# 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 a whitening-based algorithm to compute an approximation of backbone. The exact backbone is computed by applying iterative test backbone on the approximations. We implemented our approach in a tool BONE and conducted experiments on instances from Industrial and Random tracks of SAT Competitions between 2002 and 2015. Experimental results demonstrate that BONE is comparable to the state-of-the-art tool cb100. Moreover, BONE 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 $\Phi$, an truth assignment is a map from boolean variables to literals appeared in $\Phi$. In an assignment, a literal can only be assigned as TRUE or FALSE. Given an assignment $\lambda$, if there exist at least one literal in a clause $\phi \in \Phi$ that has been assigned as TRUE according to $\lambda$, $\phi$ is called to be satisfied by $\lambda$. If there exists a $\lambda$ that satisfy every clause $\phi \in \Phi$, $\lambda$ is a model or a satisfied assignment of $\Phi$. Backbones of a propositional formula $\Phi$ is a cluster of literals that are TRUE in every model of $\Phi$(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 S?rensson 2004) for most of them. Implicant enumeration(McMillan 2002; Ravi and Somenzi 2004) enumerates implicants of formula $\Phi$ one by one and updates the backbones estimation in each iteration. The negation of an implicant is added to $\Phi$ in order to avoid finding the same implicant. For an implicant $\lambda$, the negation of $\lambda$ 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 $\Phi$ 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 $\Phi_{\mathsf{BL}}$. The negation of $\Phi_{\mathsf{BL}}$ 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 $\Phi$ and formed $\Phi'$. If $\Phi'$ 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 $\Phi'$ is not satisfiable any longer. Along with the estimation, the clauses number of $\Phi'$ is monotone increasing due to the continuously disjunction in each iteration, which dramatically promote the complexity of $\Phi'$. 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 $\Phi$ 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 S?rensson 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 $\lambda$ 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 $\mathsf{HDBS}$, respectively. For each literals in $\mathsf{HDBS}$, 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 $\mathsf{HDBS}$ 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* $\phi$ 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* $\Phi$ 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 $\Phi$ 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 $\Phi$, let $\mathsf{var}(\Phi)$ (resp. $\mathsf{lit}(\Phi)$ and $\mathsf{cls}(\Phi)$) denote the set of variables (resp. literals and clauses) used in $\Phi$. We use $\neg\mathsf{lit}(\Phi)$ to denote the set $\{\neg l \mid l \in \mathsf{lit}(\Phi)\}$. The *size* $|\Phi|$ of $\Phi$ is the number of literals of $\Phi$. Given a formula $\Phi$ 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 $\lambda : \mathcal{X} \rightarrow \{0, 1\}$, where 1 (resp. 0) denotes true (resp. false). Given an assignment $\lambda$, let $\lambda[x \mapsto b]$ for some $b \in \{0, 1\}$ be the assignment which is equal to $\lambda$ except for $\lambda[x \mapsto b](x) = b$. An assignment $\lambda$ *satisfies* a formula $\Phi$, denoted by $\lambda \models \Phi$, iff assigning $\lambda(x)$ to $x$ for $x \in \mathsf{var}(\Phi)$ makes $\Phi$ true. An assignment $\lambda$ is a *model* of $\Phi$ if $\lambda \models \Phi$. A formula $\Phi$ is *satisfiable* iff there exists an assignment $\lambda$ such that $\lambda \models \Phi$. Given a formula $\Phi$, the *satisfiability problem* is to decide whether $\Phi$ is satisfiable or not.

