Everything You Always Wanted to Know About Blocked Sets (But Were Afraid to Ask)

Tomáš Balyo, Andreas Fröhlich, Marijn J. H. Heule, Armin Biere

Charles University, Prague
Johannes Kepler University, Linz
The University of Texas, Austin

SAT 2014

Overview

Satisfiability

CNF formula

- A *Boolean variable* has two values: True and False
- A *literal* is Boolean variables or its negation
- A *clause* is a disjunction (or) of literals
- A *CNF formula* is a conjunction (and) of clauses

$$F = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee x_3) \wedge (\overline{x}_3)$$

Satisfiability

A CNF formula is *satisfiable* it has a satisfying assignment.
The problem of satisfiability (SAT) is to determine whether a given CNF formula is satisfiable.

Resolution

Resolution Rule

$$\frac{C_1 = (l \vee D_1) \quad C_2 = (\neg l \vee D_2)}{C_1 \otimes_l C_2 = (D_1 \vee D_2)}$$

By $C_1 \otimes_l C_2$ we denote the resolution of C_1 and C_2 on the literal l.

- The clauses D_1 and D_2 must not contain a pair of complementary literals ($C_1 \otimes_l C_2$ would be a tautology).

Resolution

Resolution Rule

$$\frac{C_1 = (l \vee D_1) \quad C_2 = (\neg l \vee D_2)}{C_1 \otimes_l C_2 = (D_1 \vee D_2)}$$

By $C_1 \otimes_l C_2$ we denote the resolution of C_1 and C_2 on the literal l.

- The clauses D_1 and D_2 must not contain a pair of complementary literals ($C_1 \otimes_l C_2$ would be a tautology).
- If C_1 or C_2 is a unit (one-literal) clause we call it a *unit resolution*.

Resolution

Resolution Rule

$$\frac{C_1 = (l \vee D_1) \quad C_2 = (\neg l \vee D_2)}{C_1 \otimes_l C_2 = (D_1 \vee D_2)}$$

By $C_1 \otimes_l C_2$ we denote the resolution of C_1 and C_2 on the literal l.

- The clauses D_1 and D_2 must not contain a pair of complementary literals ($C_1 \otimes_l C_2$ would be a tautology).
- If C_1 or C_2 is a unit (one-literal) clause we call it a *unit resolution*.

Unit Propagation

If a unit clause $C = (x)$ or $C = (\overline{x})$ is part of a formula F, then F is equivalent to $F_{x=1}$ or $F_{x=0}$, respectively, and can be replaced by it. This process is called *unit propagation*. By $UP(F)$ we denote the fixpoint obtained by iteratively performing unit propagation until no more unit clauses are part of the formula.

Blocked Sets

Literal blocks a clauses

A literal l blocks a clause C, $l \in C$ w.r.t. a CNF formula F if for each clause $C' \in F$ with $\bar{l} \in C'$, $C \otimes_l C'$ is a tautology.

Blocked Sets

Literal blocks a clauses

A literal l blocks a clause C, $l \in C$ w.r.t. a CNF formula F if for each clause $C' \in F$ with $\bar{l} \in C'$, $C \otimes_l C'$ is a tautology.

Blocked clauses

A clause C is blocked w.r.t. a CNF formula F if it is a tautology or it has a literal $l \in C$ that blocks it (blocking literal).

Blocked Sets

Literal blocks a clauses

A literal l blocks a clause C, $l \in C$ w.r.t. a CNF formula F if for each clause $C' \in F$ with $\bar{l} \in C'$, $C \otimes_l C'$ is a tautology.

Blocked clauses

A clause C is blocked w.r.t. a CNF formula F if it is a tautology or it has a literal $l \in C$ that blocks it (blocking literal).

Blocked Clause Elimination

Blocked clause elimination (BCE) is the process of removing blocked clauses from a formula F until fixpoint. The obtained formula is denoted by $BCE(F)$.

Blocked Sets

Literal blocks a clauses

A literal l blocks a clause C, $l \in C$ w.r.t. a CNF formula F if for each clause $C' \in F$ with $\bar{l} \in C', C \otimes_l C'$ is a tautology.

Blocked clauses

A clause C is blocked w.r.t. a CNF formula F if it is a tautology or it has a literal $l \in C$ that blocks it (blocking literal).

