Solving ALLSAT with Backbone Information

Abstract

The ALLSAT (ALL-SATisfiable) problem focus on finding all satisfiable assignments of a given propositional formula, whose applications include model checking, automata translation and the test-case generation. Traditional approaches to solve such problem requires a large amount of SAT calls. In this paper, we introduce BASOLVER, a backbone-based ALLSAT solver for propositional formulas. Compared to extant works, BASOLVER utilizes backbone information to obtain shorter partial assignments. As a shorter partial assignment potentially represents a set of more full assignments, the backbone information is able to accelerate the ALLSAT computation. Experiments show that even finding backbone variables pays additional cost, BASOLVER performs better than extant tools including MBLOCKING, BC, NBC and BDD. Given the same time and memory resources, BASOLVER solves the largest amount of benchmarks and performs the fastest on the formulas that can be solved by all 5 tools.

1 Introduction

The SAT (SATisfiable) applications, such as model checking [Biere *et al.*, 2003; Bradley, 2012], program analysis [Cadar *et al.*, 2008; Beyer and Keremoglu, 2011; Kroening and Tautschnig, 2014], network verification [Lopes *et al.*, 2015; Majumdar *et al.*, 2014; Zhang *et al.*, 2012], quantifier elimination [Brauer *et al.*, 2011] and predicate abstraction [Lahiri *et al.*, 2003], have gained considerable attentions due to the increasing capability of modern SAT solvers. Also, among several applications like unbounded hardware model checking [Li *et al.*, 2017], logic minimization [Sapra *et al.*, 2003], and temporal logic planning [Li *et al.*, 2019], computing all satisfiable assignments of the model is demanded. This paper focuses on the ALLSAT (ALL-SATisfiable) problem, which aims to compute all satisfiable assignment of a propositional formula, and presents an efficient solver BASOLVER to help tackle such problem.

Although the satisfiability of propositional formulas is known as a NP-complete problem, modern SAT solvers are able to find a satisfiable assignment within affordable time. However, directly enumerating all satisfiable assignments of a formula may be infeasible because the number of such assignments can be exponential to the size of the formula. As a result, solving the ALLSAT problem remains as a challenging task. As far as we know, there are mainly two kinds of solutions to ALLSAT, i.e., the blocking-based [McMillan, 2002] and non-blocking-based [Grumberg *et al.*, 2004]. The blocking-based strategy uses the SAT solver to find a satisfiable assignment, and then blocks such assignment in the original formula so as to avoid finding the same solution repeatedly. Blocking partial assignments instead of the full ones is a common optimization used in the blocking-based solution. The motivation comes from that the SAT solver normally is good at enumerating partial assignments while not at the full ones. Meanwhile, the non-blocking solution backtracks the decision diagram inside the SAT solver to find every satisfiable assignment of the given formula. Once a (full) satisfiable assignment is found by the SAT solver, the non-blocking approach backtracks to a previous decision level with some specific strategies and flip the value of the decision variable at that level to generate a new assignment. The ALLSAT computation is complete as soon as every branch value of variables in the decision diagram is visited. The advantages of the non-blocking strategy are to avoid the changes of the formula and utilizes the full information of the decision diagrams in the SAT solver, while the blocking-based one is more straightforward and the implementation does not require changes to the SAT solver.

The ALLSAT solver BASOLVER we introduced is a blocking based solver, we use Minisat v2.1.1 [Eén and Sörensson, 2003] as the underlying SAT solver and implemented BASOLVER with C++. The main innovation of BASOLVER is that it uses the backbone information to reduce the length of the partial solutions. Every backbone variable is removed from the partial solution and added to the formula as a unit clause. In this way, BASOLVER gets much shorter partial solutions without changing the correctness of the partial solutions. The number of SAT solving in the finding of every solution in BASOLVER is reduced since every single shorter partial solution represents more full solutions. Comparing to the blocking based ALLSAT solvers with different coverage techniques [Jin *et al.*, 2005] [Morgado and Marques-Silva, 2005], BASOLVER generates much shorter partial solutions for most of the formulas. Although finding backbone variables requires additional SAT solving, BASOLVER is still faster than the state-of-the-art tool [Yu *et al.*, 2014] (MBLOCKING).

