Towards Backbone Computing: A Greedy-Whitening Based Approach

Abstract

Backbone is widely used in random SAT solving, ALL-SAT computing and planning. In this paper, we propose a greedy-whitening based approach for computing backbones of propositional Boolean formulae. We present a greedy-based algorithm with four heuristic searches to compute an under-approximation of non-backbone, and propose a whitening-based algorithm to compute an approximation of backbone from which the exact backbone is computed by applying iterative test backbone. We implemented our approach in Tool and conducted experiments on instances from industria/random track of SAT Competitions between xxx and yyy. Experimental results demonstrate that Tool is comparable to the state-of-the-art tool cb100. Moreover, Tool needs less SAT solver calls and outperforms cb100 on most of hard instances.

Introduction

Backbones are firstly generalized from the coloring problem. A pair of nodes are backbones if they always have the same color in every possible k-coloring(Walsh and Slaney 2001).

In satisfiability problem, given a formulae Φ, an truth assignment is a map from boolean variables to literals appeared in Φ. In an assignment, a literal can only be assigned as TRUE or FALSE. Given an assignment λ, if there exist at least one literal in a clause $\phi \in \Phi$ that has been assigned as TRUE according to λ, ϕ is called to be satisfied by λ. If there exists a λ that satisfy every clause $\phi \in \Phi$, λ is a model or a satisfied assignment of Φ. Backbones of a propositional formula Φ is a cluster of literals that are TRUE in every model of Φ(Bollobás et al. 2001; Kilby et al. 2005a).

Backbones have been studied in random 3-SAT problems (Dubois and Dequen 2001), optimization problems(Culberson and Gent 2001; Kilby et al. 2005b; Walsh and Slaney 2001), as well as Maximal Satisfiability(MSS) problems(Menaï 2005). As shown in (Zhang, Rangan, and Looks 2003), backbones are applied to local minima of Walksat and improves the performance of Walksat by making biased moves in a local search. Another similar application of backbones information is shown in (Zhang and Looks 2005), in this experiment, backbones information significantly contributes to the acceleration of Lin-Kernighan(LK) local search family algorithms when dealing with Travel Salesman Problem(TSP). A more recent application of backbones arises in (Zhu et al. 2011). A number of backbones extracting approaches are proposed and applied to post silicon fault localisation. The results show that backbones extracting using SAT solvers are suitable for large scale applications with only a little SAT solver calls.

As shown above, finding backbones is the key in many practical applications, such as planning problem and constraint satisfaction problem. It has been proved that backbones computing is a co-NP problem(Janota 2010), which have rise huge challenges. A number of backbones extraction algorithms have been proposed in recent years. All state-of-the-art backbones extraction approaches employ SAT solvers, MiniSAT(En and Srensson 2004) for most of them. Implicant enumeration(McMillan 2002; Ravi and Somenzi 2004) enumerates implicants of formula Φ one by one and updates the backbones estimation in each iteration. The negation of an implicant is added to Φ in order to avoid finding the same implicant. For an implicant λ, the negation of λ is $\bigvee_{l \in \lambda} \neg l$. Standard implicants reducing techniques can be applied to mitigate the size of blocking clauses. It have to enumerate all models of Φ before the algorithm terminates, which is unnecessary costly.

Zhu et al, proposed an iterative SAT testing algorithm (Zhu et al. 2011). The algorithm maintains an estimation of backbones Φ_{BL}. The negation of Φ_{BL} is the disjunction of each complementing literals in backbones estimation, i.e., $\neg\Phi_{\mathsf{BL}} = \bigvee_{\neg l}, l \in \Phi_{\mathsf{BL}}$. In each iteration, a clause that formed by $\neg\Phi_{\mathsf{BL}}$ conjuncted to Φ and formed Φ'. If Φ' is satisfiable, it means that at least one non-backbone is in the backbone estimation, estimation is refined by removing the non-backbone literal. The process is repeated until Φ' is not satisfiable any longer. Along with the estimation, the clauses number of Φ' is monotone increasing due to the continuously disjunction in each iteration, which dramatically promote the complexity of Φ'. In other words, for each iteration, it takes longer CPU time than the last iteration.

The Core Based Algorithm presented in (Janota, Lynce, and Marques-Silva 2015) is stable and effectiveness. It considers complementing of the model as assumptions input for

SAT solver in each iteration. If Φ is unsatisfied under given assumptions, a core is returned by SAT solver to indicate reasons. According to the implementation of MiniSAT 2.2 (En and Srensson 2004), the reason is a part of the given assumptions. Whenever there is exactly one literal in the reason, the literal is a backbone. If there is more than one literal in the reason, they will be marked as visited. If every literal in an iteration is visited, iterative SAT testing will be invoked to test the rest of unmarked literals. According to the author, for the lower percentages of backbones, Core Based Algorithms are significantly better. When the percentage of backbone is over 25%, Core Based Algorithm behave very similarly Iterative SAT Testing Algorithm.

Revisited previous researches, backbones computing of low density backbones formulae is quiet efficiencies and efficiency. In this paper, we focus on the hard instance, proposed a novel approach to spot non-backbones ahead and to estimate backbones. Instead of accumulating clauses to formula like Iterative SAT testing does, we test the literals one by one by leveraging assumptions. Assumptions are a group of literals. We consider assumptions as a kind of constraint input, it means that the literals in assumptions must be assigned to true in an assignment. If there exists a model λ such that it satisfy every literals in assumptions, the formula is satisfied under these assumptions. According to the satisfiability theory, assumptions are equivalence to clauses that only contains the assumption literal. Compared with clauses conjunction, assumptions won't increase the complexity of formula since no clauses are added to the formula. Also, assumptions is more suitable for iterative SAT testing since it can be reset in each iteration.

Using variables clauses coverage analysis, a group of non-backbone literals and an estimation of backbones are obtained, refereed as $\overline{\mathsf{BL}}_u$ and HDBS, respectively. For each literals in HDBS, Minisat with assumptions will be evoked. The only assumption in each iteration is the current testing literal. If Minisat returns false, the literal is a backbone, it will be conjuncted with the original formula to simplify it.

