texttt{pmc}}$ is less than 1s (resp. 10s, 50s) for 80% (resp. 90%, 99%) of the instances.

The empirical results clearly show both the efficiency and the robustness of our preprocessing techniques. As to efficiency, the number of instances which can be solved within the time limit when a preprocessing is used is always higher, and sometimes significantly higher, than the the corresponding number without preprocessing. Similarly, the cumulated time needed to solve the commonly solved instances is always smaller (and sometimes significantly smaller) than



(a) Cachet vs. *eq*+Cachet (b) Cachet vs. #*eq*+Cachet (c) *eq*+Cachet vs. #*eq*+Cachet

(d) Time (e) Size of the d−DNNF (f) Cache hits

Figure 1: Comparing Cachet with (*eq* or #*eq*)+Cachet (above) and C2D with *eq*+C2D (below) on a large panel of instances from the SAT LIBrary.

A	B	#s(A or B)	#s(A) - #s(B)	#s(B) - #s(A)	#s(A and B)	T_A	T_B
Cachet	*eq*+Cachet	1047	0	28	1019	98882.6	83887.7
Cachet	#*eq*+Cachet	1151	1	132	1018	97483.9	16710.2
eq+Cachet	#*eq*+Cachet	1151	1	104	1046	111028.0	18355.4
C2D	*eq*+C2D	1274	7	77	1190	123923.0	53653.2

Table 1: A synthesis of the empirical results about the impact of preprocessing on Cachet and C2D

the corresponding time without any preprocessing. Interestingly, *eq*+C2D also leads to substantial space savings compared to C2D (our experiments showed that the size of the resulting d−DNNF formulae can be more than one order of magnitude larger without preprocessing, and that the cumulated size is more than 1.5 larger). This is a strong piece of evidence that the practical impact of pmc is not limited to the model counting issue, and that the *eq* preprocessing can also prove useful for equivalence-preserving knowledge compilation. As to robustness, the number of instances solved within the time limit when no preprocessing has been used and not solved within it when a preprocessing technique is considered remains very low, whatever the approach to model counting under consideration. Finally, one can also observe that the impact of the equivalence/gates detection and replacement is huge (#*eq*+Cachet is a much better performer than *eq*+Cachet).

Conclusion

We have implemented a preprocessor pmc for model counting which includes several preprocessing techniques, especially vivification, occurrence reduction, backbone identification, as well as equivalence, AND and XOR gate identification and replacement. The experimental results we have obtained show that pmc is useful.

This work opens several perspectives for further research. Beyond size reduction, it would be interesting to determine some explanations to the improvements achieved by taking advantage of pmc (for instance, whether they can lead to a significant decrease of the treewidth of the input CNF formula). It would be useful to determine the "best" combinations of elementary preprocessings, depending on the benchmark families. It would be nice to evaluate the impact of pmc when other approaches to model counting are considered, especially approximate model counters (Wei and Selman 2005; Gomes and Sellmann 2004) and other compilation-based approaches (Koriche et al. 2013; Bryant 1986; Darwiche 2011). Assessing whether pmc prove useful upstream to other knowledge compilation techniques (for instance (Boufkhad et al. 1997; Subbarayan, Bordeaux, and Hamadi 2007; Fargier and Marquis 2008; Bordeaux and Marques-Silva 2012)) would also be valuable.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. This work is partially supported by the project BR4CP ANR-11-BS02-008 of the French National Agency for Research.

References

Apsel, U., and Brafman, R. I. 2012. Lifted MEU by weighted model counting. In *Proc. of AAAI'12*.

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern sat solver. In *Proc. of IJCAI'09*, 399–404.

Audemard, G.; Lagniez, J.-M.; and Simon, L. 2013. Just-in-time compilation of knowledge bases. In *Proc. of IJCAI'13*, 447–453.

Bacchus, F., and Winter, J. 2004. Effective preprocessing with hyper-resolution and equality reduction. In *Proc. of SAT'04*, 341–355.

Bordeaux, L., and Marques-Silva, J. 2012. Knowledge compilation with empowerment. In *Proc. of SOFSEM'12*, 612–624.

Bordeaux, L.; Janota, M.; Marques-Silva, J.; and Marquis, P. 2012. On unit-refutation complete formulae with existentially quantified variables. In *Proc. of KR'12*, 75–84.