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

It is known that the backbone $\mathsf{BL}(\Phi)$ for each formula $\Phi$ is unique and a backbone literal $l$ of $\Phi$ 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 $\Phi$ and a literal $l$, deciding whether $l$ is a backbone literal is co-NP-complete.*

**Definition 2** (Satisfied literal)**.** *Given a model $\lambda$ of the formula $\Phi$ and a clause $\phi \in \Phi$, for each literal $l \in \phi$, $l$ is a* satisfied literal *of $\phi$ iff $\lambda \models l$. $l$ is a* unique satisfied literal *of $\phi$ 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

In this section, we present the overview of our approach BONE which is depicted in Figure 1. BONE comprises three

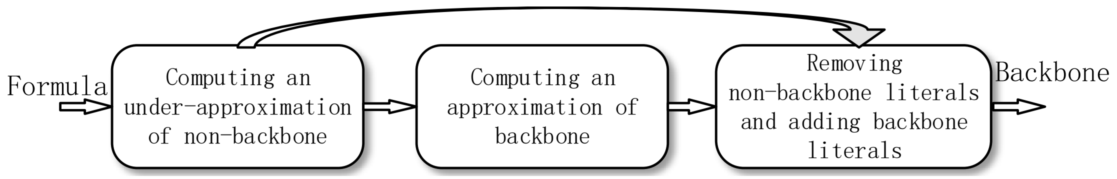
Formula
Computing an under-approximation of non-backbone
Computing an approximation of backbone
Removing non-backbone literals and adding backbone literals
Backbone

Figure 1: Overview of our approach

components. Taking a satisfiable formula $\Phi$ as an input, Bone first computes an under-approximation $\overline{\mathsf{BL}}_{\downharpoonleft}(\Phi) \subseteq \overline{\mathsf{BL}}(\Phi)$ of the non-backbone of $\Phi$. Then, Bone computes an approximation $\mathsf{BL}_{\approx}(\Phi)$ of the backbone of $\Phi$ based on the set $\overline{\mathsf{BL}}_{\downharpoonleft}(\Phi)$, where each literal in $l \in \mathsf{BL}_{\approx}(\Phi)$ has a high possibility to be a backbone literal of $\Phi$. Finally, Bone removes non-backbone literals from $\mathsf{BL}_{\approx}(\Phi)$ and adds backbone literals into $\mathsf{BL}_{\approx}(\Phi)$ to compute the exact backbone of $\Phi$.

**Computing an under-approximation of non-backbone.** Given a satisfiable formula $\Phi$, we first compute a model $\lambda$ of $\Phi$ by calling a SAT solver. From the model $\lambda$, we can compute a base under-approximation of non-backbone. Later, we apply a Greedy-based algorithm to add more non-backbone literals into the base under-approximation, which results in the under-approximation $\overline{\mathsf{BL}}_{\downharpoonleft}(\Phi)$.

**Computing an approximation of backbone.** At this step, we apply an improved Whitening-based algorithm to compute the approximation $\mathsf{BL}_{\approx}(\Phi)$. The improved Whitening-based algorithm is an extension of the Whitening-based algorithm proposed by (Li, Ma, and Zhou 2009) which was used to compute *frozen variables*, that are the variables whose values are fixed in a non-singleton set of models. We use the Whitening-based algorithm of (Li, Ma, and Zhou 2009) to compute the approximation $\mathsf{BL}_{\approx}(\Phi)$ in which some non-backbone literals are removed by a heuristic approach.

**Removing non-backbone literals and adding backbone literals.** Finally, we first iteratively select one literal $l$ from $\mathsf{BL}_{\approx}(\Phi) \setminus \overline{\mathsf{BL}}_{\downharpoonleft}(\Phi)$ such that $\lambda \models \neg l$ and test whether $l$ is a backbone literal or not by checking the satisfiability of $\Phi \wedge l$. Intuitively, literals in $\mathsf{BL}_{\approx}(\Phi) \setminus \overline{\mathsf{BL}}_{\downharpoonleft}(\Phi)$ have a high probability to be backbone literals. If $\Phi \wedge l$ is satisfiable, then $l$ is a non-backbone literal. Otherwise, $l$ is a backbone literal. Then, we do the same testing for literals from $\mathsf{lit}(\Phi) \setminus (\overline{\mathsf{BL}}_{\downharpoonleft}(\Phi) \cup \mathsf{BL}_{\approx}(\Phi))$.

## Computing $\overline{\mathsf{BL}}_{\downharpoonleft}(\Phi)$

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

### An Naïve Algorithm

Definition. Lemma. Intuition. Proof.