The approach leveraging Whitening Algorithm and other pervious backbones computing algorithms, designed and implemented a Multi Estimation Based Algorithms (MEB) , experiments are conducted to evaluate MEB, results showed that MEB is compatible with Core Based Algorithm when dealing with low density backbones formulae. For hard formulae with dense backbones, MEB outperforms Core Based Approach considering the total SAT solver time and SAT calls number. Compared with Core Based Algorithm which directly calculate backbones. MEB estimate non-backbone literals and backbone literals ahead. Unlike the Iterative SAT Testing approach, the estimation doesn't need to rely on SAT solvers. Only one SAT solver call is needed during the estimation process. Experiments indicate that, the average dense of backbones in HDBS is over 50%, which helps MEB to recognize more backbones in less SAT solver calls than CB needs. Especially for hard formulae, MEB will save a large amount of CPU time since it invoking less Minisat than CB does.

This paper is organized as follows. Section 2 introduces the concept of backbones. Section 3 relates backbones estimate reducing procedures to Core Based Backbones Extraction Algorithms and evolved to Filter Core Based(FCB) Backbone Extraction Algorithms. Section 4 presents the experimental evaluation of FCB. Section 5 makes a conclusion and discusses about future work.

Preliminaries

We fix a finite set $\mathcal{X}$ of *Boolean variables*. A *literal* l is either a Boolean variable $x \in \mathcal{X}$ or its negation $\neg x$. The negation of a literal $\neg x$ is x, i.e., $\neg\neg x = x$. A *clause* ϕ is a disjunction of literals $\bigvee_{i=1}^{n} l_i$, which may be regarded as the set of literals $\{l_i \mid 1 \leq i \leq n\}$.

A *formula* Φ over $\mathcal{X}$ is a Boolean combination of variables $\mathcal{X}$. We assume that formulae are given in conjunctive normal form (CNF), namely each formula Φ is a conjunction of clauses $\bigwedge_{i=1}^{n} \phi_i$ which may be regarded as a set of clauses $\{\phi_i \mid 1 \leq i \leq n\}$. Given a formula Φ, let $\mathsf{var}(\Phi)$ (resp. $\mathsf{lit}(\Phi)$ and $\mathsf{cls}(\Phi)$) denote the set of variables (resp. literals and clauses) used in Φ. We use $\neg\mathsf{lit}(\Phi)$ to denote the set $\{\neg l \mid l \in \mathsf{lit}(\Phi)\}$. The *size* $|\Phi|$ of Φ is the number of literals of Φ. Given a formula Φ and a literal $l \in \mathsf{lit}(\Phi)$, let $\Phi_l \subseteq \Phi$ be the set of clauses $\{\phi \in \Phi \mid l \in \phi\}$.

An *assignment* is a function $\lambda : \mathcal{X} \to \{0, 1\}$, where 1 (resp. 0) denotes true (resp. false). Given an assignment λ, let $\lambda[x \mapsto b]$ for some $b \in \{0, 1\}$ be the assignment which is equal to λ except for $\lambda[x \mapsto b](x) = b$. An assignment λ *satisfies* a formula Φ, denoted by $\lambda \models \Phi$, iff assigning $\lambda(x)$ to x for $x \in \mathsf{var}(\Phi)$ makes Φ true. An assignment λ is a *model* of Φ if $\lambda \models \Phi$. A formula Φ is *satisfiable* iff there exists an assignment λ such that $\lambda \models \Phi$. Given a formula Φ, the *satisfiability problem* is to decide whether Φ is satisfiable or not.

Definition 1 (Backbone)**.** *Given a satisfiable formula Φ, a literal l is a* backbone literal *of Φ iff for all assignments λ such that $\lambda \models \Phi$, $\lambda \models l$. The* backbone $\mathsf{BL}(\Phi)$ *of Φ is the set of backbone literals of Φ.*

It is known that the backbone $\mathsf{BL}(\Phi)$ for each formula Φ is unique and a backbone literal l of Φ must be in $\mathsf{BL}(\Phi)$ (Janota, Lynce, and Marques-Silva 2015). The backbone of an unsatisfiable formula can be defined as an empty set. Therefore, in this work, we focus on satisfiable formulae. We will use $\overline{\mathsf{BL}}(\Phi)$ to denote the set $\mathsf{lit}(\Phi) \setminus \mathsf{BL}(\Phi)$.

Theorem 1. *(Janota 2010) Given a satisfiable formula Φ and a literal l, deciding whether l is a backbone literal is co-NP-complete.*

Definition 2 (Satisfied literal)**.** *Given a model λ of the formula Φ and a clause $\phi \in \Phi$, for each literal $l \in \phi$, l is a* satisfied literal *of ϕ iff $\lambda \models l$. l is a* unique satisfied literal *of ϕ if there is no satisfied literal l' of $\phi \setminus \{l\}$.*

Let us consider the formula $\Phi = \{\neg x_1 \vee \neg x_2, x_1, x_3 \vee x_4\}$, then, $\mathsf{var}(\Phi) = \{x_1, x_2, x_3, x_4\}$, $\mathsf{lit}(\Phi) = \{\neg x_1, x_1, \neg x_2, x_3, x_4\}$, $\Phi_{\neg x_2} = \{\neg x_1 \vee \neg x_2\}$ and $\mathsf{BL}(\Phi) = \{x_1, \neg x_2\}$.

Overview of Our Approach

TBD.

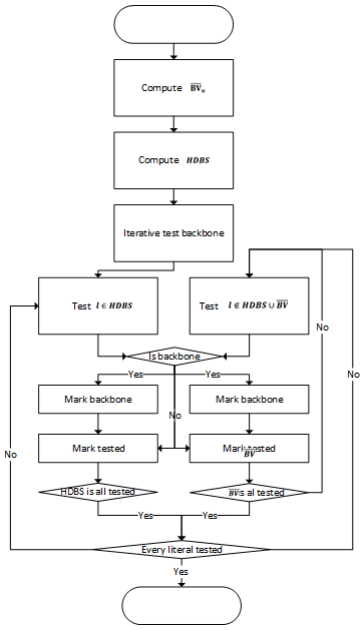

Figure 1: Flow Path of MEB

In this section, we present the overview of our approach Tool which is depicted in Figure 1. Tool comprises three components: Computing **NB**$\lfloor$, Computing *h*BL and Removing non-backbone literals from *h*BL.