We compared BASOLVER with 4 other tools, including 2 blocking based tools, MBLOCKING [Yu *et al.*, 2014] and BC, one non-blocking based tool , NBC and one BDD-based tool, BDD [Toda and Soh, 2016]. Among the 5 tools, BASOLVER solves (finds every solution of the given formulas) the most formulas (79) within the given time and memory limitation, while the number of solved formulas for MBLOCKING, BC, NBC and BDD are 65, 64, 53 and 50 respectively. For the formulas that are solved by both BASOLVER and MBLOCKING, BASOLVER uses 23% less computing time than MBLOCKING does. BASOLVER uses 37%, 68% and 31% less computing time than BC, NBC and BDD does respectively for the formulas that are mutually solved by BASOLVER and one of the comparing tools. BASOLVER also gets the shortest blocking clauses among the three blocking based tools, the average length of blocking clauses for BASOLVER is 1026, which is 20 times less than MBLOCKING (22182) and 84% less than BC (6180).

The remainder of the paper is organized as follows. We describe notations and preliminaries in Section 2. The algorithms of BASOLVER are discussed in Section 3, experimental results are shown in Section 4 and related work are discussed in Section 5. We conclude the paper in Section 6.

2 Preliminaries

Let X be a finite set of Boolean variables. For every Boolean variable $x \in X$, x can only be assigned to 0 or 1, otherwise the value of x is NOT_KNOWN. A literal l is either a Boolean variable x or its negation $\neg x$. For a literal l, the corresponding variable of l is $x(l)$. A clause c is a disjunction of literals, a literal l is in a clause c ($l \in c$) if and only if l appears in the disjunction that compose the clause c. A variable x is in a clause c if and only if literal x or literal $\neg x$ is in the clause c. A SAT formula F is a conjunction of clauses, a clause c is in the formula F ($c \in F$) if and only if c appears in the conjunction that compose the formula F. A variable x is in a formula F if and only if there exists a clause $c \in F$ such that $x \in c$ or $\neg x \in c$.

An *assignment* a of a given SAT formula F is a function that maps each variable $x \in F$ to 0, 1, or NOT_KNOWN. For example, an assignment $a : \{x_1, x_2, ..., x_k\} \mapsto \{1, 1, ..., 0\}$ is an assignment of the formula F, where k is the number of Boolean variables in F. For a variable $x \in F$, the value of x in the assignment a is $a(x)$, the value of a clause $c \in F$ in the assignment a is $a(c)$, and the value of F in the assignment a is $a(F)$. $\neg a(x) = 1$ if $a(x) = 0$ and $\neg a(x) = 0$ if $a(x) = 1$. For an assignment a if there does not exist a variable $x \in F$, such that $a(x)$=NOT_KNOWN, then a is a full assignment of the given formula F. An assignment p is a partial assignment if there exists at least one variable x such that $p(x)$=NOT_KNOWN.

For a given assignment v, v is a *solution* of the given formula F if and only if the value of F in the assignment v is 1, i.e., $v \models F$ if and only if $v(F) = 1$. If v is a full assignment of F then v is also a full solution of F. v is a partial solution of F if v is a partial assignment. The solutions are written as the conjunctions of literals for short, for example, for a solution v such that $v(a) = 1$ and $v(b) = 0$, then v is written as $v = a \wedge \neg b$.

For a given partial solution v, v is able to represent at least two different full solutions v_1 and v_2, for every variable x such that $v(x) \neq$NOT_KNOWN, $v_1(x) = v_2(x) = v(x)$, and for every variable x' such that $v(x')$ = NOT_KNOWN, $v_1(x') \neq v_2(x')$ and $v_1(x') \neq$ NOT_KNOWN, $v_2(x') \neq$NOT_KNOWN.

For example, given a formula $F = (a \vee b) \wedge (a \vee \neg b)$, $v = a$ is partial solution of F and $v_1 = a \wedge b$, $v_2 = a \wedge \neg b$ are two different full solution represents by v.

Definition 1 (Backbone Variable). *For a given satisfiable formula F, and a variable $x \in F$, x is a backbone variable if $x \wedge F$ or $\neg x \wedge F$ is unsatisfiable.*

Lemma 1 (Backbone Assignment). *For a backbone variable x of a given formula F, the value of x must be always assigned to 1 or 0 in all solutions.*

Definition 2 (Essential Literals). *For a given formula F, a literal $x \in F$ and a full solution $v \models F$, x is an essential literal of F with v, if and only if there exists a clause $c \in F$ such that $v(x) = 1, x \in c$ and for every other literal $x' \in c, x \neq x'$, $v(x') = 0$.*

An essential variable x is the only reason that make a clause c satisfiable in a formula F with the given solution. For an essential variable x with solution v, the assignment a is not a solution of F such that for every variable $x' \neq x$, $a(x') = v(x')$ and $a(x) = \neg v(x)$.