Blocked Clause Elimination

Blocked clause elimination (BCE) is the process of removing blocked clauses from a formula F until fixpoint. The obtained formula is denoted by $BCE(F)$.

Blocked Set

A CNF formula F is a blocked set if $BCE(F) = \emptyset$.

Blocked Sets

Example

$F = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee x_3) \wedge (\overline{x}_3)$

The first literal of each clause is the blocking literal. The clauses are removed from left to right.

Basic properties:

- Removing a blocked clause preserves satisfiability.
- Blocked sets are always satisfiable.
- We define the class of all blocked sets $\mathcal{BS} := \{F \mid BCE(F) = \emptyset\}$.
- If $G \subset F$ and C is blocked w.r.t. F, then C is blocked w.r.t. G
- If $F \in \mathcal{BS}$ and $G \subseteq F$ then $G \in \mathcal{BS}$.
- BCE is confluent, i.e., the order of blocked clause removal is not important
- Pure literal elimination is a special case of BCE.

$\mathcal{BS}$ not closed under (unit) resolution

Proposition

Blocked sets are not closed under resolution, not even unit resolution.

Proof: This is a blocked set.

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee x_3) \wedge (\overline{x}_3)$$

This is a unit resolvent.

$$(x_1 \vee x_2) = (x_1 \vee x_2 \vee x_3) \otimes (\overline{x}_3)$$

This is not a blocked set.

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee x_3) \wedge (\overline{x}_3) \wedge (x_1 \vee x_2)$$

□

$\mathcal{BS}$ not closed under partial assignments

Proposition

Blocked sets are not closed under partially assigning variables, furthermore, a blocked set may become non-blocked even in the subspace where this formula remains satisfiable.

Proof:

$$F = (\overline{x}_3 \vee \overline{x}_1 \vee x_4) \wedge (x_3 \vee x_2 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_1) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge (\overline{x}_5 \vee \overline{x}_4)$$

This formula F is a blocked set, however, neither $F_{x_3=0}$ nor $F_{x_3=1}$ is a blocked set.

$$\begin{aligned} F_{x_3=0} &= (x_2 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_1) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge (\overline{x}_5 \vee \overline{x}_4) \\ F_{x_3=1} &= (\overline{x}_1 \vee x_4) \wedge (\overline{x}_2 \vee \overline{x}_1) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge (\overline{x}_5 \vee \overline{x}_4) \end{aligned}$$

$\mathcal{BS}$ closed under unit propagation

Proposition

Blocked sets are closed under unit propagation ($F \in \mathcal{BS} \rightarrow UP(F) \in \mathcal{BS}$).

Proof (sketch):

- The clauses in $UP(F)$ can be eliminated in the same order as in F.
- Examine all possible (3) cases.

Solving Blocked Sets

Solve (Blocked set B)

$\beta := [*, *, \ldots, *]$
for Clause $C \in$ reverse($eliminationStack$) **do**
  if C is satisfied under β **then** continue
  $V :=$ getUnassignedVars(C)
  if $V = \emptyset$ **then** flip the blocking literal of C in β
  else
    select $x \in V$
    set x in β to a value that satisfies C
return β

Solving Blocked Sets

Solve (Blocked set B)

```
β := [*, *, ..., *]
for Clause C ∈ reverse(eliminationStack) do
    if C is satisfied under β then continue
    V := getUnassignedVars(C)
    if V = ∅ then flip the blocking literal of C in β
    else
        select x ∈ V
        set x in β to a value that satisfies C
return β
```

Example: $(x_1 \vee x_2) \wedge (x_3 \vee \overline{x_2}) \wedge (x_3 \vee \overline{x_2} \vee \overline{x_1})$

- start with [*,*,*]

Solving Blocked Sets

Solve (Blocked set B)
    $\beta := [*, *, \ldots, *]$
    for Clause $C \in$ reverse($eliminationStack$) **do**
        if C is satisfied under β **then** continue
        $V :=$ getUnassignedVars(C)
        if $V = \emptyset$ **then** flip the blocking literal of C in β
        else
            select $x \in V$
            set x in β to a value that satisfies C
    return β

Example: $(x_1 \vee x_2) \wedge (x_3 \vee \overline{x_2}) \wedge (x_3 \vee \overline{x_2} \vee \overline{x_1})$

- start with [*,*,*]
- $(x_3 \vee \overline{x_2} \vee \overline{x_1})$ is not satisfied, $V = \{x_1, x_2, x_3\}$, select x_1, [F,*,*]