We propose two algorithms for computing $\overline{\mathsf{BL}}_u\Phi$. The first algorithm computing **MBNB**$\lfloor$. Second one **HBNB**$\lfloor$....

MBNB$\lfloor$computes a complete non-backbone literals set of a given model. It extracts the relation between literals and clauses, and generate a new model by applying single model rotation. The satisfiability of an assignment generated by single model rotation is related with the literal that complemented in the rotation. If a complementing a literal won't make any clause unsatisfiable, then a new model is generated. It's obvious that is there are at least two satisfied literal in a clause, then a new assignment generated by complementing one of the literals won't change the satisfiability of this clause. Therefore, by only complementing such literals, new models obtained from single model rotation.

HBNB$\lfloor$computes extends the result of **MBNB**$\lfloor$with chain model rotation. With multiply rotations, we generate more models with from the initial model. It employs four heuristic depth first searches using Greedy strategies to find virous possible combinations of model rotation. The search is guided by the least/most coverage of literals which indicates the frequency of literals, and the maximal/minimal ratio of literals number to total length of clauses that contain the literal which indicated the importance of the literal to the formula.

Based on $\overline{\mathsf{BL}}_u\Phi$, we apply optimized Whitening Algorithm to compute an approximation of backbone literals by eliminating one of the pattern that cause a false positive of Whitening Algorithm. We observe that Whitening Algorithm fail to recognize non-backbone literals when there is a large proportion of clauses that only have unique satisfied literal to a given model. We rule out non-backbone literals in this situation by distinguishing a special pattern among them. Among a group of clauses that only contain unique satisfied literal, there may exists a possible model rotation chain. By complementing the unique satisfied literal and more literals in each clause, another satisfied literal may emerge. ....

For both high and low density of backbones, our approach is able to remove non-backbones ahead and accelerate the extraction.

Computing $\overline{\mathsf{BL}}_u(\Phi)$

An Naïve Algorithm

Whitening Algorithm can't be applied to backbones extraction directly because it suffers from the over-estimation of non-backbones, i.e, backbones can also be selected to the $\overline{\mathsf{BL}}_u(\Phi)$ by the Whitening Algorithm. To fix this defect, only a part of Whitening Algorithm is employed. Given a model λ, there is a possibility that more models can be obtained only by complementing several literals in λ without calling a SAT solver. According to the theory of satisfiability, a model λ is a truth assignment that satisfies every clause in the formula, i.e., there does not exist a clause ϕ that contains no literal in model λ. With a given model λ, non-backbones estimation approach is able to compute several similar models that only one literal in complemented. The literals that has been complemented are non-backbones.

Algorithm 1: Non-backbones under-approximation

Input : Φ: a formula
Output: $\overline{\mathsf{BL}}_u(\Phi)$: under-approximation of non-backbones

1 $\Psi = \overline{\mathsf{BL}}_u(\Phi) = \emptyset$;
2 $(b, \lambda) = \mathsf{SAT}(\Phi)$;
3 **if** $b == 0$ **then return** $\mathsf{lit}(\Phi)$;
4 **for** $\phi \in \Phi$ **do**
5 $\quad \Psi = \Psi \cup \{\phi \in \Phi \mid \exists x_1, x_2, x_1 \neq x_2, \lambda(x_1), \lambda(x_2) \models \phi\}$;
6 $\quad \overline{\mathsf{BL}}_u(\Phi) = \overline{\mathsf{BL}}_u(\Phi) \cup \{x \in \mathsf{lit}(\Phi) \mid \forall \phi \in \Phi : \lambda(x) \models \phi \Longrightarrow \phi \in \Psi\}$;
7 **return** $\overline{\mathsf{BL}}_u(\Phi)$;

Given a formula Φ, Algorithm 1 computes a set of backbone literals $\overline{\mathsf{BL}}_u(\Phi)$ which is a under-approximation of $\overline{\mathsf{BL}}(\Phi)$. Algorithm 1 maintains a set Ψ of clauses and a set of candidate literals of $\overline{\mathsf{BL}}_u(\Phi)$, both of them are initialized to $\emptyset$ at Line 1. At Line 2, an off-the-shelf SAT Solver is called which returns a pair (b, λ), where b denotes whether Φ is satisfiable or not. If b is TRUE, then λ is an assignment that satisfies Φ. If b is FALSE, $\lambda = \emptyset$, Algorithm 1 terminates and returns the set $\mathsf{lit}(\Phi)$ at Line 3. The Loop at Lines $4-6$ iteratively updates the sets $\overline{\mathsf{BL}}_u$ and Ψ for each clause $\phi \in \Phi$. For each clause $\phi \in \Phi$, if there are at least two literals in ϕ that are satisfied by the assignment λ, then the clause ϕ is added into Ψ at Line 5. If there is a literal that only satisfy clauses in Ψ, the literal is added into $\overline{\mathsf{BL}}_u(\Phi)$ at Line 6.

Lemma 1. *Given a satisfied formula Φ, a model λ. Let $\overline{BL}(\Phi)$ be the non-backbones of Φ. $x \in lit(\Phi), x \in \overline{BL}(\Phi)$ iff $\exists \lambda' \models \Phi, \lambda \neq \lambda', \lambda(x) = \neg\lambda'(x)$.*

Proof. Suppose literal $x \in \mathsf{lit}(\Phi) \in \overline{\mathsf{BL}}(\Phi)$, according to the definition of $\overline{\mathsf{BL}}(\Phi)$, it must exist two satisfied $\lambda, \lambda' \models \Phi$ such that $\lambda(x) = \neg\lambda'(x)$, i.e., $\forall x \in \overline{\mathsf{BL}}(\Phi) \Longrightarrow \exists \lambda \neq \lambda', \lambda(x) = \neg\lambda'(x), \lambda \models \Phi, \lambda' \models \Phi$.

Suppose λ, λ' are two satisfied assignments for a formula Φ, $\lambda(x) = \neg\lambda'(x)$. According to the definition of $\overline{\mathsf{BL}}(\Phi)$, x is in $\overline{\mathsf{BL}}(\Phi)$. □

Theorem 2. *Given a satisfiable formula Φ, let $\overline{BL}_u(\Phi)$ be the set of variables obtained by applying Algorithm 1, then $\overline{BL}_u(\Phi) \subseteq \overline{BL}(\Phi)$.*

Proof. Given a satisfied formula Φ, a satisfied assignment $\lambda \models \Phi$. Suppose literal $x \in \overline{\mathsf{BL}}_u(\Phi)$, let Φ_x be the set of clauses that uses x.

