On Computing Generalized Backbones

Alessandro Previti*, Alexey Ignatiev†,‡, Matti Järvisalo*, and Joao Marques-Silva†
*HIIT, Department of Computer Science, University of Helsinki, Finland
†LASIGE, Faculty of Science, University of Lisbon, Portugal
‡ISDCT SB RAS, Irkutsk, Russia

***Abstract*—The concept of backbone variables, i.e., variables that take the same value in all solutions—or, equivalently, never take a specific value—finds various important applications in the context of Boolean satisfiability (SAT), motivating the development of efficient algorithms for determining the set of backbone variables of a given propositional formula. Notably, this problem surpasses the complexity of merely deciding satisfiability. In this work we consider generalizations of the concept of backbones in SAT to non-binary (and potentially infinite) domain constraint satisfaction problems. Specifically, we propose a natural generalization of backbones to the context of satisfiability modulo theories (SMT), applicable to a range of different theories as well as CSPs in general, and provide two generic algorithms for determining the backbone in this general context. As two concrete instantiations, we focus on two central SMT theories, the theory of linear integer arithmetic (LIA) with infinite integer domains, and the theory of bit vectors (BV), and empirically evaluate the potential of the proposed algorithms on both LIA and BV instances.**

I. Introduction

The concept of *backbone variables* [1]–[3], i.e., variables that take the same value in all solutions, has been studied extensively in connection with different combinatorial problems [2]–[16] in the context of Boolean satisfiability (SAT) [17]–[21] and also in the context of constraint satisfaction problems (CSPs) [22]–[25]. In recent years, backbones have been used in a number of relevant practical applications [26]–[30]. Backbone variables provide important information. For example, the existence of backbone variables precludes the existence of supersolutions [31]. Moreover, backbone variables of SAT instances encoding real-world problem instances can have various kinds of meaningful interpretations depending on the context. For some examples, in interactive product configuration identifying a backbone variable during the configuration process can prevent users from attempting to further construct unavailable configurations [28]; backbones can represent faults in fault localization in integrated circuits [26], [27]; in causal structure discovery, backbone variables can represent equivalence classes of causal structures [32]; and in abstract argumentation, backbones can be used to realize the so-called ideal semantics [33].

Although the problem of determining the backbone (variables) of a given propositional formula surpasses the complexity of merely deciding satisfiability, motivated by the wide range of applications efficient algorithms for determining the set of backbone variables of a given propositional formula have been recently developed [20]. In the context of CSPs, a notion of backbones has been proposed under the name of frozen/fixable variables [5], [22], [23], defined as variables that take the same value in all solutions. However, in comparison with the CSP case, the definition of backbone variables we propose in this paper is more general. Although our proposal has connections to earlier work on minimal constraint networks [34] and minimal labelings in qualitative constraints networks [35], a major difference is that we take a variable-oriented view as opposed to a constraint-oriented view.

Specifically, in this work we consider generalizations of the concept of backbones in SAT to non-binary (and potentially infinite) domain constraint satisfaction problems. Differently from the concept of frozen variables, we propose a more generic concept of backbones. Instead of requiring a variable to take a fixed value in all solutions (as is the case e.g. in the traditional notion of backbone literals in the context of SAT), we seek for the greatest domain reduction of individual variables, and more generally consider a variable to be contained in the backbone if its domain can be reduced while maintaining the set of solutions. This proposed natural generalization of backbones applies e.g. to the context of satisfiability modulo theories (SMT) for a range of different theories, as well as CSPs in general.

From the computational perspective, we provide two generic algorithms for determining the backbone in this general context. As two concrete instantiations in the realm of SMT, we focus on two SMT theories: the theory of linear integer arithmetic (LIA)—with infinite integer domains—and the theory of bit vectors (BV). Deciding the satisfiability of a system of linear inequalities is important in different settings ranging from formal verification to scheduling [36], [37]. Bit-vector formulas as well find many applications, including bounded model checking, analysis of hardware circuits, static analysis, and test generation [37].

We empirically evaluate the potential of the proposed algorithms on both LIA and BV instances. To the best of our knowledge, no algorithms in the generality as presented here have been previously proposed; in the context of SMT, backbones have to the best of our knowledge been studied only in very specific context, see e.g. [19].

It should be noted that, while efficient algorithms for computing the backbone of SAT instances by iteratively calling a SAT solver have been developed, such algorithms are not directly applicable to non-binary domains. Specifically, while several SAT encodings for finite-domain CSPs with non-binary variables have been developed, such encodings are not generally applicable for computing the backbone of

the original CSP under the generalized notion considered here. For instance, the so-called *log encoding* [38]. takes Boolean variables for representing the individual bits of the domain value assigned to a variable. A similar approach is also standard practice in BV solvers many of which *bit-blast* the the BV formula into a SAT instance and apply a SAT solver to determine satisfiability [37]. Hence, applying a SAT backbone computation algorithm on the resulting SAT instance will only detect frozen bits in the representation of the domain values, instead of precise information on the admissible domain values.

