3. Else, Δ is built from clauses of $\overline{\alpha}$ and Γ (and possibly from clauses of Π, too). In this case, since at least one clause of $\overline{\alpha}$ is in Δ and since Δ is unsatisfiable, we have that $(\Delta \setminus \overline{\alpha}) \wedge \overline{\alpha}$ is unsatisfiable, i.e., α is a deductive consequence of $\Delta \setminus \overline{\alpha}$. Although it cannot be represented by a set of clauses in a direct manner, this information can be exploited and implemented easily (and without significant additional space cost) using clause selectors markers *à la* (Oh et al. 2004), as follows. Each clause β of Σ is associated to a new corresponding dedicated literal noted λ_β, called selector, and β is replaced by $\beta \vee \neg\lambda_\beta$ in Σ. A clause is active in (i.e., belongs to) a set of clauses if its selector is assigned to 1. With these selectors, $\Delta \setminus \overline{\alpha} \models \alpha$ can be represented by the clause $(\neg\lambda_{\delta_1} \vee \cdots \vee \neg\lambda_{\delta_m} \vee \lambda_\alpha)$ where δ_i ($i \in [1..m]$) are the m clauses of $\Delta \setminus \overline{\alpha}$. When all clauses of $\Delta \setminus \overline{\alpha}$ are active then α must be also active.

It is easy to show that all those improvements do not alter the correctness of the algorithm.

Related work

Handling over-constrained systems though maximal satisfiable or minimal correction subsets has long been an active subject of research in A.I. (see pioneering work for example in (Meseguer et al. 2003)). In the general constraint networks setting, a seminal approach called QUICKXPLAIN has been described in (Junker 2004). In the same framework, improved techniques can be found in e.g. (Hemery et al. 2006).

In order to solve a constraint network instance or extract its MSSes and CoMSSes, it is often more efficient to encode the network through a set of Boolean clauses and benefit from SAT technology (provided that the size of the Boolean instance does not blow-up). In the SAT domain, many approaches have been proposed to compute MSSes and CoMSSes. The earliest approaches were based on DPLL-based SAT solvers (see e.g. (Birnbaum and Lozinskii 2003)) but are quite inefficient compared to more recent approaches (Liffiton and Sakallah 2008) based on MAX-SAT. As stressed in (Marques-Silva et al. 2013), the latter approaches also proved more efficient than various algorithms based on iterative calls to a SAT solver like (Bailey and Stuckey 2005). Much research has also been conducted in model-based diagnosis. Noticeably, a recent approach, called FastDiag (hereafter noted BFD for Basic FastDiag), (Felfernig, Schubert, and Zehentner 2012) has adapted QUICKXPLAIN from (Junker 2004) to the Boolean case and proves often very competitive.

Recently (Marques-Silva et al. 2013) experimentally compared the best MSS and CoMSS extraction approaches in the Boolean setting, namely the above BFD algorithm, BLS (depicted in Algorithm 1), EFD and ELS for Enhanced FastDiag and Enhanced Linear Search through additional features discussed in (Marques-Silva et al. 2013). The same authors also proposed a novel algorithm for computing CoMSSes that they experimentally showed to be the most efficient and robust one for CoMSSes computation, compared with the aforementioned list of approaches. Roughly, this algorithm, called CLD, extracts one CoMSS by using the `extendSatPart` principle iteratively (and also using backbone considerations, among other things). The authors showed that CLD experimentally outperforms all the competing approaches.