Since Φ is a satisfied formula, according to the theory of satisfiability, it must exist at least one model $\lambda \models \Phi$. It exists a model $\lambda' = \lambda[x \mapsto \neg\lambda(x)]$, $\lambda' \models \Phi \setminus \Phi_x$. For every clause $\phi \in \Phi_x$, there exists another x_1, $x \neq x_1$ such that $\lambda(x_1) \models \phi$ (Line 5, 6), $\lambda'(x_1) = \lambda(x_1)$ therefore $\lambda' \models \Phi_x$. For every clause $\phi \in \Phi_{\neg x}$, $\lambda(x) \not\models \phi$, it exists another x_2, such that $\lambda(x_2) \models \phi$ (Line 7), $\lambda'(x_2) = \lambda(x_2)$, therefore $\lambda' \models \Phi_{\neg x}$. According to the above proof $\lambda' \models \Phi \setminus \Phi_x \cup \Phi_x \cup \Phi_{\neg x} = \Phi$. According to Lemma 7, $\exists \lambda, \lambda' \models \Phi, \lambda(x) = \neg\lambda'(x) \Longrightarrow x \in \overline{\mathsf{BL}}(\Phi)$. □

In this section, we proved that there is no backbones in $\overline{\mathsf{BL}}_u(\Phi)$, it's safe to directly remove the literals in $\overline{\mathsf{BL}}_u(\Phi)$ from backbones estimation.

Heuristic-based Algorithm

With different models returned by MINISAT, the result of the algorithm in previous section diverged. To shrink the gaps, greedy based literals selection strategies are proposed. With a given literals selection strategy, a chain of literals can be complemented at the same time. In other words, a chain of models will be obtained by the changing assignemnets of literals, coverage of clauses for each literals are updated simultaneously. More models will be generated in this way.

Four Algorithms with different heuristic literals selection strategies are proposed in this subsection. Let $\overline{\mathsf{BL}}_0$ be the set of literals that only satisfy clauses that have at least two satisfied literals by a satisfied assignment, i.e., , $\overline{\mathsf{BL}}_0 = \{x \in \mathsf{lit}(\Phi) \mid \forall l_x \models \phi : \exists l_{x'} \neq l_x, l_{x'} \models \phi\}$. The four different algorithms choose exactly one variable $x \in V(\Phi)$ and generate a new assignment $\lambda' = \lambda[\lambda(x) \mapsto \neg\lambda(x)]$. Ψ and $\overline{\mathsf{BL}}_i$ defined in Algorithm 1 will be updated using λ'.

Heuristic searching strategies that used at Line 9 in Algorithm 2 are:

1. Let $|\mathsf{lit}(\phi)|$ be the number of clauses that used in a clause $\phi \in \Phi$, for every literal $x \in \overline{\mathsf{BL}}_1(\Phi)$, select exactly one literal at each iteration of the loop such that $\max\{|\phi_x|\}$.

Algorithm 2: Heuristic extension for computing $\overline{\mathsf{BL}}_u$

Input : Φ: a formula
Output: $\overline{\mathsf{BL}}_i(\Phi)$: under-approximation of non-backbones

1 choose search strategy i to compute $\overline{\mathsf{BL}}_i(\Phi)$;
2 $\Psi = \overline{\mathsf{BL}}_i(\Phi) = \emptyset$;
3 $(b, \lambda) = \mathsf{SAT}(\Phi)$;
4 **if** $b == 0$ **then return** $\mathsf{lit}(\Phi)$;
5 **for** $\phi \in \Phi$ **do**
6 $\quad \Psi = \Psi \cup \{\phi \in \Phi \mid \exists x_1, x_2, x_1 \neq x_2, \lambda(x_1), \lambda(x_2) \models \phi\}$;
7 $\quad \overline{\mathsf{BL}}_i(\Phi) = \overline{\mathsf{BL}}_i(\Phi) \cup \{x \in \mathsf{lit}(\Phi) \mid \forall \phi \in \Phi : \lambda(x) \models \phi \Longrightarrow \phi \in \Psi\}$;
8 **repeat**
9 $\quad$ select exactly one literal using strategy i;
10 $\quad \Psi = \Psi \backslash \{\phi \in \Psi, \lambda(x) \models \phi, |\{l \in \phi \mid l \models \phi\}| = 2\}$;
11 $\quad \Psi = \Psi \cup \{\forall \phi \in \Phi, \neg\lambda(x) \models \phi\}$;
12 $\quad \lambda = \lambda[x \mapsto \neg\lambda(x)]$;
13 $\quad \overline{\mathsf{BL}}_i(\Phi) = \overline{\mathsf{BL}}_i(\Phi) \cup \{x \in \mathsf{lit}(\Phi) \mid \forall \phi \in \Phi : \lambda(x) \models \phi \Longrightarrow \phi \in \Psi\}$;
14 **until** *No Update of* $\overline{\mathsf{BL}}_i(\Phi)$;
15 **return** $\overline{\mathsf{BL}}_i(\Phi)$;

2. Let $|\mathsf{lit}(\phi)|$ be the number of clauses that used in a clause $\phi \in \Phi$, for every literal $x \in \overline{\mathsf{BL}}_2(\Phi)$, select exactly one literal at each iteration of the loop such that $\min\{|\phi_x|\}$.
3. Let $|\mathsf{lit}(\phi)|$ be the number of clauses that used in a clause $\phi \in \Phi$, let $|\Phi_x|$ be the total length of clauses $\phi \in \Phi$ that used x, i.e., $|\Phi_x| = \sum\{\phi \in \Phi \mid \phi \in \Phi_x\}$ for every literal $x \in \overline{\mathsf{BL}}_3(\Phi)$, select exactly one literal at each iteration of the loop such that $\max\{|\phi_x| \div |\Phi_x|\}$.
4. Let $|\mathsf{lit}(\phi)|$ be the number of clauses that used in a clause $\phi \in \Phi$, let $|\Phi_x|$ be the total length of clauses $\phi \in \Phi$ that used x, i.e., $|\Phi_x| = \sum\{\phi \in \Phi \mid \phi \in \Phi_x\}$ for every literal $x \in \overline{\mathsf{BL}}_4(\Phi)$, select exactly one literal at each iteration of the loop such that $\min\{|\phi_x| \div |\Phi_x|\}$.