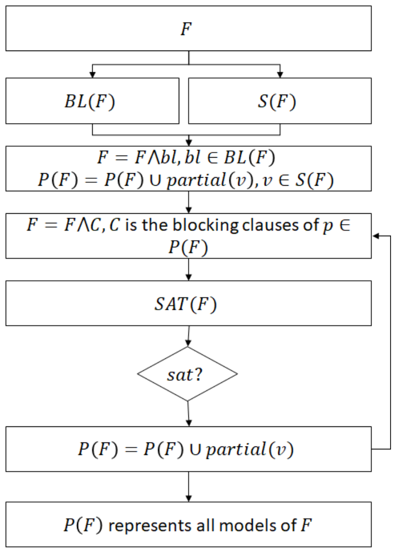
For example, given a formula $F = (a \vee b) \wedge (a \vee \neg b)$, a is a backbone variable since $\neg a \wedge F$ is unsatisfiable. There are only two solutions of F, $v_1 = a \wedge b$, and $v_2 = a \wedge \neg b$, the assignment of backbone variable a is always 1 in every solution of F. For the solution v_1, the literal a is an essential literal since for the clause $a \vee \neg b$, $v_1(a) = 1$ and for every other literal (the literal $\neg b$) in $a \vee \neg b$, $v_1(\neg b) = 0$.

3 ALLSAT Solver Using Backbone Information

We propose and implement an ALLSAT solver BASOLVER, BASOLVER uses backbone information to get shorter partial solutions comparing to other tools. With the help of shorter partial solutions, the number of SAT solving in BASOLVER reduced and the efficiency increased.

3.1 Overview of Algorithms

Figure 1 shows the workflow of BASOLVER. Given a formula F, BASOLVER first computes the backbone variables of F using tools from [Zhang *et al.*, 2018]. For every backbone variable bl of F, $bl \in BL(F)$, and bl is added to F as a unit clause. During the computing of backbone variables, solutions are also generated and stored in $S(F)$. The partial solution of each solution $v \in S(F)$ is stored in $P(F)$. BASOLVER then computes the partial solution $p = partial(v)$ and the blocking clause C for every solution in $S(F)$, and adds the partial solutions to $P(F)$. All backbone variables are removed from the partial solution p of a solution v, and



F
$BL(F)$
$S(F)$
$F = F \wedge bl, bl \in BL(F)$
$P(F) = P(F) \cup partial(v), v \in S(F)$
$F = F \wedge C$, C is the blocking clauses of $p \in P(F)$
$SAT(F)$
$sat?$
$P(F) = P(F) \cup partial(v)$
$P(F)$ represents all models of F

Figure 1: Workflow of BASOLVER

every essential but non-backbone literal remains in p. If every clause is satisfied with the current p, i.e., $\forall c \in F, p(c) = 1$, the generation of p completes, otherwise, more literals are added to p based on a greedy strategy. The blocking clause C of a partial solution p is the negation of p. For example, if $p = a \wedge b \wedge c$, then the blocking clause C of p is $C = \neg a \vee \neg b \vee \neg c$. After generating the blocking clause C, F is updated with the conjunction between F and C, i.e. $F = F \wedge C$. If $F \wedge C$ is unsatisfiable, then BASOLVER has found every solution of the formula F. Otherwise, a new solution v' is returned by the SAT solver, BASOLVER then computes the partial solution p' and the blocking clause C' again and updates the formula F with $F \wedge C'$. The updating of the formula F continues until F is unsatisfiable. BASOLVER then adds every backbone variable in $BL(F)$ to every partial solution, and every solution of the given formula F is represented in at least one of the partial solution.

3.2 Algorithms of BASOLVER

Algorithm 1 shows the main algorithm of BASOLVER, where backbone(F) uses the algorithms of backbone computing in [Zhang *et al.*, 2018]. The detail of blocking(F) is showed in Algorithm 2. In Algorithm 1, at Line 1, the set of partial solutions and blocking clauses are initialized with $\emptyset$. The backbone variables of F is computed, stored in $BL(F)$ and added to F as unit clauses. The solutions obtained from the backbone computing are stored in $S(F)$. For every solution $v \in S(F)$, the partial solution p and the blocking clause C are computed, and stored in $P(F)$ and $BC(F)$ respectively. At Line 3, the blocking clauses in $BC(F)$ are added to F, ret is assigned with the satisfiability of F, returning by the Minisat Solver. If F is satisfiable, then v is a new solution of F, the partial solution p and the blocking clause C of the new solution v is computed at Line 4, and F is updated with $F \wedge C$ again. The while loop at Line 3 keeps if F is satisfiable, otherwise BASOLVER finishes the computing of every solution for the input formula, which is represented by at least one of the partial solutions in $P(F)$.