Solving Blocked Sets

Solve (Blocked set B)

$\beta := [*, *, \ldots, *]$
for Clause $C \in$ reverse($eliminationStack$) **do**
  if C is satisfied under β **then** continue
  $V :=$ getUnassignedVars(C)
  if $V = \emptyset$ **then** flip the blocking literal of C in β
  else
    select $x \in V$
    set x in β to a value that satisfies C
return β

Example: $(x_1 \vee x_2) \wedge (x_3 \vee \overline{x_2}) \wedge (x_3 \vee \overline{x_2} \vee \overline{x_1})$

- start with [*,*,*]
- $(x_3 \vee \overline{x_2} \vee \overline{x_1})$ is not satisfied, $V = \{x_1, x_2, x_3\}$, select x_1, [F,*,*]
- $(x_3 \vee \overline{x_2})$ is not satisfied, $V = \{x_2, x_3\}$ select x_2, [F,F,*]

Solving Blocked Sets

```
Solve (Blocked set B)
  β := [*, *, ..., *]
  for Clause C ∈ reverse(eliminationStack) do
    if C is satisfied under β then continue
    V := getUnassignedVars(C)
    if V = ∅ then flip the blocking literal of C in β
    else
      select x ∈ V
      set x in β to a value that satisfies C
  return β
```

Example: $(x_1 \vee x_2) \wedge (x_3 \vee \overline{x_2}) \wedge (x_3 \vee \overline{x_2} \vee \overline{x_1})$

- start with [*,*,*]
- $(x_3 \vee \overline{x_2} \vee \overline{x_1})$ is not satisfied, $V = \{x_1, x_2, x_3\}$, select x_1, [F,*,*]
- $(x_3 \vee \overline{x_2})$ is not satisfied, $V = \{x_2, x_3\}$ select x_2, [F,F,*]
- $(x_1 \vee x_2)$ is not satisfied, $V = \emptyset$ flip x_1, [T,F,*]

Solving Blocked Sets

Solve (Blocked set B)

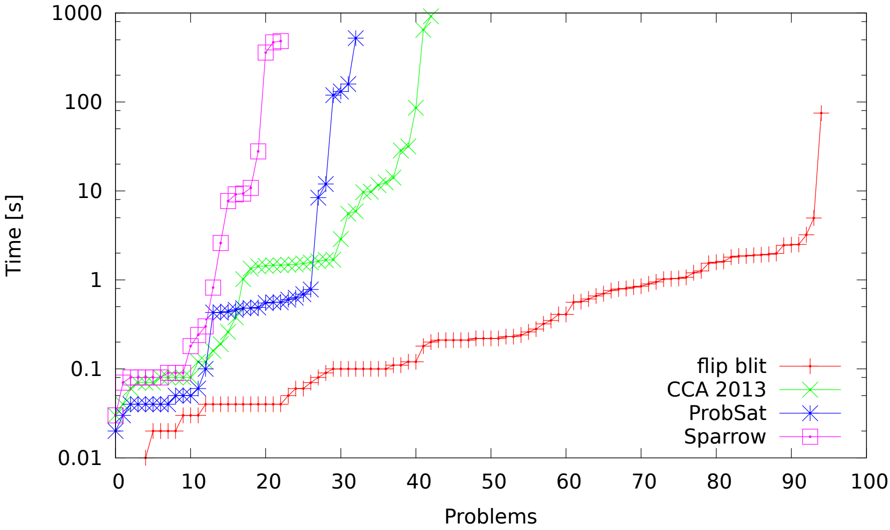
S1 $\beta := [*, *, \dots, *]$
S2 **for** Clause $C \in$ reverse($eliminationStack$) **do**
S3 **if** C is satisfied under β **then** continue
S4 $V :=$ getUnassignedVars(C)
S5 **if** $V = \emptyset$ **then** flip the blocking literal of C in β
S6 **else**
S7 select $x \in V$
S8 set x in β to a value that satisfies C
S9 **return** β

Properties:

- The algorithm can find partial satisfying assignments.
- It runs in linear time.
- The algorithm is non-deterministic (select on line S7).
- All the solutions can be found (by doing proper selections).

Smells Like Local Search

What if WalkSat always flipped the blocking literal?



Getting Blocked Sets

- Are blocked sets commonly found among application formulas?