**Definition 3** (Free Literals of a Given Model)**.** *Given a formula $\Phi$, a model $\lambda \models \Phi$, a free literal $l \in \mathit{lit}(\Phi)$ is a literal such that a new model will generated by complementing $l$, i.e., $\lambda[l \mapsto \neg l] \models \Phi$.*

**Lemma 1** (Free Literals)**.** *Given a formula $\Phi$, a model $\lambda \models \Phi$, a literal $l \in \mathit{lit}(\Phi)$ is a* free literal *iff $\forall \phi \in \Phi, l \in \phi \wedge \lambda \models l \implies \exists l' \in \phi, l' \neq l \wedge \lambda \models l'$.*

The satisfiable problem trying to find an assignment that satisfy each clause in a given formula. For a clause that satisfied by the given model, it will change to unsatisfiable or maintain satisfiable when the model changes. We try to make every clause maintains satisfiable and generate a new model by negating the assignment of only one literal.It's trivial that clauses will maintain satisfiable when the complementing literal doesn't appear in the clause. For clauses that contain the changing literal, satisfiable will be maintained iff there is at least another literal which continues assigning the clause to TRUE. Therefore, if a literal only satisfy clauses that have at least two satisfied literals, it's safe to conclude that it's a free literal.

According to the definition of backbone, it's obvious that the set of free literals is a subset of $\overline{\mathsf{BL}}(\Phi)$.

*Proof.* Given a formula $\Phi$, a model $\lambda \models \Phi$. Suppose a literal $l \in \mathsf{lit}(\Phi)$ is a free literal. If $l$ never shows up in any clause (trivial), then there is no such $\phi \in \Phi$ such that $l \in \phi \wedge \lambda \models l$, $\exists l' \in \phi, l' \neq l \wedge \lambda \models l'$ always holds. If $l$ shows up in some clauses, then there is a new model $\lambda[l \mapsto \neg l] \models \Phi$, no clause changes to unsatisfiable because of the changing of $l$. It means that $\forall \phi \in \Phi$, if $l \in \phi$, there must exists another literal $l' \neg l$, assigning $\phi$ to TRUE. Therefore, $\forall \phi \in \Phi, l \in \phi \wedge \lambda \models l \implies \exists l' \in \phi, l' \neq l \wedge \lambda \models l'$.

Suppose a literal $l \in \mathsf{lit}(\Phi)$, if $\forall \phi \in \Phi, l \in \phi \wedge \lambda \models l \implies \exists l' \in \phi, l' \neq l \wedge \lambda \models l'$, it means that for every clause that containing $l$, the satisfiable will maintain when changing $l$ to its negation by $l'$. Therefore, there must exist a new model $\lambda[l \mapsto \neg l] \models \Phi$. □

### Heuristic-based Algorithm

We apply a Greedy-Based Algorithm with different heuristic searching to extend the result of free literals and refereed as $\overline{\mathsf{BL}}_{\downharpoonleft}$. We leverages Greedy to depth first searches. The quota we choose are frequency represented by the number of appearance of a literal and the entropy represented by the ratio of appearance number of a literal to the total length of clauses that contains the literal. Each quota guided a Greedy search with maximal or minimal value.

The algorithm take the free literals of a given model as input and assigned to $\overline{\mathsf{BL}}_{\downharpoonleft}$ as shown in Line 1. From Line 2

**Algorithm 1:** Heuristic extension for computing $\overline{\mathsf{BL}}_u$

```
Input  : $\Phi$: a formula, $\lambda$: a model, free literals of $\lambda$
Output: $\overline{\mathsf{BL}}_\lfloor$: under-approximation of non-backbones
1 $\overline{\mathsf{BL}}_\lfloor$:=free literals of $\Phi$ to a given model $\lambda$;
2 repeat
3     select a list of literals using Greedy heuristic;
4     change the selected literal one by one to generate a
      new model;
5     add new free literals to $\overline{\mathsf{BL}}_\lfloor$.
6 until No Update of $\overline{\mathsf{BL}}_\lfloor$;
7 return $\overline{\mathsf{BL}}_\lfloor$;
```

to Line 6, $\overline{\mathsf{BL}}_\lfloor$ is expanded. Different heuristic are chosen at Line 3 to select a list of literals and changing the assignment of them with a increased or decreased order.