Theorem 3. *Let $\overline{BL}_1(\Phi), \overline{BL}_2(\Phi), \overline{BL}_3(\Phi), \overline{BL}_4(\Phi)$ be the set of literals obtained by applying Algorithm 2 with search strategy $1, 2, 3, 4$, then $\overline{BL}_1(\Phi) \subseteq \overline{BL}(\Phi)$, $\overline{BL}_2(\Phi) \subseteq \overline{BL}(\Phi)$, $\overline{BL}_3(\Phi) \subseteq \overline{BL}(\Phi)$, $\overline{BL}_4(\Phi) \subseteq \overline{BL}(\Phi)$.*

Proof. Given a satisfied formula Φ, a literal $x \in \overline{\mathsf{BL}}_i(\Phi)$. It exists a satisfied assignment λ such that $\forall \phi \in \Phi_x, x \models \lambda :\Longrightarrow \phi \in \Psi$ according to Line 7, 13 in Algorithm 2, i.e., it exists another variable $x', \lambda(x') \models \phi$. According to the satisfiability theory, it must exist a satisfied assignment $\lambda' = \lambda(x \mapsto \neg\lambda(x)) \models \Phi$. According to Lemma 7, $x \in \overline{\mathsf{BL}}(\Phi)$. □

The final $\overline{\mathsf{BL}}_u(\Phi)$ result of our approach is $\overline{\mathsf{BL}}_u = \bigcup_{i=1}^{4} \overline{\mathsf{BL}}_i(\Phi) \cup \overline{\mathsf{BL}}$.

Compared with $\overline{\mathsf{BL}}_u(\Phi)$, the strategies applied in this sections can be viewed as four different kinds of depth-first

search, while $\overline{\mathsf{BL}}_u(\Phi)$ is a width-first search. With different search methods, we are able to explore more paths and obtain more diverse models.

Computing $h\mathsf{BL}(\Phi)$

Algorithm 3: Backbones Estimation of Φ, $\mathsf{HDBS}(\Phi)$

Input : Φ: a formula
Output: $\mathsf{HDBS}(\Phi)$:Backbones Estimation of Φ

1 $\overline{\mathsf{BC}} = \mathsf{HDBS}(\Phi) = \emptyset$;
2 $\overline{\mathsf{BL}}_e = \overline{\mathsf{BL}}_u$;
3 $(b, \lambda) = \mathsf{SAT}(\Phi)$;
4 **if** $b == 0$ **then return** $\mathsf{lit}(\Phi)$;
5 **for** $\phi \in \Phi$ **do**
6 $\quad \overline{\mathsf{BC}} = \overline{\mathsf{BC}} \cup \{\phi \in \Phi \mid \exists x \in \phi, x \in \overline{\mathsf{BL}}_u\}$;
7 $\quad \overline{\mathsf{BL}}_e(\Phi) = \overline{\mathsf{BL}}_e(\Phi) \cup \{x \in \mathsf{lit}(\Phi) \mid \forall \phi \in \Phi : \lambda(x) \models \phi \Longrightarrow \phi \in \overline{\mathsf{BC}}\}$;
8 **repeat**
9 $\quad \overline{\mathsf{BC}} = \overline{\mathsf{BC}} \cup \{\phi \in \Phi \setminus \overline{\mathsf{BC}} \mid \exists x \in \overline{\mathsf{BL}}_e(\Phi) \vee \exists \neg x \in \overline{\mathsf{BL}}_e, x \in \mathsf{lit}(\phi)\}$;
10 $\quad \overline{\mathsf{BL}}_e(\Phi) = \overline{\mathsf{BL}}_e(\Phi) \cup \{x \in \mathsf{lit}(\Phi) \mid \forall \phi \in \Phi : \lambda(x) \models \phi \Longrightarrow \phi \in \Psi\}$;
11 **until** *No Update of* $\overline{BL}_e$;
12 $\mathsf{HDBS} = \mathsf{lit}(\Phi) \setminus \overline{\mathsf{BL}}_e(\Phi)$;
13 **for** $x \in$ *HDBS*(Φ) **do**
14 $\quad k = 1$;
15 $\quad \Phi_x^0 = \{x\}$;
16 $\quad$ **repeat**
17 $\quad\quad \Phi_x^k = \{\phi \in \Phi \mid \neg\mathsf{lit}(\Phi_x^{k-1}) \in \phi\}$;
18 $\quad\quad$ **if** $\neg x \in$ *lit*(Φ_x^k) **then**
19 $\quad\quad\quad \mathsf{HDBS}(\Phi) = \mathsf{HDBS}(\Phi) \setminus \{x\}$;
20 $\quad\quad\quad Break$;
21 $\quad\quad k = k + 1$;
22 $\quad$ **until** $\Phi_{l_x}^k = \Phi$;
23 **return** $\mathsf{HDBS}(\Phi)$;

To increase the density of backbones, we propose HDBS approach to estimate literals that are highly likely to be backbones. In other words, HDBS increase the backbones density by estimations. There are two parts of HDBS approach. The first part extends $\overline{\mathsf{BL}}_u$ to find more likely backbone literals. The new estimation of variables is called $\overline{\mathsf{BL}}_e$. It relax the constraint of estimation selection constraints.

The second part of HDBS approach is a deletion based approach. HDBS is initialized with the complement of $\overline{\mathsf{BL}}_e$. There is a possibility that there are still non-backbones not in $\overline{\mathsf{BL}}_e$. If that happens, there must exist another non-backbone literal that $\overline{\mathsf{BL}}_e$ failed to recognise. Then, both of the two literals are added to HDBS, in this way, HDBS is updated.

