MiniSAT_BCD and abcdSAT: Solvers Based on Blocked Clause Decomposition

Jingchao Chen
School of Informatics, Donghua University
2999 North Renmin Road, Songjiang District, Shanghai 201620, P. R. China

***Abstract*—MiniSAT_bcd and abcdSAT are submitted to SAT Race 2015. They both are based on blocked clause decomposition (BCD). Except for that the learnt clause database maintenance is different, their solving policy are the same.**

I. Introduction

MiniSAT_bcd and abcdSAT are a sequential CDCL solver, which are submitted to SAT+UNSAT track of SAT Race 2015. The core solving polcy of the two solvers are the same. Only there are differences in the learnt clause database maintenance.

II. A Solving Technique based Based on Blocked Clause Decomposition

Here we introduce a new solving technique, which is based on Blocked Clause Decomposition (BCD). The so-called blocked clause can be defined formally as follows. Given a CNF formula F, a clause C, a literal $l \in C$ is said to block C w.r.t. F if (i) C is a tautology w.r.t. l, or (ii) for each clause $C' \in F$ with $\bar{l} \in C'$, the resolvent $C' \otimes_l C$ is a tautology. When l blocks C w.r.t. F, the literal l and the clause C are called a blocking literal and a blocked clause, respectively.

In theory, any CNF formula can be decomposed two blocked subsets such that both can be solved by BCE (Blocked Clause Elimination). BCD has been applied widely to find backbone variables and implied binary equivalences through SAT sweeping. However, it is applied rarely to improve the performance of SAT solvers. As far as I know, Balyo et al. [7] are the first to use BCD to improve the performance of the state-of-the-art SAT solvers. Nevertheless, the conspicuous defect of their method is that it is required to add auxiliary variables. Its basic idea may be described as follows. Each variable x_i is allowed to have several versions. Let one blocked set be $C_1 \vee \cdots \vee C_m$. In the order of $C_m, \ldots, C_1$, we design its next version for the blocking literal for each clause. Assuming that each clause C has the form of $C = x_i \vee y_{j_1} \vee \cdots \vee y_{j_k}$, where x_i is the blocking literal. Let $x_{i,\$}$ be the current version of x_i. Its next version is defined as follows.

$$x_{i,\$+1} := x_{i,\$} \vee (y_{j_1,\$} \wedge \cdots \wedge y_{j_k,\$})$$

Then these formulas are re-encoded in CNF. Clearly, due to that a blocking literal is mapped to multiple versions, it will add the vast amount of auxiliary variables. Hence, no solver submitted to SAT competition 2014 used such a solving policy.

Apart from the solving policy of Balyo et al, we use directly the large blocked set to the performance of SAT solvers without re-encoding it. Therefore, our BCD-based solving policy need not map each blocking literal into multiple auxiliary variables Our basic idea is to use directly the large blocked to modify the pickBranchLit procedure of a CDCL solver. In details, we pick a decision literal of some level in the order of in the reverse order of blocked clauses. Let B_l be the blocked clause set of blocking literal l. Here is the pseudo-code of *pickBranchLit* algorithm.

```
**Algorithm** pickBranchLit
  **if** current level is from 1 to 3 **then**
        Let $l$ be decision literal at 0 level
        **for** each clause $C \in B_l$ picked in the reverse order **do**
              **if** $C$ is satisfied **then continue**
              pick a decision literal $l' \in C$
              **return** $l'$
  run the usual pickBranchLit
```

The drawback of our BCD-based solving policy is to only use the left (large) blocked set without using the right blocked set. So we hope to have the decomposition as unbalanced as possible. For this reason, we develop a fast blocked clause decomposition with high quality called *MixDecompose* [8], which is adopted by our solvers as a decomposition algorithm. Without BCD-based solving policy, no MiniSat-style solver solved korf-18. Using BCD-based solving policy, our MiniSat-style solver solved it.

III. The Other Solving Techniques

We made a slight modification on the Fourier-Motzkin elimination procedure of Lingeling [2] so that more variables in cardinality can be eliminated. For example, for the cardinality constraints:

$$\bar{x} + y + z \leq 2$$
$$x + z + w \leq 1$$

Applying Fourier-Motzkin procedure, eliminating x yields

$$y + 2z + w \leq 2$$

However, the solver Lingeling cannot eliminate x since it requires that the right constant must be less than or equal to one. Our solver can eliminate x, as long as it is less than 4.