Algorithm 1 Main Algorithm of BASOLVER

```
Input  : A formula $F$
Output: Every satisfiable assignments of $F$
1 $P(F) := \emptyset$  $BC(F) := \emptyset$  $BL(F), S(F) := backbone(F)$
  $F := F \bigwedge bl, bl \in BL(F)$  foreach $v \in S(F)$ do
2     $p, C := blocking(v)$  $P(F) := P(F) \cup p$  $BC(F) :=$
      $BC(F) \cup C$
3 $F := F \bigwedge C, C \in BC(F)$  $(ret, v) := SAT(F)$  while $ret$
  do
4     $p, C := blocking(v)$  $P(F) := P(F) \cup p$  $BC(F) :=$
      $BC(F) \cup C$  $F := F \wedge C$  $(ret, v) := SAT(F)$
5 return $P(F)$
```

In Algorithm 2, the partial solutions and the blocking clauses are computed. The input of Algorithm 2 is a formula F, the backbone variables $BL(F)$ of the given formula F and a full solution v returned by the Minisat Solver. The partial solution p and the blocking clause C of v are initialized as $\emptyset$ at Line 6. For every literal l in v such that $v(l) = 1$, the algorithm first checks that if $x(l)$ is a backbone variable at Line 8. If $x(l)$ is a backbone variable, then l is not added to p at this step. Otherwise, l is added to p if there exists a clause c such that l is an essential variable with the given solution v (l is the only literal in c assigns to 1 in v, from Line 10 to Line 13). $\neg l$ is added to the blocking clause C if l is added to the partial solution p. Starting from Line 15, the algorithm checks if every clause $c \in F$ is satisfied by the p, i.e., for every clause $c \in F$, there exists at least one literal $l \in c$, and $p(l) = 1$. If a clause c is not satisfied by the partial solution p, and there does not exist a backbone variable $bl \in c$, $v(bl) = 1$, then a literal $l \in c, v(l) = 1$ is added to p, and the literal $\neg l$ is added to C. In order to make p a partial solution (instead of a partial assignment) of v, all backbone variables are added to p at Line 18, but the blocking clauses remain the same since the backbone variables are already added to F as unit clauses. After executing the loop starting from Line 15 and the adding of backbone variables at Line 18, every clause in F is satisfied by the partial solution p, p and C are then returned by Algorithm 2.

Theorem 1. *For a backbone variable bl of a given formula F, if a partial solution $p \models F$, then bl in p and $p(bl) = 1$.*

Proof. Given a formula F and a backbone variable bl of F, suppose there exists a partial solution p such that $p(bl) =$ NOT_KNOWN. Then there must exist two different solutions of F, such that for every variable $x \neq bl$, $v_1(x) = v_2(x) = v(x)$, and $v_1(bl) = \neg v_2(bl) = 1$ or $v_1(bl) = \neg v_2(bl) = 0$. In both cases, bl is not a backbone variable of F which contradicts with the assumptions that bl is a variable of F. $\square$

According to Theorem 1, there does not exist a partial solution p such that a backbone variable bl is not in p. Therefore, it is correct and essential to add all backbone variables back to every partial solution.

Algorithm 2 Computing the Partial Solutions and the Blocking Clauses

```
Input : A formula $F$, a full solution $v \models F$ returned by
        Minisat Solver, the backbone variables $BL(F)$ of $F$
Output: the partial solution $p$ and the blocking clauses $C$ of $v$
$p := \emptyset$  $C := \emptyset$  foreach $l \in v$ do
    if $x(l) \in BL(F)$ then
        continue
    if $\exists c \in F, l \in c$ then
        unique:=true  foreach $l' \neq l, l' \in c$ do
            if $v(l') =$ then
                unique:=false  break
        if unique then
            $p := l \wedge l$  $C := C \vee \neg l$
foreach $c \in F, (\nexists bl \in c, bl \in BL(F)) \wedge (\nexists l \in p, l \in c)$ do
    foreach $l' \in c$ do
        $p := l' \wedge l'$  $C := C \vee \neg l'$  break
foreach $bl \in BL(F)$ do
    $p := p \wedge bl$
return $p, C$
```

The main contribution of BASOLVER is to use backbone variables in the computing of partial solutions and the blocking clauses. As proved in Theorem 1, every backbone variable must appear in every partial solution. Therefore, BASOLVER removes the backbone variables and generates blocking clauses without them in it. To simplify the following SAT solving, backbone variables are used as initial assumptions by adding them back as unit clauses. With shorter blocking clauses, the complexity of the following SAT solving decreases. BASOLVER is more efficient when the number of backbone variables in the formulas are larger.

With the help of backbone variables, for some of the formulas, ALLSAT computing terminates without any blocking clauses. For a given formula F, whenever every clause $c \in F$ is satisfied by at least one backbone literal of F, BASOLVER terminates. Since the set of backbones is the only one partial solution that represents every full solution of F.