Line $5-10$ compute $\overline{\mathsf{BL}}_e$. The loop from Line $8-11$ iterative updated $\overline{\mathsf{BC}}$ and HDBS. In Line 9, if a clause $\phi \in \Phi$ contains a literal or its negation l such that $l \in \overline{\mathsf{BL}}_u \vee \neg l \in \overline{\mathsf{BL}}_u$, ϕ is added to $\overline{\mathsf{BC}}$. Given a model λ, suppose $l \in \phi$, $l \in \lambda$, according to Lemma 7, there exists a model $\lambda' \neq \lambda \models \Phi$. Therefore, clause ϕ will be satisfied under any truth assignment of l. Given a model λ, suppose $l \in \phi$, $\neg l \in \lambda$, according to Lemma 7, there exists a model λ' with $\lambda'(l) = \neg\lambda(l)$. It means that clause ϕ will be satisfied by l when complementing $\neg l$. Therefore, clause ϕ will be satisfied under any truth assignment of l. With the updated $\overline{\mathsf{BC}}$, $\overline{\mathsf{BL}}_e$ is updated in Line 10.

Line 12 initialize HDBS as the complement of $\overline{\mathsf{BL}}_e$, since our goal is to find a group of literals with dense backbones, we want to find non-backbone literals in HDBS without applying MINISAT. Line 13-22 explains how non-backbones are found in HDBS. Given a literal $l \in \mathsf{HDBS}$, with the pre-condition that $l \notin \overline{\mathsf{BL}}_e$, there doesn't exist tow different model λ, λ', such that $\lambda(l) = \neg\lambda'(l)$. In other words, there must exist another non-backbone literal $x_1 \in \mathsf{HDBS}$ in Φ_x, such that x_1 complemented together with x. Suppose $\phi_1, \phi_2, \phi_3 \notin \overline{\mathsf{BC}}$, suppose $x_1, \neg x_1, x_2, \neg x_2, x_3, \neg x_3 \in \mathsf{HDBS}$, $x_1 \in \phi_1, \neg x_2 \in \phi_1$, $x_2 \in \phi_2, \neg x_3 \in \phi_2$, $x_3 \in \phi_3, \neg x_1 \in \phi_3$. Given a model λ, where $\lambda(x_1) = 1, \lambda(x_2) = 1, \lambda(x_3) = 1$. x_1, x_2, x_3 are the only satisfied literal in ϕ_1, ϕ_2, ϕ_3 respectively. It can be observed that if x_1 is a non-backbone literal, in order to generate a λ' that satisfy ϕ_1, ϕ_2 and ϕ_3, x_1, x_2, x_3 have to complemented simultaneously. Based on the observation, line 13-22 tries to find a group of literals that have to complement at the same time to generate a new model. Φ_x^0 is initialized with the a literal $x \in \mathsf{HDBS}$, and line 17-21 will find the group of literals that have to negate together if x is a non-backbone literal. If there is no literal found by the end of line 22, x will remained in HDBS. In a iterative searching process, Φ_x^k is computed with the information of k-1 step at Line 17. For each step of Φ_x^k, it stands for the clauses that contains at least one literal that showed up in Φ_x^{k-1}. If the negation literal of x is founded during the search, the search terminates for x and move to next literal in HDBS.

With the above steps, the percentage of backbone literals in HDBS will be relatively high. With dense backbones HDBS, MEB will achieve the best performance.

Theorem 4. *The complexity of Algorithm 3 from Line 5 to 23 is in polynomial time.*

Proof. Suppose $|\mathsf{lit}(\Phi)| = m$, $|\mathsf{cls}(\Phi)| = n$. Suppose the average length of $\phi \in \Phi$ is k, i.e., $|\phi \in \Phi| = k$. Suppose the average number of clauses that each literals appears is l, i.e., $|\Phi_l| = l$. For Line 5 to 8, there is a loop for each clause in $\phi \in \Phi$ to test if ϕ is in $\overline{\mathsf{BC}}$ or not. The complexity of each iteration is $O(k)$. The complexity of Line 5 to 8 is $O(m * k)$. For Line 9 to 11, there is a loop for each literal that have been selected to $\overline{\mathsf{BL}}_e$. In each iteration of the loop, $\overline{\mathsf{BC}}$ is updated with the complexity of $O(1)$. The complexity of Line 10 in the loop is $O(l)$. The worst case is that every literal $l \in \mathsf{lit}(\Phi)$ have been selected to $\overline{\mathsf{BL}}_e$ one by one. The complexity of Line 9 to 11 is $O(m * l)$. For Line 13 to 23, the complexity of Line 14, 15 is $O(1)$. There is a loop of literals that in HDBS. For the worst case, Line 23 won't be executed until every literal of HDBS had been visited. In

the first iteration of the loop, the complexity of Line 17 is $O(k * l)$. In the second iteration, the complexity of Line 17 is $O(k * l * l)$. In the i^{th} iteration of the loop, the complexity of Line 17 is $O(k * l^i)$. Therefore, the complexity of Line 17 is $O(k * l^n)$. The complexity of Line 18-22 is $O(1)$. The complexity of Line 13 to 23 is $O(l * l^i)$. As discussed above, the complexity of Line 5 to Line 23 in Algorithm 3 is $O(k * l^i + m * k + m * l)$. □

With $\overline{\mathsf{BL}}_u$ and HDBS above, both backbones estimation HDBS and non-backbones estimation $\overline{\mathsf{BL}}_u$ can be obtained. First of all, HDBS are tested iteratively by adding one assumption to the formula at a time. In each iteration, MINISAT returns true if the literals in assumptions is a non-backbone or false if the literal is a backbone. The first several MINISAT calls may be slow depending on the complexity of the formula. With more backbones being recognise by the testing, the testing procedure will be dramatically accelerated. After teasing of HDBS, the complement of $(\mathsf{HDBS} \cup \overline{\mathsf{BL}}_u)$ is iterative tested one by one, which can be finished in a blink. The flow path of MEB approach is showed in Fig 1.

It first start with computing $\overline{\mathsf{BL}}_u$ and HDBS. Iterative SAT solver calls are applied to each literal in HDBS, in the flow path, a literal l is marked as tested after the MINISAT call. If MINISAT returns true, then l is marked as backbones. Otherwise, l is a non-backbone and can't contribute to MEB anymore. The complement of $\mathsf{HDBS} \cup \overline{\mathsf{BL}}_u$ is the next test candidate set. If every literal of formula is already marked tested, MEB terminates.