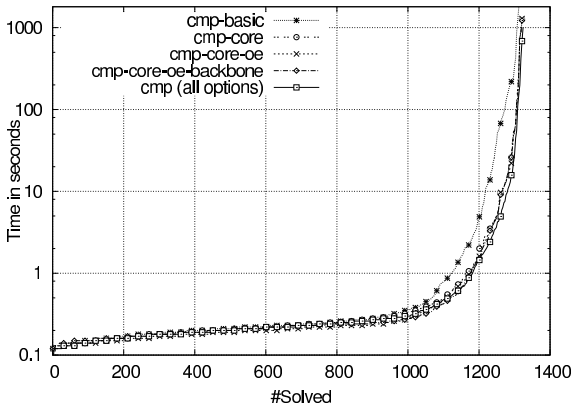

Figure 2: CMP variants on SAT/MAX-SAT instances

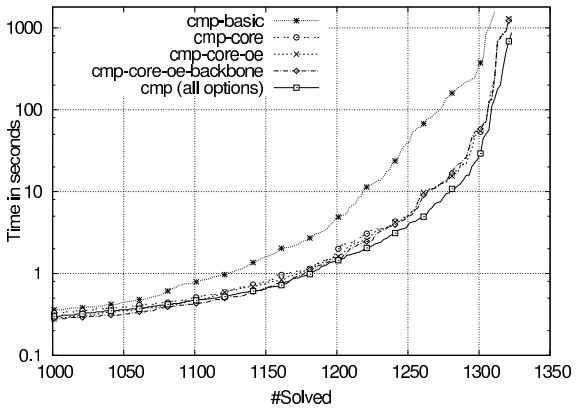

Figure 3: Zoom in on Figure 2

Although it implements the backbones and `extendSatPart` features, CMP is very different in nature as its key principle is the search for transition constraints rather than iterating on `extendSatPart`. Moreover, it includes the novel *opposite enforced* and `exploitCore` features.

Experimental results

All experimentations have been conducted on Intel Xeon E5-2643 (3.30GHz) processors with 7.6Gb RAM on Linux CentOS. Time limit was set to 30 minutes.

Two series of benchmarks have been considered. The first one was made of the 1343 benchmarks used and referred to in (Marques-Silva et al. 2013): they are small-sized industrial-based instances from SAT competitions www.satcompetition.org and structured instances from the MAX-SAT evaluations maxsat.ia.udl.cat:81. We enriched this experimentation setting by also considering a second series of benchmarks, made of *all* the 295 instances used for the 2011 MUS competition organized in parallel with the SAT one. Note that in all Figures that we are going to present,

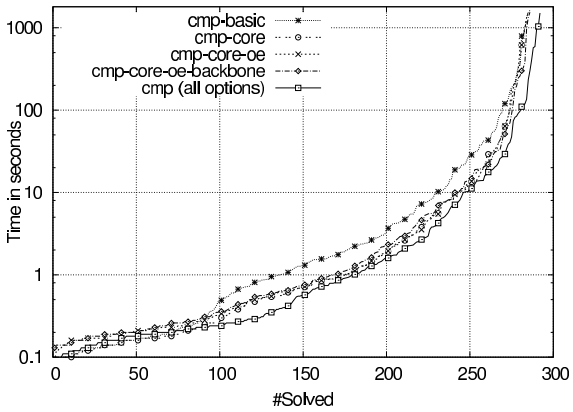

Figure 4: CMP variants on MUS instances

	SAT/MAX-SAT inst.	MUS inst.
Number of instances	1343	295
CMP-*basic*	1312	285
CMP-*core*	1321	285
CMP-*core-oe*	1321	286
CMP-*core-oe-backbone*	1322	286
CMP (all options)	1327	292

Table 1: Number of successfully partitioned instances

when the y-values represent the CPU times to partition a given number x of instances through different approaches, this does not mean that the x partitioned instances are necessarily identical and that each instance is partitioned in an identical way by all the considered approaches.

CMP is implemented in C++. MINISAT (Eén and Sörensson 2004) was selected as the CDCL SAT-solver. CMP and all experimentation data are available from www.cril.univ-artois.fr/documents/cmp/.

First, the actual benefits of the four optional parts of CMP have been assessed experimentally.