Given a formula $F = (a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d)$. Table 1 shows the ALLSAT computing procedure of F. BASOLVER first computes the backbone variables of F, a is the backbone variable of F, and the solution found during the backbone computing is $v = a \wedge b \wedge c \wedge d$. The partial solution p and the blocking clause C returned by Algorithm 2 are $p = c$ and $C = \neg c$, respectively. Then F is updated with the backbone variables and the blocking clauses, the backbone variable a is added to F as unit clause, and the blocking clause $\neg c$ is also added to F. The updated F is $F = (a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge a \wedge \neg c$. A new solution $v' = a \wedge b \wedge \neg c \wedge \neg d$ for the formula F is computed. $p = \neg d$ and $C = d$ are returned by Algorithm 2. Then F is updated with $F = (a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge a \wedge \neg c \wedge d$. F is unsatisfiable, and the ALLSAT computing of $F = (a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d)$ finished. The solutions of $F = (a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d)$ are represented with either $a \wedge \neg c$ or $a \wedge d$.

4 Evaluation

We implemented BASOLVER based on Minisat v2.1.1, using C++. BASOLVER, MBLOCKING and BC use the Minisat Solver as an oracle while NBC and BDD change the implementation of Minisat according to the algorithms.

The experiments are conducted on clusters with 64GB memory limitation and 10 hours runtime limitation. The operating systems are Red-Hat Enterprise Linux. We can not guarantee that the performances of CPUs in the cluster are exactly the same, thus all experiments are repeated 3 times and the average results are computed to enhance fairness. BASOLVER runs sequentially in the experiments and does not use the incremental features in the Minisat Solver.

We use formulas from industrial tracks of SAT competitions from 2011 to 2017 as benchmarks. There are 1816 formulas in total, 552 of them are known as unsatisfiable, 608 of them are satisfiable and the satisfiability of the rest remain unknown. Therefore, the size of the final benchmark is 608. The origin of these formulas includes encryption problems, planning problems, hardware model checking problems and fault localization problems, etc.

4.1 Overall Performance of the Tools

Table 2 shows the overall performance of the tools. The first column of Table 2 is the name and the number of the formulas. For clarification, we only include the formulas that at least one of the tools solves them. The second column is the average number of variables, and the third column is the average number of clauses of the formulas. The forth column is the average number of solutions in the formulas. Starting from the fifth column, the number of formulas solve by BASOLVER, MBLOCKING, BC, NBC and BDD are listed respectively. There are 79 formulas that are solved by at least one of the tools. BASOLVER solves all 79 formulas, MBLOCKING solves 65 formulas, BC solves 64 formulas, NBC solves 53 formulas and BDD solves 51 formulas. For the other 4 comparing tools (MBLOCKING, BC, NBC and BDD), there exists at least one group of formulas that the tools only solve a small part of them. For example, NBC only solves 1 formula in the Dimacs group, and MBLOCKING failed to solve a formula in the Complete group. BASOLVER solves the most formulas among the 5 tools, which indicates that the efficiency of BASOLVER is better than the comparing tools with the given benchmark.

For the 31 formulas that are solved by all the 5 tools, BASOLVER uses the least computing time (9558.64 seconds), which is 51% less than MBLOCKING (19427.51 seconds), 48% less than BC (18221,86 seconds), 54% less than NBC (21263.27 seconds) and 67% less than BDD (28901.35 seconds). Since only BASOLVER solves all the 79 formulas, we only compare the computing time among the formulas that are solved by both BASOLVER and one of the comparing tools (including MBLOCKING, BC, NBC, and BDD).

Formula	Full Solution	Backbone Variables	Partial Solution	Blocking Clauses
$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d)$	$a \wedge b \wedge c \wedge d$	a	c	$\neg c$
$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge a \wedge \neg c$	$a \wedge b \wedge c \wedge d$	a	$\neg d$	d
$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge a \wedge \neg c \wedge d$	Unsatisfiable			

Table 1: Computing Procedure of BASolver

Formulas	AveVAR	AveCL	AveFST	#AveSATInst	BASolver	MBlocking	BC	NBC	BDD
Dimacs(15)	680	17079	508.14	1	15	15	13	1	15
AProve(2)	8565	28931	7.32	78	2	2	2	2	2
Complete(4)	600	27140	0.02	1	4	0	4	4	0
Encryption(12)	8616	82773	44.75	1	12	10	12	11	6
Manthey(24)	5179	23692	780	2.58	24	18	14	21	16
Mp1(12)	16301	185948	630.41	305.75	12	11	12	9	8
Others(10)	11593	44557	325.19	1131	10	9	7	5	4
Total(79)	7323	393119	327.97	227.79	79	65	64	53	51

Table 2: Overall Performance of ALLSAT Computing Tools

There are 65 formulas solved by both BASolver and MBlocking, MBlocking uses less computing time than BASolver for 11 of the formulas, for the rest 54 formulas, BASolver uses less computing time. In total, BASolver uses 86274.54 seconds and MBlocking uses 110932.01 seconds, which is 22% more than BASolver uses.