Boufkhad, Y., and Roussel, O. 2000. Redundancy in random SAT formulas. In *Proc. of AAAI'00*, 273–278.

Boufkhad, Y.; Grégoire, E.; Marquis, P.; Mazure, B.; and Saïs, L. 1997. Tractable cover compilations. In *Proc. of IJCAI'97*, 122–127.

Bryant, R. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35(8):677–692.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172(6-7):772–799.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proc. of IJCAI'11*, 819–826.

del Val, A. 1994. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proc. of KR'94*, 551–561.

Domshlak, C., and Hoffmann, J. 2006. Fast probabilistic planning through weighted model counting. In *Proc. of ICAPS'06*, 243–252.

Een, N., and Biere, A. 2005. Effective preprocessing in sat through variable and clause elimination. In *Proc. of SAT'05*, 61–75.

Fargier, H., and Marquis, P. 2008. Extending the knowledge compilation map: Krom, Horn, affine and beyond. In *Proc. of AAAI'08*, 442–447.

Gomes, C., and Sellmann, M. 2004. Streamlined constraint reasoning. In *Proc. of CP'04*. 274–289.

Han, H., and Somenzi, F. 2007. Alembic: An efficient algorithm for cnf preprocessing. In *Proc. of DAC'07*, 582–587.

Heule, M.; Järvisalo, M.; and Biere, A. 2010. Clause elimination procedures for cnf formulas. In *Proc. of LPAR'10*, 357–371.

Heule, M. J. H.; Järvisalo, M.; and Biere, A. 2011. Efficient cnf simplification based on binary implication graphs. In *Proc. of SAT'11*, 201–215.

Järvisalo, M.; Biere, A.; and Heule, M. 2012. Simulating circuit-level simplifications on cnf. *Journal of Automated Reasoning* 49(4):583–619.

Koriche, F.; Lagniez, J.-M.; Marquis, P.; and Thomas, S. 2013. Knowledge compilation for model counting: Affine decision trees. In *Proc. of IJCAI'13*, 947–953.

Liberatore, P. 2005. Redundancy in logic I: CNF propositional formulae. *Artificial Intelligence* 163(2):203–232.

Lynce, I., and Marques-Silva, J. 2003. Probing-based preprocessing techniques for propositional satisfiability. In *In Proc. ICTAI'03*, 105–110.

Monasson, R.; Zecchina, R.; Kirkpatrick, S.; Selman, B.; and Troyansky, L. 1999. Determining computational complexity from characteristic 'phase transitions'. *Nature* 33:133–137.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. In *Proc. of DAC '01*, 530–535.

Ostrowski, R.; Grégoire, E.; Mazure, B.; and Saïs, L. 2002. Recovering and exploiting structural knowledge from cnf formulas. In *Proc. of CP'02*, 185–199.

Palacios, H.; Bonet, B.; Darwiche, A.; and Geffner, H. 2005. Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proc. of ICAPS'05*, 141–150.

Piette, C.; Hamadi, Y.; and Saïs, L. 2008. Vivifying propositional clausal formulae. In *Proc. of ECAI'08*, 525–529.

Roth, D. 1996. On the hardness of approximate reasoning. *Artificial Intelligence* 82(1–2):273–302.

Sang, T.; Beame, P.; and Kautz, H. A. 2005. Performing Bayesian inference by weighted model counting. In *Proc. of AAAI'05*, 475–482.

Sang, T.; Bacchus, F.; Beame, P.; Kautz, H.; and Pitassi, T. 2004. Combining component caching and clause learning for effective model counting. In *Proc. of SAT'04*.

Subbarayan, S., and Pradhan, D. K. 2004. Niver: Non increasing variable elimination resolution for preprocessing sat instances. In *Proc. of SAT'04*, 276–291.

Subbarayan, S.; Bordeaux, L.; and Hamadi, Y. 2007. Knowledge compilation properties of tree-of-BDDs. In *Proc. of AAAI'07*, 502–507.

Valiant, L. G. 1979. The complexity of computing the permanent. *Theoretical Computer Science* 8:189–201.

Wei, W., and Selman, B. 2005. A new approach to model counting. In *Proc. of SAT'05*, 324–339.

Zhang, H., and Stickel, M. E. 1996. An efficient algorithm for unit propagation. In *Proc. of ISAIM96*, 166–169.