The basic version of CMP, i.e. Algorithm 2 without any of the options, was re-named CMP-*basic*. CMP-*basic* was then tentatively enhanced by taking the options into account in an incremental way. Only adding `exploitCore` (Algo2:line 15) gave rise to CMP-*core*; adding also the opposite-enforced feature (Algo2:line 8) yielded CMP-*core-oe*. CMP-*core-oe-backbone* was obtained by also activating the backbone literals tentative enhancement (Algo2:line 11). From now on, CMP denotes Algorithm 2 with all options, which thus included `extendSatPart` (Algo2:line 5) as well.

Figures 2 and 4 show gradual improvements when each of the options was taken into account in a cumulative way: each additional option allowed for some additional efficiency gain. Figure 3 is a focus on the rightest part of curves of Figure 2. Table 1 shows that the number of successful partitionings also increased according to the considered range of options. Noticeably, the `exploitCore` (*core*) option delivered the most significant improvement in terms of both computing time and number of successfully partitioned instances. Other ways to combine the options together allowed

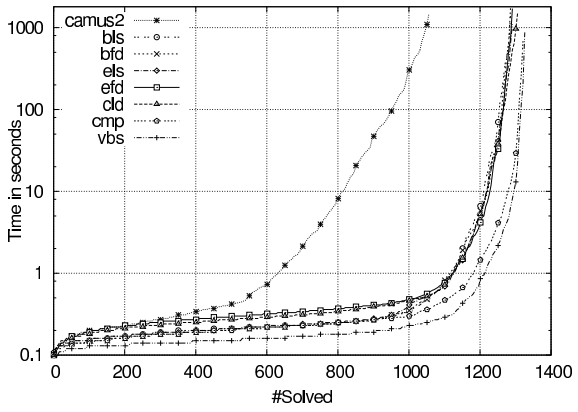

Figure 5: Comparison on SAT and MAX-SAT instances

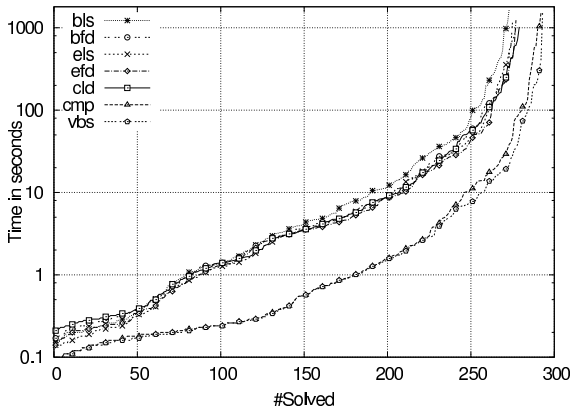

Figure 6: Comparison on MUS instances

similar observations to be made.

Then, we compared CMP with existing approaches that we mentioned earlier: namely, the BFD, BLS, CLD, EFD and ELS algorithms, which are described in (Marques-Silva et al. 2013) and implemented in the MCS_{LS} tool logos.ucd.ie/wiki/doku.php?id=mcsls. The version 2 of CAMUS (sun.iwu.edu/~mliffito/camus/ (Liffiton and Sakallah 2008; 2009)), named CAMUS2, was also tested on SAT/MAX-SAT benchmarks but this earlier system allowed a significantly smaller number of instances to be partitioned, only. Table 2 shows that CMP-*basic* itself allows more instances to be partitioned than any of the competitors. Figures 5 and 6 also compare the investigated approaches. We have also drawn the VBS (*Virtual Best Solver*) curve, which represents for each instance the best computing time amongst the tested methods. Clearly, CMP appeared best performing and was very close to VBS. For each method, Table 2 gives the number of instances that were successfully partitioned, with the highest score for CMP, too.