There are 64 formulas solved by both BASolver and BC, BC uses less computing time than BASolver for 22 of the formulas, for the rest 44 formulas, BASolver uses less computing time. In total, BASolver uses 92198.9 seconds and BC uses 110932.01 seconds, which is 37% more than BASolver uses.

There are 53 formulas solved by both BASolver and NBC, NBC uses less computing time than BASolver for 20 of the formulas, for the rest 33 formulas, BASolver uses less computing time. In total, BASolver uses 28024.15 seconds and NBC uses 123761.88 seconds, which is 78% more than BASolver uses.

There are 51 formulas solved by BASolver of BDD, BDD uses less computing time than BASolver for 9 of the formulas, for the rest 42 formulas, BASolver uses less computing time. In total, BASolver uses 74686.244 seconds and BDD uses 107023.29 seconds, which is 31% more than BASolver uses.

Based on the comparison of number of solved formulas and computing time between BASolver and the other comparing tools, we conclude that BASolver is able to solve the most formulas with the least computing time, and MBlocking ranks the second place in the comparison. BC solved the third most number of formulas. The NBC and BDD tools solve less formulas with more computing time. It indicates that for formulas from the industrial benchmark, blocking based tools have advantages than non-blocking based strategies and BDD based strategies. We then compare the performance of blocking based tools, including BASolver, MBlocking and BC.

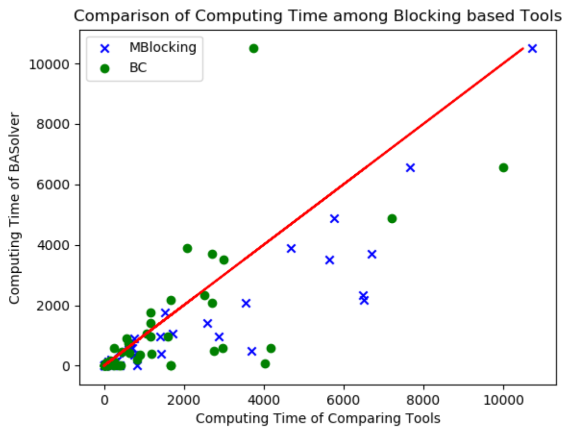


Figure 2: Comparison of Computing Time between BASolver and Other Tools

4.2 Compassion among Blocking based Tools

There are 53 formulas that are solved by all the the three tools. Figure 2 shows the computing time comparison among the 53 formulas. The x-axis is the computing time of the comparing tools, and the y-axis is the computing time of BASolver. The line (red) is the base line, set up by the computing time of BASolver, the circles (green) are the comparing data of BC, the crosses (blue) are the computing data of MBlocking. For the circles and crosses that are above the line, the computing time of the comparing tools are less than BASolver. Results show that for the majority of the formulas, BASolver needs less computing time than both MBlocking and BC. And MBlocking needs more computing time than BC on general. It is because that BC only uses the decision variables in the blocking clauses, therefore, the blocking clauses of BC is shorter than MBlocking.

Figure 3 shows the length of blocking clauses of the

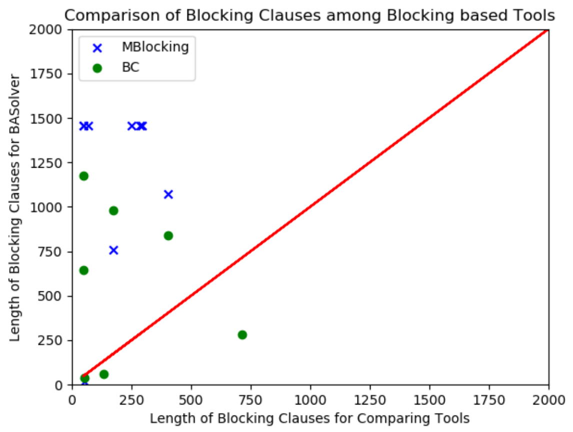


Figure 3: Comparison of Length of Blocking Clauses between BASOLVER and Other Tools

three tools. The x-axis is the length of blocking clauses in MBLOCKING and BC, and the y-axis is the length of blocking clauses in BASOLVER. The line (red) is the base line set up by the length of blocking clauses in BASOLVER, the circles (green) are the comparing data in BC, and the crosses (blue) are the comparing data in MBLOCKING. The scatter only shows the formulas which average length of blocking clauses is less than 2000. From Figure 3, we can see that for most of the formulas, the length of average blocking clauses in BASOLVER is shorter than MBLOCKING and BC, which is an important reason that BASOLVER uses less computing time than MBLOCKING and BC.