Algorithm 4: Backbones Extracting Using MUC

Input : Φ: a formula
Output: $\mathsf{BL}(\Phi)$:Backbones of Φ

```
1  BL = ∅;
2  Φ' = Φ;
3  (b, λ) = SAT(Φ);
4  if b == 0 then return lit(Φ);
5  for TRUE do
6      for l ∈ λ do
7          Φ' = Φ' ∧ ¬l;
8      (b, ω) = MUC(Φ');
9      if b == 1 then return BL(Φ);
10     if b == 0 then
11         BL = BL ∪ {l | l ∈ ω};
12         Φ' = Φ' \ ω;
13 return BL(Φ);
```

Experiments results

Our tool MEB (Multi Estimation Based Backbones Extraction) was built with C++ using minisat(En and Srensson 2004) as the underlying SAT solver. It contains the $\overline{\mathsf{BL}}_u$ computing and HDBS computing proposed in Section 3. The experiments were conducted on a cluster of IBM iDataPlex 2.83 GHz, each instance was running with a timeout of 1 hour and memory limit of 8 GB.

Since SAT solver is an oracle in the extraction of backbones, calling a SAT solver and waiting for results takes the majority of CPU time. We set up two metrics for comparisons, total SAT solvers CPU time(st) and SAT solver calls number(sc).

Experimental Strategies

Earlier work(Janota, Lynce, and Marques-Silva 2015) carried out numerous evaluations of different backbones extraction Algorithms. It shows that Core Based algorithms is more consistent and efficiency for large scale instances. Hence, we compared MEB against CB from the st and sc aspects. In order to present an overall evaluations, we focus on the following cluster of benchmarks: (1) Maximal Satisfiable Subset(MSS) from unsatisfiable instances. (2) Satisfiable instances from SAT competition[1]. (3) Easy satisfiable instances from SATLIB[2].

The reason we choose MSS instances is that due to the theory of satisfiability, the density of backbones in MSS instances will be higher on average. It helps reveal how MEB and CB performances with a higher density backbones. Easy satisfiable instances are chosen to apprise the total SAT calls number without a possibility of timeout or memory leak. In other words, we want to make sure that each instances in this cluster is easy enough for MEB and CB to get results in blink. The selected satisfiable instances are all from industrial and is guarantied that both MEB and CB is able to finish computing under the resource limits.

MSS Extraction Experimental Results

In this section, the effectiveness of MEB and CB for MSS instances are presented.

Definition 3 (Maximal Satisfiability Sub-formulae(MSS))**.** *Given an unsatisfied propositional formula Φ, a sub-formula $\Phi' \subseteq \Phi$ is called a Maximal Satisfiability sub-formula (MSS) of Φ iff Φ' is satisfiable and $\Phi' \wedge (\phi \in (\Phi \setminus \Phi'))$ is unsatisfiable.*

MSS instances are obtained from UUF250 family using the enumeration function of LBX(Mencıa, Previti, and Marques-Silva 2015) tool. UUF250 family are formulae that have 250 variables and 1065 clauses. The fixed number of variables and clauses of UUF250 family sustains unbiased results. There are two reasons that we choose UUF250 family. First of all, the upper bound of variables and clauses number are fixed. It means that the difficulty of each instances are similar. Moreover, the average SAT call time is more than one second. It means that earlier test per variables approach are not efficiency enough. Due to the time and memory limit, it's hard for LBX to enumerate every MSS formula, since MEB suits for dense backbones formulae, we prefer to choose MSS with higher backbones percentage. With more constraints, it's more likely to generate

[1] http://www.satcompetition.org/

[2] http://www.cs.ubc.ca/ hoos/SATLIB/benchm.html

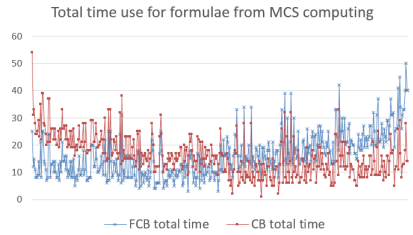

Figure 2: Total time use for formulae from MCS computing

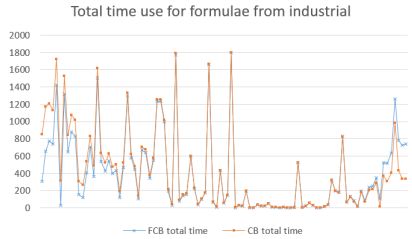

Figure 3: Total time use for formulae from industrial

more backbones, therefore, longer MSS formulae are chosen to form MSS benchmark.

Figure 2 describes a comparison of SAT call time is conducted between CB and MEB approach, indicating the spent CPU time on the y-axis. The first observation is that, for most of the formulae, MEB spent less CPU time than CB does. More precisely, for every formula in UUF250 family, there exists a MSS instance, such that MEB needs less time than CB does to compute backbones. However, the SAT call numbers of MEB is not always less than CB does. This means that MEB approach not only able to reduce the SAT call numbers, but also able to simplify SAT calls in each iteration. This can be explained that for high density backbones formulae, MEB can gain by excluding non-backbones ahead in linear $O(n^2)$ time.

Industrial Experimental Results

For the industrial part, most of the instances were drawn from hardware and software verification. The selection is motivated by the goals of finding instances that contains relatively less clauses and variables and takes a bit longer for SAT solvers to give an implicant. We try to focus on the hard SAT instances because for instances that are easy for SAT solvers, the trivial Test by Variables will be efficiency enough.

In Figure 3, the total SAT solver time of 104 formulae from industrial track of SAT competition are plotted. The backbones in industrial and practical instances are usually sparse. The experiments results indicate that for most formulae, MEB doesn't need more time than CB does. For some of the formulae, CB need more time than MEB does. It means that in the aspect of time cost, MEB and CB are compatible. The following experiments will show that MEB needs less SAT calls number than CB does on average.

This is because that both CB and MEB requires iterations of SAT solver calls. The SAT solvers computation spent most of CPU time in backbones extraction. If the CPU time of each SAT solver calls is relatively short, there are less possibilities that the instance will become easier. In general, there are more than one model for a satisfiable formula, CB will performs really fast if the model it got from MINISAT randomly suit the algorithm quite well, i.e., for most of the iterations, MINISAT always returns a core that only have one literal. In such cases, CB is faster than MEB.