Discussion and perspectives

The extensive experimentations that we have conducted show that CMP is more robust than previous approaches and allows more Boolean instances to be successfully par-

	SAT/MAX-SAT inst.	MUS inst.
Number of instances	1343	295
BLS	1287	273
BFD	1287	276
ELS	1293	277
EFD	1291	277
CLD	1307	279
CMP-*basic*	1312	285
CMP (all options)	1327	292
VBS	1327	293

Table 2: Number of successfully partitioned instances

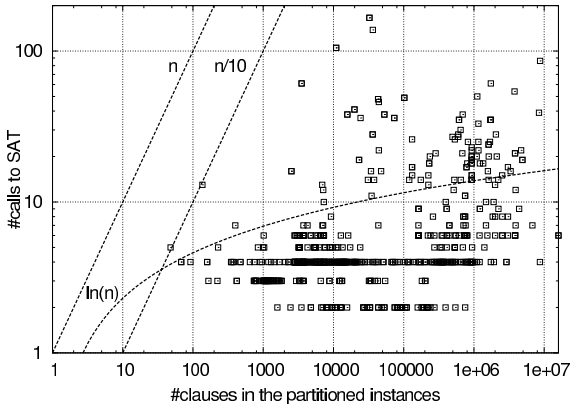

Figure 7: Number of calls by CMP to the SAT solver

titioned. Obviously, this does not entail that CMP would be more efficient for *every* instance. In this respect, it is worth mentioning that CMP exhibits a higher worst-case complexity than for example BLS, namely the basic linear search approach. Indeed, when n is the number of clauses in the instance, CMP requires $O(n^2)$ calls to a SAT solver in the worst case whereas BLS requires a linear number of such calls, only. Note however that the number of calls by CMP to the SAT solver always remains very significantly lower than n for all successfully partitioned instances (Figure 7).

We envision several paths for further research. First, the method could be enhanced by making use of incremental SAT solvers (Lagniez and Biere 2013; Audemard, Lagniez, and Simon 2013). Second, CMP can offer a way to approximate MAX-SAT when solving this latter problem is out of reach for hard instances. In this respect, although the features in CMP are not directly exportable to MAX-SAT algorithms, we believe that the way MSSes are computed in CMP can lead to novel ways to compute MAX-SAT. Also, CMP delivers approximate solutions for a basic version of MAX-SAT: in the future, we plan to study how to push the envelope in order to address *weighted* MAX-SAT. Finally, it would also be interesting to extend CMP to address the problem of enumerating CoMSS (or MSS) and study whether or not this could improve recent practical computational results about this issue (Marques-Silva et al. 2013).

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, IJCAI'09, 399–404.

Audemard, G., and Simon, L. 2012. Refining restarts strategies for sat and unsat. In *Principles and Practice of Constraint Programming - 18th International Conference (CP'12)*, volume 7514 of *Lecture Notes in Computer Science*, 118–126. Springer.

Audemard, G.; Lagniez, J.-M.; Mazure, B.; and Saïs, L. 2011. On freezing and reactivating learnt clauses. In *Proceedings of the 14th International Conference on Theory and Application of Satisfiability Testing (SAT'11)*, 188–200. Springer.

Audemard, G.; Lagniez, J.-M.; and Simon, L. 2013. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT'13)*, volume 7962 of *Lecture Notes in Computer Science*, 309–317. Springer.

Bailey, J., and Stuckey, P. J. 2005. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages (PADL'2005)*, volume 3350 of *Lecture Notes in Computer Science*, 174–186. Springer.

Belov, A., and Marques-Silva, J. 2011. Accelerating MUS extraction with recursive model rotation. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design (FMCAD'11)*, 37–40. FMCAD Inc.

Besnard, P., and Hunter, A. 2008. *Elements of Argumentation*. The MIT Press.

Birnbaum, E., and Lozinskii, E. L. 2003. Consistent subsets of inconsistent systems: structure and behaviour. *J. Exp. Theor. Artif. Intell.* 15(1):25–46.

Chinneck, J. W. 2008. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, volume 118 of *International Series in Operations Research & Management Science*. Springer.

Eén, N., and Sörensson, N. 2004. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03). Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, 502–518. Springer.

Felfernig, A.; Schubert, M.; and Zehentner, C. 2012. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM* 26(1):53–62.

Fermé, E. L., and Hansson, S. O. 2011. AGM 25 years - twenty-five years of research in belief change. *J. Philosophical Logic* 40(2):295–331.