With the results of Figure 2 and 3, we observe that tools with shorter blocking clauses perform better than the ones with longer blocking clauses. The average length of blocking clauses in BASOLVER is 1026 (among the 79 formulas), the average length of blocking clauses in MBLOCKING is 22182 (among the 65 formulas) and the average length of blocking clauses in BC is 6180 (among the 64 formulas). The reason for BASOLVER with short length of blocking clauses is because by using backbone information, a huge part of variables are removed from the blocking clauses. Therefore, backbone information contributes to the computing of ALLSAT problems.

5 Related Work

There are mainly two kinds of ALLSAT solvers, blocking based solvers and non-blocking based solvers. BASOLVER is a blocking based ALLSAT solver. The main difference between blocking based and non-blocking based solvers is the use of blocking clauses. In order to avoid finding the already known solutions of the formula, blocking based tools generate the blocking clauses of each known solution and added the blocking clauses back to the solver. In the non-blocking based ALLSAT computing tools [Zhao and Wu, 2009] [Grumberg *et al.*, 2004] [Jabbour *et al.*, 2014] [Toda and Soh, 2016], backtrack techniques in the search tree are used to find more solutions of the given formula. Once a solution is found, the non-blocking based tools choose a decision level and backtracks the search tree to that level. A Different decision is made at that level and a new search path is generated based on the new decision.

A naive blocking based tool [McMillan, 2002] uses a SAT solver to find a solution of the given formula, then added the negation of the found formula as a blocking clause to the solver.

MBLOCKING [Toda and Soh, 2016] is another blocking based tool that uses the greedy strategy named minimal blocking strategy to generate the blocking clauses. For a solution of the given formula, MBLOCKING either computes the set of dominate variables based on the clauses coverage or the set of decision variables and their corresponding reason variables based on the search tree. Comparing to MBLOCKING, BASOLVER uses backbone information instead of decision variables to generate the blocking clauses. By using backbone variables, shorter blocking clauses are generated.

NBC [Toda and Soh, 2016] is a non-blocking based tool that backtracks the search tree to find every solution of the given formula. There are 4 different backtrack strategies in NBC, and by use these strategies in different orders, there are 8 different strategies in total in NBC. Each strategy performs differently on the formulas, therefore, the choice of strategies is a challenge for NBC. Comparing to NBC, BASOLVER only uses one strategy which is backbone variables to find every solution of the formula.

Another two approaches of ALLSAT computing are BDD based tools and P systems based tools. The idea of the BDD is to build an ordered binary decision tree, based on the OBDD, every solution is visited. But the building of the OBDD may take longer computing time than the pure computing of ALLSAT and the fixed order of the OBDD also affects the performance of BDD. P system based tools use P system to compute NP problems, including ALLSAT problems. P system are inherently paralleled and has been used in the solving of SAT problems recently [Ishii and Tagawa, 2010]. Ping [Ping and Ruilong, 2018] designs a family of P system and reduce the time complexity of ALLSAT computing to linear time. There are only theoretical algorithms and no experiment is conducted in the paper.

6 Conclusion

We proposed an ALLSAT computing tool BASOLVER which uses backbone information to get shorter blocking clauses and achieve higher efficiency. Comparing to other ALLSAT computing tools, BASOLVER removes backbone variables from the blocking clauses. With shorter blocking clauses, the complexity of the following SAT solving decreases, the number of SAT solving needed decreases and the efficiency of ALLSAT computing improves. Experiments show that within the given computing time and memory limitation, BASOLVER is able to solver more formulas than other comparing tools. For the formulas that are solved both by BASOLVER and the comparing tools, BASOLVER uses less computing time than the comparing tools. Therefore, BASOLVER is an efficient ALLSAT computing tool that performs the best among the existing tools with the given benchmark.

References

[Beyer and Keremoglu, 2011] Dirk Beyer and M Erkan Keremoglu. Cpachecker: a tool for configurable software verification. In *International Conference on Computer Aided Verification*, pages 184–190, 2011.

[Biere *et al.*, 2003] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. volume 58, pages 117–148, 2003.

[Bradley, 2012] Aaron R Bradley. Understanding ic3. In *Theory and Applications of Satisfiability Testing*, pages 1–14, 2012.

[Brauer *et al.*, 2011] Jörg Brauer, Andy King, and Jael Kriener. Existential quantification as incremental sat. In *International Conference on Computer Aided Verification*, pages 191–207. Springer, 2011.