Another solving technique our solvers use is hyper binary resolution, which may be defined as follows. Given a clause $l' \vee l_1 \vee \cdots \vee l_k$ and k binary clauses $l \vee \bar{l}_i$, where $1 \leq i \leq k$, we can infer the hyper binary resolvent $l \vee l'$ in one step. This inference is called Hyper Binary Resolution(HBR). The original version of MiniSat does not contain HBR, while the core procedure MiniSat in our two solvers does. Our HBR algorithm is very simple. It is neither tree-based lookahead HBR in [5], nor lazy HBR of Lingeling 587f [2]. Our HBR algorithm tests whether only each ternary clause becomes a core component of HBR in the order of clauses without using special data structure, whereas TreeLook and lazy HBR algorithm are based on literals, and test which literal yields a literal of hyper binary clauses. When the number of clauses is large, clause-based algorithms are slow. To save the computation cost, we restrict the the number of clauses traversed to 400000. In addition, from the second scan, at least two literals of a ternary clause are required to be in some new hyper binary clause. here is our simple HBR algorithm.

```
Algorithm simpleHBR($\mathcal{F}$)
  for each literal $l$ do $t[l] = 1$
  repeat
      for each ternary clause $x \vee y \vee z \in \mathcal{F}$ do
          if $t[\bar{x}] + t[\bar{y}] + t[\bar{z}] < 2$ then continue
          if $(l \vee \bar{y}) \wedge (l \vee \bar{z}) \in \mathcal{F}$ then $\mathcal{F} = \mathcal{F} \cup \{l \vee x\}$
      for each literal $l$ do $t[l] = 0$
      for each literal $l$ in new binary clause do $t[\bar{l}] = 1$
  untill no new clause is add to $\mathcal{F}$
```

HBR algorithm is used not only in our preprocessing, but also in in-processing.

IV. MINISAT_BCD AND ABCDSAT

The two solvers are the improved version of Glue_lgl_split [9]. They both are built on the top of Glucose 2.3 [1], and use Lingeling 587f [2] as preprocessor. The main difference between Glue_lgl_split and the two improved version is that MiniSAT_bcd and abcdSAT add the BCD-based solving technique, the extended Fourier-Motzkin elimination and hyper binary resolution, which are given above. The bit-encoding phase selection strategy [3] and split-merging technique used are the same as Glue_lgl_split. The notable difference between MiniSAT_bcd and abcdSAT is that their learnt clause database management is different. MiniSAT_bcd adopted the same learnt clause management technique as MiniSat_HACK_999ED [10]. Such a management technique classifies learnt clauses into core and local. Clauses with LBD(Literals Blocks Distance, for its definition, see [4]) less than or equal to 5, considered as core, are kept indefinitely unless removed by trivial simplification at decision level 0. Other clauses is classified as local. Its number is maintained roughly between 20,000 and 30,000. The clauses are maintained by the MiniSat-based clause activity scores. The learnt clause management of is the same as Glucose 2.3.

REFERENCES

[1] G. Audemard and L. Simon, "Glucose 2.3 in the sat 2013 competition," in *Proceedings of the SAT Competition 2013*, pp. 40–41. [Online]. Available: https://helda.helsinki.fi/handle/10138/40026

[2] A. Biere. Lingeling, plingeling and treengeling. [Online]. Available: http://fmv.jku.at/lingeling/

[3] J. Chen, "A bit-encoding phase selection strategy for satisfiability solvers," in *Proceedings of Theory and Applications of Models of Computation (TAMC'14)*, ser. LNCS, vol. 8402, Chennai, India, 2014, pp. 158–167.

[4] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *proceedings of IJCAI*, 2009, pp. 399–404.

[5] M. Heule, M. Järvisalo, A. Biere, "Revisiting Hyper Binary Resolution ," in *proceedings of CPAIOR*, 2013, pp. 77–93.

[6] A. Biere, D. Berre, E. Lonca, N. Manthey, "Detecting Cardinality Constraints in CNF," in *Proceedings of SAT 2014*, pp. 285–301.

[7] T. Balyo, A. Fröhlich, M. Heule, A. Biere, "Everything you always wanted to know about blocked sets (but were afraid to ask)," SAT 2014, LNCS 8561, pp. 317–332.

[8] J. Chen, "Fast blocked clause decomposition with high quality," submitted for publication.

[9] J. Chen, " Glue_lgl_split and GlueSplit_clasp with a Split and Merging Strategy," in *Proceedings of the SAT Competition 2014*, pp. 37–38. [Online]. Available: https://helda.helsinki.fi/handle/10138/135571

[10] Chanseok Oh, "MiniSat_HACK_999ED, MiniSat_HACK_1430ED and SWDiA5BY," in *Proceedings of the SAT Competition 2014*, pp. 46–47. [Online]. Available: https://helda.helsinki.fi/handle/10138/135571