In order to generate virous models from a given model, we need to negate more literals. We generate consecutive models by changing the assignments of literals one by one. Since enumerating every model of a satisfied formula is known as a NP hard problem, many heuristic strategies are applied to model generation. The selection of literals paly a vital role in it, with a different changing decisions, different models are generated. We take the idea of term frequency from statistic research, namely Term Frequency-Inverse Document Frequency(TF-IDF). In our context, literals are regarded as terms, clauses are regarded as documents. The frequency of terms are represented as the appearance number of literals. The inverse document frequency of a literal is represented as the ratio of appearance number to the total length of clauses that contained the literal.

**Lemma 2.** *The result of Algorithm 1 is a subset of $\overline{BL}$.*

*Proof.* For every model generated during the algorithms, the free literals is added to the result set of Algorithm 1. Since free literals is a subset of $\overline{\mathsf{BL}}$, the result is also a subset of $\overline{\mathsf{BL}}$. □

## Computing $\mathsf{BL}_\approx(\Phi)$

In this section, we improve Whitening Algorithm to compute the approximation of backbone, refereed as $\mathsf{BL}_\approx$. Whitening Algorithm was proposed in (Culberson and Gent 2001) originated from the coloring problem of graphs. Some non-backbone literals are removed from the result of Whitening Algorithm using heuristic strategy. We consider the community structure of a formula. We construct a community structure graph, with literals as vertexes. If two literals are in the same clause, there is a edge between them. We then initialized $\mathsf{BL}_\approx(\Phi)$ with the complementary set of the result of Whitening Algorithm. For every literal is $\mathsf{BL}_\approx(\Phi)$, if there exists a path in the community structure graph such that started with a literal $l$ and ended up with the negation of $l$, and every vertex literal in the path is in $\mathsf{BL}_\approx(\Phi)$ [something related to global property and fairness], then it's possible that new models can be generated by changing the assignment every literal on the path.

**Algorithm 2:** Backbones approximation of $\Phi$

```
Input  : $\Phi$: a formula, $\overline{\mathsf{BL}}_\lfloor$: under-approximation of
         non-backbone
Output: $\mathsf{BL}_\approx(\Phi)$: backbone approximation of $\Phi$
1  $\mathsf{BL}_\approx := \mathsf{lit}(\Phi) \setminus \mathsf{Whitening}(\Phi, \overline{\mathsf{BL}}_\lfloor)$;
2  for $x \in \mathsf{BL}_\approx(\Phi)$ do
3      $k := 1$;
4      $\Phi_x^0 := \{x\}$;
5      repeat
6          $\Phi_x^k := \{\phi \in \Phi \mid \neg\mathsf{lit}(\Phi_x^{k-1}) \in \phi\}$;
7          if $\neg x \in \mathit{lit}(\Phi_x^k)$ then
8              $\mathsf{BL}_\approx(\Phi) := \mathsf{BL}_\approx(\Phi) \setminus \{x\}$;
9              break;
10         k++;
11     until $\Phi_{l_x}^k := \Phi$;
12 return $\mathsf{BL}_\approx(\Phi)$;
```

In Line 1, Whitening Algorithms is applied with a given formula $\Phi$ and the under-approximation of non-backbone obtained in Algorithm 1. From Line 2 to Line 12, a heuristic searching approach is applied to search paths in the community structure graph. $\Phi_x^k$ represents the $i_th$ vertex literal in a path starting from literal x. Only clauses that contain the negation of current vertex literal will be explored to reduce the searching space as shown in Line 6.

With a community structure graph, the dependency between literal and clauses can be obtained. However, the graph is usually too complex to analysis. Therefore, we only manage to search paths that may indicated a possibility of changing assignments of multi literals at the same time.