Getting Blocked Sets

- Are blocked sets commonly found among application formulas? NO.

Getting Blocked Sets

- Are blocked sets commonly found among application formulas? NO.
- ... among competition benchmarks?

Getting Blocked Sets

- Are blocked sets commonly found among application formulas? NO.
- ... among competition benchmarks? NO.

Getting Blocked Sets

- Are blocked sets commonly found among application formulas? NO.
- ... among competition benchmarks? NO.

Where can we get some blocked sets?

Getting Blocked Sets

- Are blocked sets commonly found among application formulas? NO.
- ... among competition benchmarks? NO.

Where can we get some blocked sets?

Blocked Clause Decomposition

The set of clauses of any CNF formula F can be split into two blocked sets L and R. This process is called *Blocked Clause Decomposition* (BCD).

$$\mathrm{BCD}(F) = (L, R);\ F = L \cup R;\ L, R \in \mathcal{BS};\ |L| \geq |R|$$

- The larger of the two blocked sets is called L (large) and the other is R (rest). A BCD is better if L is much bigger than R.

Pure Decomposition

Pure decomposition is a simple linear BCD algorithm.

1. let $L = R = \emptyset$
2. select any $x \in Vars(F)$
3. let $F_x \subseteq F$ ($F_{\overline{x}} \subseteq F$) be the clauses of F that contain x ($\overline{x}$).
4. if $|F_x| > |F_{\overline{x}}|$ then add F_x into L and $F_{\overline{x}}$ to R otherwise add F_x into R and $F_{\overline{x}}$ to L.
5. remove F_x and $F_{\overline{x}}$ from F.
6. if $F \neq \emptyset$ then goto step 2.

- Produces blocked sets that can be solved by pure literal elimination.
- Runs in linear time. Quality of the decomposition is not great.

Improved Unit Decomposition

Two incomplete algorithms for BCD:

Unit Decomposition

Test if the non-unit clauses of F are a blocked set. If so, $L =$ non-unit clauses of F, $R =$ unit clauses of F.

Works on 77 of the 300 instances of the application track of the 2013 SAT Competition.

Improved Unit Decomposition

Two incomplete algorithms for BCD:

Unit Decomposition

Test if the non-unit clauses of F are a blocked set. If so, $L =$ non-unit clauses of F, $R =$ unit clauses of F.

Works on 77 of the 300 instances of the application track of the 2013 SAT Competition.

Improved Unit Decomposition

Run unit propagation on F without removing unsat literals from clauses. Test if the remaining clauses of F are a blocked set. If so, $L =$ remaining clauses of F, $R =$ clauses removed by UP.

Works on 95 of the 300 instances of the application track of the 2013 SAT Competition.

Improved Unit Decomposition

Two incomplete algorithms for BCD:

Unit Decomposition

Test if the non-unit clauses of F are a blocked set. If so, $L =$ non-unit clauses of F, $R =$ unit clauses of F.

Works on 77 of the 300 instances of the application track of the 2013 SAT Competition.

Improved Unit Decomposition

Run unit propagation on F without removing unsat literals from clauses. Test if the remaining clauses of F are a blocked set. If so, $L =$ remaining clauses of F, $R =$ clauses removed by UP.

Works on 95 of the 300 instances of the application track of the 2013 SAT Competition.

If these algorithms fail run pure decomposition.

BCD Post-processing

After running pure decomposition or (improved) unit decomposition some clauses can be moved to L from R to improve BCD quality.

- Clauses from R that are blocked w.r.t. L can be moved immediately.
- So called ‘blockable’ clauses can be also moved.
- For any set $S \subset R$ we can try if $BCE(L \cup S) = \emptyset$ and move it. However, this is costly.

L is called a *maximal blocking set* if none of the clauses in R can be moved to L.

Solitaire Decomposition

Solitaire Decomposition

A special kind of BCD where the the small blocked set R contains only a single unit clause is called a *Solitaire decomposition*.
Solitaire decomposition can be always achieved if we add a new variable y to each clause of F and a unit clause $(\overline{y})$.

$$C_1 \wedge \cdots \wedge C_m \rightarrow (C_1 \vee y) \wedge \cdots \wedge (C_m \vee y) \wedge (\overline{y})$$

Then $R = (\overline{y})$ and $L = (C_1 \vee y) \wedge \cdots \wedge (C_m \vee y)$.