Ginsberg, M. 1987. *Readings in nonmonotonic reasoning*. M. Kaufmann Publishers.

Grégoire, É., and Konieczny, S. 2006. Logic-based approaches to information fusion. *Information Fusion* 7(1):4–18.

Grégoire, É.; Lagniez, J.-M.; and Mazure, B. 2013a. Improving MUC extraction thanks to local search. *CoRR* abs/1307.3585.

Grégoire, É.; Lagniez, J.-M.; and Mazure, B. 2013b. Questioning the importance of WCORE-like minimization steps in MUC-finding algorithms. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'13)*, 923–930.

Grégoire, É.; Mazure, B.; and Piette, C. 2007a. Boosting a complete technique to find MSS and MUS thanks to a local search oracle. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, 2300–2305.

Grégoire, É.; Mazure, B.; and Piette, C. 2007b. Local-search extraction of MUSes. *Constraints* 12(3):325–344.

Guo, L., and Lagniez, J.-M. 2011. Dynamic polarity adjustment in a parallel SAT solver. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI'11)*, 67–73.

Hamscher, W.; Console, L.; and de Kleer, J., eds. 1992. *Readings in Model-Based Diagnosis*. Morgan Kaufmann.

Hemery, F.; Lecoutre, C.; Saïs, L.; and Boussemart, F. 2006. Extracting MUCs from constraint networks. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI'06)*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, 113–117. IOS Press.

Junker, U. 2004. QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI'04)*, 167–172. AAAI Press.

Kilby, P.; Slaney, J. K.; Thiébaux, S.; and Walsh, T. 2005. Backbones and backdoors in satisfiability. In *Proceedings, The 20th National Conference on Artificial Intelligence (AAAI'05)*, 1368–1373. AAAI Press / The MIT Press.

Lagniez, J.-M., and Biere, A. 2013. Factoring out assumptions to speed up mus extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT'13)*, volume 7962 of *Lecture Notes in Computer Science*, 276–292. Springer.

Lecoutre, C. 2009. *Constraint Networks: Techniques and Algorithms*. Wiley.

Liffiton, M. H., and Sakallah, K. A. 2008. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning* 40(1):1–33.

Liffiton, M. H., and Sakallah, K. A. 2009. Generalizing core-guided Max-SAT. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *Lecture Notes in Computer Science*, 481–494. Springer.

Marques-Silva, J., and Sakallah, K. A. 1996. GRASP: a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design (ICCAD'96)*, 220–227. IEEE Computer Society.

Marques-Silva, J.; Heras, F.; Janota, M.; Previti, A.; and Belov, A. 2013. On computing minimal correction subsets. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*.

Meseguer, P.; Bouhmala, N.; Bouzoubaa, T.; Irgens, M.; and Sánchez, M. 2003. Current approaches for solving over-constrained problems. *Constraints* 8(1):9–39.

Monasson, R.; Zecchina, R.; Kirkpatrick, S.; Selman, B.; and Troyansky, L. 1999. Determining computational complexity from characteristic 'phase transitions'. *Nature* 400:133.

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 530–535. ACM.

Oh, Y.; Mneimneh, M. N.; Andraus, Z. S.; Sakallah, K. A.; and Markov, I. L. 2004. AMUSE: A minimally-unsatisfiable subformula extractor. In *Proceedings of the 41th Design Automation Conference (DAC'04)*, 518–523. ACM.

Papadimitriou, C. H. 1993. *Computational Complexity*. Addison-Wesley.

Pipatsrisawat, K., and Darwiche, A. 2007. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, 294–299. Springer.

Rossi, F.; van Beek, P.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*. Elsevier.

Zhang, L., and Malik, S. 2002. The quest for efficient boolean satisfiability solvers. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, 17–36. Springer.

Zhang, L.; Madigan, C.; Moskewicz, M.; and Malik, S. 2001. Efficient conflict driven learning in boolean satisfiability solver. In *Proc. of the Proceedings of IEEE/ACM International Conference on Computer Design (ICCAD'01)*, 279–285.