**Example 1.** *Given a formula, suppose $\phi_1$, $\phi_2$, $\phi_3$ are in $\Phi$, suppose $x_1$, $\neg x_1$, $x_2$, $\neg x_2$, $x_3$, $\neg x_3 \in \mathsf{BL}_\approx$, $x_1 \in \phi_1, \neg x_2 \in \phi_1$, $x_2 \in \phi_2, \neg x_3 \in \phi_2$, $x_3 \in \phi_3, \neg x_1 \in \phi_3$. Given a model $\lambda$, 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 $\phi_1, \phi_2, \phi_3$ respectively. It can be observed that if $x_1$ is a non-backbone literal, in order to generate a $\lambda'$ that satisfy $\phi_1, \phi_2$ and $\phi_3$, $x_1, x_2, x_3$ have to complemented simultaneously.*

The paths that we found using Algorithm 2 fits the pattern that shown in previous example.

**Theorem 2.** *The complexity of Algorithm 2 from Line 2 to 12 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., $|\Phi_l| = l$. The complexity of Line 3, 4 is $O(1)$. There is a loop of literals that in $\mathsf{BL}_\approx$. For the worst case, Line 12 won't be executed until every literal of $\mathsf{BL}_\approx$ had been visited. In the first iteration of the loop, the complexity of Line 6 is $O(k * l)$. In the second iteration, the complexity of Line 6 is $O(k * l * l)$. In the $i^{th}$ iteration of the loop, the complexity of Line 6 is $O(k * l^i)$. Therefore, the complexity of Line 6 is $O(k*l^n)$. The complexity of Line 7 to Line 10 is

$O(1)$. As discussed above, the complexity of Line 2 to Line 12 in Algorithm 2 is $O(k * l^i + m * k + m * l)$. $\square$

## Experiments results

Our tool MEB (Multi Estimation Based Backbones Extraction) was built with C++ using minisat(En and S?rensson 2004) as the underlying SAT solver. It contains the $\overline{\mathsf{BL}}_u$ computing and $\mathsf{HDBS}$ 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 4** (Maximal Satisfiability Sub-formulae(MSS))**.** *Given an unsatisfied propositional formula $\Phi$, a sub-formula $\Phi' \subseteq \Phi$ is called a Maximal Satisfiability sub-formula (MSS) of $\Phi$ iff $\Phi'$ 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 more backbones, therefore, longer MSS formulae are chosen to form MSS benchmark.

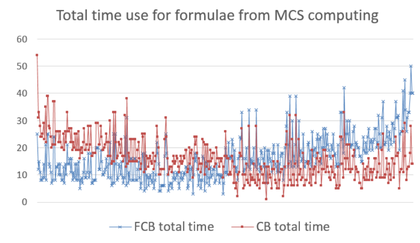

Figure 2: Total time use for formulae from MCS computing

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

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

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

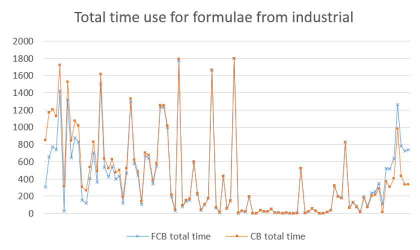

Figure 3: Total time use for formulae from industrial

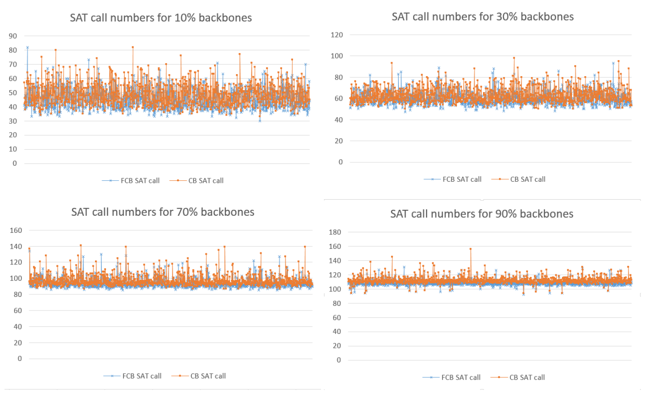

Figure 4: Total SAT calls number for fixed percentages of backbones formulae

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.

## 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 S?rensson 2004] En, N., and S?rensson, 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.

[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.