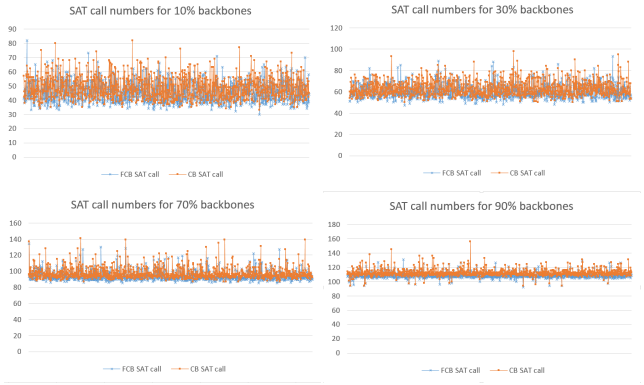

Figure 4: Total SAT calls number for fixed percentages of backbones formulae

Fixed Percentage Experiments Results

For the fixed percentage part, we take advantages from the CBS family with a fixed backbones percentage of 10%, 30%, 70% and 90% respectively, also from SATLIB. For each group of the fixed backbone percentage instances, there are 1000 instances in the group.

With the simply instances, it's sufficient to evaluate the efficiency of algorithms by only calculate the SAT calls number. Figure 4 illustrates the SAT call numbers needed by MEB and CB for different percentages. It's obvious that for most of the instances, MEB need less SAT call numbers than CB does, despite the percentage of backbones. It proves that MEB is compatible to CB no matter the backbones are dense or not from the SAT calls number perspective.

The results reveal that MEB is as efficiency as CB in general. For dense backbones formulae, MEB needs less total SAT solver time. The experiment results present that MEB outperforms CB with MSS formulae that generated from unsatisfiable formulae. It's because that MEB is more efficient when dealing with dense backbones. As a result, for dense backbones, MEB is better, no matter the formula is from MSS computing, practical verification or crafted. As discussed above, for unsatisfiable formulae, the density of backbones will increase simultaneously with the size of MSS formula. In the application of hardware model checking, planning and fault localization, the backbones are dense. With a proper use of other techniques, MEB will reach better performance in application.

Conclusion

This paper develops improvements to algorithms for backbones extraction. It's devises new preprocessing techniques for backbones extraction by computing under-approximation of backbones and non-backbones. In addition, the paper implemented FCB and compared with CB with formulae from MCSes extraction, practical applications and 4 groups of formulae with a fixed percentage of backbones.

The experimental results indicate that FCB is able to reduce the SAT calls number as well as shorten the CPU time cost by a single SAT call. However, for some of the formulae, CB needs less SAT calls than FCB does, which opening a possibility for portfolio-based solvers. In other words, a suitable model will reduce the SAT calls number for both CB and FCB.

In the future, we plan to explore more practical applications for backbones, such as hardware model checking, SAT model counting and SAT model enumeration.

References

[Bollobás et al. 2001] Bollobás, B.; Borgs, C.; Chayes, J. T.; Kim, J. H.; and Wilson, D. B. 2001. The scaling window of the 2-sat transition. *Random Structures & Algorithms* 18(3):201–256.

[Culberson and Gent 2001] Culberson, J., and Gent, I. 2001. Frozen development in graph coloring. *Theoretical computer science* 265(1):227–264.

[Dubois and Dequen 2001] Dubois, O., and Dequen, G. 2001. A backbone-search heuristic for efficient solving of hard 3-sat formulae. In *IJCAI*, volume 1, 248–253.

[En and Srensson 2004] En, N., and Srensson, N. 2004. *An Extensible SAT-solver*. Springer Berlin Heidelberg.

[Janota, Lynce, and Marques-Silva 2015] Janota, M.; Lynce, I.; and Marques-Silva, J. 2015. Algorithms for computing backbones of propositional formulae. *AI Commun.* 28(2):161–177.

[Janota 2010] Janota, M. 2010. *SAT Solving in Interactive Configuration*. Ph.D. Dissertation, University College Dublin.

[Kilby et al. 2005a] Kilby, P.; Slaney, J.; Thiébaux, S.; Walsh, T.; et al. 2005a. Backbones and backdoors in satisfiability. In *AAAI*, volume 5, 1368–1373.

[Kilby et al. 2005b] Kilby, P.; Slaney, J.; Walsh, T.; et al. 2005b. The backbone of the travelling salesperson. In *IJCAI*, 175–180.

[Li, Ma, and Zhou 2009] Li, K.; Ma, H.; and Zhou, H. 2009. From one solution of a 3-satisfiability formula to a solution cluster: frozen variables and entropy, 2009. *Physical Review E* 79:031102.

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

[Menaï 2005] Menaï, M. E. B. 2005. A two-phase backbone-based search heuristic for partial max-sat–an initial investigation. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 681–684. Springer.

[Mencıa, Previti, and Marques-Silva 2015] Mencıa, C.; Previti, A.; and Marques-Silva, J. 2015. Literal-based mcs extraction. In *IJCAI*, volume 15, 1973–1979.

[Parisi 2003] Parisi, G. 2003. On local equilibrium equations for clustering states. *Computer Science*.

[Ravi and Somenzi 2004] Ravi, K., and Somenzi, F. 2004. Minimal assignments for bounded model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 31–45. Springer.

[Walsh and Slaney 2001] Walsh, T., and Slaney, J. 2001. Backbones in optimization and approximation. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*.

[Zhang and Looks 2005] Zhang, W., and Looks, M. 2005. A novel local search algorithm for the traveling salesman problem that exploits backbones. In *IJCAI*, 343–350.

[Zhang, Rangan, and Looks 2003] Zhang, W.; Rangan, A.; and Looks, M. 2003. Backbone guided local search for maximum satisfiability. In *IJCAI*, 1179–1186. Citeseer.

[Zhu et al. 2011] Zhu, C. S.; Weissenbacher, G.; Sethi, D.; and Malik, S. 2011. Sat-based techniques for determining backbones for post-silicon fault localisation. In *IEEE International High Level Design Validation and Test Workshop*, 84–91.