It is better to first do BCD and then add y only to the clauses in R and move them into L.

Reencoding

Instead of implementing the algorithm for solving blocked sets we encode its progression on a given BS into SAT.

- For each original variable $x_1, \ldots, x_n$ of F, we will have several new variables called *versions*. By $x_{i,k}$, we denote the k-th version of x_i ($x_{i,0}$ is x_i) and by $x_{i,\$}$, the latest version of x_i
- For each clause $C = (x_i \vee y_{j_1} \vee \cdots \vee y_{j_k})$, where x_i is the blocking literal, we create a new version of x_i. The value of the new version is given by the following definition.

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

Example: $(\overline{x}_1 \vee x_3 \vee x_2) \wedge (x_3 \vee x_4 \vee \overline{x}_1) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3)$.

$$\begin{aligned} x_{1,1} &:= \quad x_{1,0} \vee (x_{2,0} \wedge x_{3,0}) \\ x_{3,1} &:= \quad x_{3,0} \vee (\overline{x}_{4,0} \wedge x_{1,1}) \\ \overline{x}_{1,2} &:= \quad \overline{x}_{1,1} \vee (x_{3,1} \wedge x_{2,0}) \end{aligned}$$

Reencoding

The equations $x_{i,\$+1} := x_{i,\$} \vee (\overline{y}_{j_1,\$} \wedge \cdots \wedge \overline{y}_{j_k,\$})$ are then:

- Directly encoded into CNF (using $k+2$ clauses for each):

$$(\overline{x}_{i,\$} \vee x_{i,\$+1}) \wedge (x_{i,\$+1} \vee y_{j_1,\$} \vee \cdots \vee y_{j_k,\$}) \wedge \bigwedge_{l=1}^{k} (\overline{y}_{j_l,\$} \vee \overline{x}_{i,\$+1} \vee x_{i,\$})$$

  The final versions of the variables $(x_{i,\$})$ are then substituted in the clauses of the small blocked set.

OR

- Expressed as an And-Inverter-Graph (AIG) circuit, then simplified using circuit simplification techniques (dc2 of ABC) and finally the simplified AIG is translated into CNF.
  We use solitaire decomposition and the unit clause of the small blocked set represents the output of the AIG.

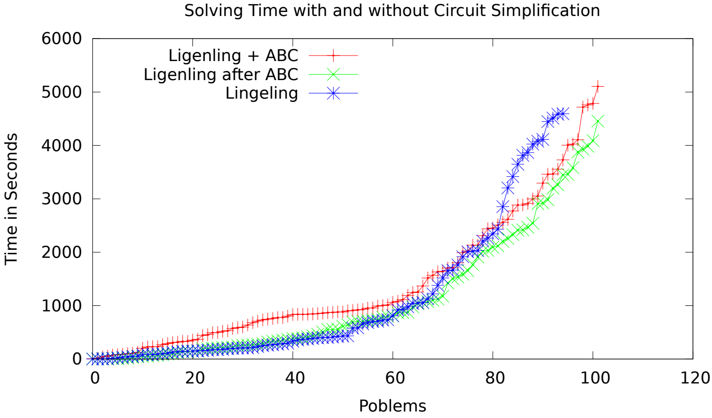
Experiments

- We evaluated the proposed methods on the 300 instances from the application track of the SAT Competition 2013.
- We used a 32-node cluster with Intel Core 2 Quad (Q9550 @2.83GHz) processors and 8GB of memory, time limit of 5000 seconds and a memory limit of 7000MB.
- We compared the performance of Lingeling with and without reencoding.

Results for the direct CNF encoding:

- With reencoding we can solve 8 more formulas (240/300).
- Useful only for hard instances (solved longer than 1 hour).
- Better BCD gives better runtimes.

Experiments – AIG Method



The time required to solve the benchmark formulas, which can be decomposed with a quality of at least 90% (135 of 300 instances). Reencoding helps to solve 7 more formulas.

Conclusion

- We showed some structural properties of blocked sets.
- We introduced new algorithms for solving blocked sets and decomposing formulas into blocked sets.
- We showed that BCD can be used to reencode formulas and improve the performance of SAT solving on hard instances.

Future work:

- Is it possible to enumerate all the solutions of a blocked set?
- Can blocked sets be used to improve local search SAT solvers?
- Other usages of blocked sets?
- More efficient BCD.

Thank You