Concurrent Cube-and-Conquer (Poster Presentation)*

Peter van der Tak[1], Marijn J.H. Heule[1,2], and Armin Biere[3]

[1] Delft University of Technology, The Netherlands
[2] University of Texas, Austin, United States
[3] Johannes Kepler University Linz, Austria

Satisfiability solvers targeting industrial instances are currently almost always based on conflict-driven clause learning (CDCL) [5]. This technique can successfully solve very large instances. Yet on small, hard problems lookahead solvers [3] often perform better by applying much more reasoning in each search node and then recursively splitting the search space until a solution is found.

The *cube-and-conquer* (CC) approach [4] has shown that the two techniques can be combined, resulting in better performance particularly for very hard instances. The key insight is that lookahead solvers can be used to partition the search space into subproblems (called cubes) that are easy for a CDCL solver to solve. By first partitioning (*cube* phase) and then solving each cube (*conquer* phase), some instances can be solved within hours rather than days. This cube-and-conquer approach, particularly the *conquer* phase, is also easy to parallelize.

The challenge to make this technique work in practice lies in developing effective heuristics to determine when to stop partitioning and start solving. The current heuristics already give strong results for very hard instances, but are far from optimal and require some fine tuning to work well with instances of different difficulty. For example, applying too much partitioning might actually result in a considerable increase of run time for easy instances. On the other hand, applying not enough partitioning reduces the benefits of cube-and-conquer.

The most important problem in developing an improved heuristic is that in the partitioning phase no information is available about how well the CDCL solver will perform on a cube. In CC's heuristics, performance of CDCL is assumed to be similar to that of lookahead: if lookahead refutes a cube, CDCL is expected to be able to refute similar cubes fast, and if CDCL would solve a cube fast, lookahead is expected to be able to refute it fast too. However, due to the different nature of lookahead and CDCL, this is not always true.

To improve cutoff heuristics, we propose *concurrent cube-and-conquer* (CCC): an online approach that runs the cube and conquer phases concurrently. Whenever the lookahead solver makes a new decision, this decision is sent to the CDCL solver, which adds it as an assumption [2]. If CDCL refutes a cube fast, it will refute it before lookahead makes another decision. This naturally cuts off easy branches, so that the cutoff heuristic is no longer necessary.

Although this basic version of CCC already achieves speedups, it can be improved further by applying a (slightly different) cutoff heuristic. This heuristic

* The 2nd and 3rd author are supported by FWF, NFN Grant S11408-N23 (RiSE). The 2nd author is supported by DARPA contract number N66001-10-2-4087.

attempts to identify cubes similar to those that CDCL already solved, rather than estimating CDCL performance based on lookahead performance (as originally in CC). Cutting off has two advantages: often CDCL can already solve a cube efficiently without the last few decision variables; further partitioning these already easy cubes only hampers performance. Additionally, cutting off allows multiple cubes to be solved in parallel.

Other than improving performance of cube-and-conquer by replacing the cutoff heuristic, CCC also aims at solving another problem: on some instances (C)CC performs worse than CDCL regardless of the configuration of the solvers and heuristics. It seems that lookahead sometimes selects a decision l_{dec} which results in two subformulas $F \wedge l_{\text{dec}}$ and $F \wedge \neg l_{\text{dec}}$ that are not easier to solve separately by CDCL. If the decision is not relevant to CDCL search, (C)CC forces the CDCL solver to essentially solve the same problem twice. We propose two metrics that can detect this behavior, in which case CCC is aborted within 5 seconds and the problem is solved by CDCL alone.

Our experiments show that CCC works particularly well on crafted instances. Without selection of suitable instances, cube-and-conquer and CCC cannot compete with other solvers. However the proposed predictor based on CCC accurately selects instances for which cube-and-conquer techniques are not suitable and for which a CDCL search is preferred. It is thereby able to solve several more application and crafted instances than the CDCL and lookahead solvers it was based on. CCC solves 24 more crafted instances within one hour over all the SAT 2009 and 2011 competition instances than Plingeling [1], where both solvers use four threads. For application instances, Plingeling solves one more instance for a one hour timeout but CCC is slightly better for lower timeouts (anything below 2500 seconds).

We believe that CCC is particularly interesting as part of a portfolio solver, where our predictor can be used to predict whether to apply cube-and-conquer techniques. The authors of SATzilla specifically mention in their conclusion that identifying solvers that are only competitive for certain kinds of instances still has the potential to further improve SATzilla's performance substantially [6].

References

1. A. Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Technical Report 10/1, Johannes Kepler University, Linz, Austria, 2010.
2. N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *ENTCS*, 89(4):543–560, 2003.
3. M. J. H. Heule. *SmArT Solving: Tools and techniques for satisfiability solvers*. PhD thesis, Delft University of Technology, 2008.
4. M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *Proc. HVC'11*, 2011. To be published.
5. J. P. Marques-Silva, I. Lynce, and S. Malik. *Conflict-Driven Clause Learning SAT Solvers*, volume 185 of *FAIA*, chapter 4, pages 131–153. IOS Press, February 2009.
6. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *J. Artif. Intell. Res. (JAIR)*, 32:565–606, 2008.