[Cadar *et al.*, 2008] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In *Operating Systems Design and Implementation*, pages 209–224, 2008.

[Eén and Sörensson, 2003] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[Grumberg *et al.*, 2004] Orna Grumberg, Assaf Schuster, and Avi Yadgar. Memory efficient all-solutions sat solver and its application for reachability analysis. In *International Conference on Formal Methods in Computer-Aided Design*, pages 275–289. Springer, 2004.

[Ishii and Tagawa, 2010] Akihiro Fujiwara Ishii, Kota and Hirofumi Tagawa. Asynchronous p systems for sat and hamiltonian cycle problem. In *2010 Second World Congress on Nature and Biologically Inspired Computing, IEEE*, pages 513–519, 2010.

[Jabbour *et al.*, 2014] Said Jabbour, Jerry Lonlac, Lakhdar Sais, and Yakoub Salhi. Extending modern sat solvers for models enumeration. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pages 803–810. IEEE, 2014.

[Jin *et al.*, 2005] HoonSang Jin, HyoJung Han, and Fabio Somenzi. Efficient conflict analysis for finding all satisfying assignments of a boolean circuit. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 287–300. Springer, 2005.

[Kroening and Tautschnig, 2014] Daniel Kroening and Michael Tautschnig. Cbmc – c bounded model checker. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–391, 2014.

[Lahiri *et al.*, 2003] Shuvendu K Lahiri, Randal E Bryant, and Byron Cook. A symbolic approach to predicate abstraction. In *International Conference on Computer Aided Verification*, pages 141–153. Springer, 2003.

[Li *et al.*, 2017] Jianwen Li, Shufang Zhu, Yueling Zhang, Geguang Pu, and Moshe Y. Vardi. Safety model checking with complementary approximations. In *2017 IEEE/ACM International Conference on Computer-Aided Design*, pages 95–100, 2017.

[Li *et al.*, 2019] Jianwen Li, Kristin Y. Rozier, Geguang Pu, Yueling Zhang, and Moshe Y. Vardi. Sat-based explicit ltlf satisfiability checking. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 2946–2953, 2019.

[Lopes *et al.*, 2015] Nuno P Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. Checking beliefs in dynamic networks. In *12th $\{$USENIX$\}$ Symposium on Networked Systems Design and Implementation ($\{$NSDI$\}$ 15)*, pages 499–512, 2015.

[Majumdar *et al.*, 2014] Rupak Majumdar, Sai Deep Tetali, and Zilong Wang. Kuai: A model checker for software-defined networks. In *2014 Formal Methods in Computer-Aided Design*, pages 163–170. IEEE, 2014.

[McMillan, 2002] Ken L McMillan. Applying sat methods in unbounded symbolic model checking. In *International Conference on Computer Aided Verification*, pages 250–264. Springer, 2002.

[Morgado and Marques-Silva, 2005] António Morgado and Joao Marques-Silva. Good learning and implicit model enumeration. In *17th IEEE International Conference on Tools with Artificial Intelligence*, pages 6–pp. IEEE, 2005.

[Ping and Ruilong, 2018] Z. H. U. Jian C. H. E. N. Haizhu Ping, G. U. O. and Y. A. N. G. Ruilong. A linear-time solution for all-sat problem based on p system. pages 367–373, 2018.

[Sapra *et al.*, 2003] Samir Sapra, Michael Theobald, and Edmund Clarke. Sat-based algorithms for logic minimization. In *Proceedings 21st International Conference on Computer Design*, pages 510–517. IEEE, 2003.

[Toda and Soh, 2016] Takahisa Toda and Takehide Soh. Implementing efficient all solutions sat solvers. *Journal of Experimental Algorithmics (JEA)*, 21:1–12, 2016.

[Yu *et al.*, 2014] Yinlei Yu, Pramod Subramanyan, Nestan Tsiskaridze, and Sharad Malik. All-sat using minimal blocking clauses. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 86–91. IEEE, 2014.

[Zhang *et al.*, 2012] Shuyuan Zhang, Sharad Malik, and Rick McGeer. Verification of computer switching networks: An overview. In *International Symposium on Automated Technology for Verification and Analysis*, pages 1–16. Springer, 2012.

[Zhang *et al.*, 2018] Yueling Zhang, Min Zhang, Geguang Pu, Fu Song, and Jianwen Li. Towards backbone computing: A greedy-whitening based approach. *AI Communications*, 31(3):267–280, 2018.

[Zhao and Wu, 2009] Weinan Zhao and Weimin Wu. Asig: An all-solution sat solver for cnf formulas. pages 508–513, 2009.