The rest of this paper is organized as follows. After necessary preliminaries (Section II), we introduce the notion of generalized backbones (Section III), and propose two generic algorithms for determining generalized backbones (Section IV). Before the conclusions in Section VI, we present results from an empirical evaluation of the two algorithms implemented in the contexts of linear integer arithmetic and bit-vector theories in SMT (Section V).

II. Preliminaries

We will propose a very general definition of backbones, covering a range of constraint satisfaction formalisms, as well as two generic algorithms for computing these generalized backbones. For concreteness, we will illustrate the concepts and algorithms in the context of SMT, and more precisely, for the theories of LIA and BV. Before proceeding with the notion of generalized backbones and algorithms for computing them, we will hence now provide background on SMT, giving definitions valid for LIA and BV theories.

A first-order theory $\mathcal{T}$ is a set of first-order sentences ([36], [37], [39]) over a signature $\mathcal{S}$, where the signature $\mathcal{S}$ specifies a set of predicate symbols, function symbols and constants. A first-order model $\mathcal{M}$ is a pair $\langle \mathcal{U}, \mathcal{I} \rangle$, where the set $\mathcal{U}$ represents a *universe*, and $\mathcal{I}$ represents an *interpretation* that assigns a semantic to every symbol in $\mathcal{S}$. In what follows, F denotes a first-order formula modulo a theory. $\mathit{Var}(F)$ denotes the set of variables (which are distinct from $\mathcal{S}$). Given a model $\mathcal{M}$, a *valuation* ω is a partial map from $\mathit{Var}(F)$ to $\mathcal{U}$. For simplicity, we assume $\mathit{Var}(F)$ to be the set of variables that occur free in formula F.

For a given valuation $\omega : \mathit{Var}(F) \to \mathcal{U}$, we write $\mathcal{M}, \omega \models F$ to indicate that the formula F is true, according to the usual semantic of first-order logic, in model $\mathcal{M}$, with ω giving the valuation of the free variables in F.

Definition 1. *Formula F is satisfiable modulo $\mathcal{T}$ when there exists a model $\mathcal{M} = \langle \mathcal{U}, \mathcal{I} \rangle$ of $\mathcal{T}$ and an assignment $\omega \in \mathit{Var}(F) \to \mathcal{U}$ such that $\mathcal{M}, \omega \models F$. The pair $\langle \mathcal{M}, \omega \rangle$ is a* satisfying assignment (SA) *for F.*

A concept used throughout the paper is that of *partial satisfying assignment*.

Definition 2. *A* partial satisfying assignment *for a formula F is a pair $\langle \mathcal{M}, \omega \rangle$, where $\mathcal{M}$ is a model and ω is a valuation over $\mathcal{M}$ such that $dom(\omega) \subseteq \mathit{Var}(F)$ and such that for any valuation of $\alpha \in \mathit{Var}(F) \setminus dom(\omega) \to \mathcal{U}$, we have that $\langle \mathcal{M}, (\omega \cup \alpha) \rangle$ is a satisfying assignment for $\mathcal{F}$.*

Definition 3. *A partial satisfying assignment $\langle \mathcal{M}, \omega \rangle$ for F is a* minimal satisfying assignment *(mSA) if for any valuation $\alpha \in \mathit{Var}(F) \to \mathcal{U}$ such that $dom(\alpha) \subset dom(\omega)$, the pair $\langle \mathcal{M}, \alpha \rangle$ is not a satisfying assignment for F. A* minimum satisfying assignment *(MSA) is an mSA of smallest size.*

Minimum and minimal satisfying assignments of $\neg F$ are referred to as *minimum falsifying assignment* (MFA) and *minimal falsifying assignment* (mFA) of F respectively.

III. Generalizing Backbones

The concept of the backbone, i.e., the set of backbone literals, of a propositional formula is a well-known concept in SAT. For the following, we will assume basic understanding of propositional formulas and satisfiability.

Definition 4 (Propositional backbones)**.** *The backbone B of a propositional formula is the set consisting of those literals in F which take the same value in all truth assignments satistying F. Each literal in B is a* backbone literal of F.

In this work, we aim for a general notion of backbone literals to constraint satisfaction problems over discrete, possibly infinite non-binary domains.

Assumption 1. *The definition of generalized backbones and the algorithms proposed in this work are essentially constraint agnostic, under the following assumptions on the constraint language.*

(i) Variable domains are a subset of the set of integers.

(ii) There is a constraint solver which can decide instances of the form $\exists X \forall Y . F(X, Y)$ for any sentence in the language.

Note that property (i) allows for both finite-domain variables, as well as infinite domains as long as there is a total order over the elements of the domain.

A key intuition behind the following definition of generalized backbones is that we define backbones via the largest set of domain values each of which appears in some solution of the constraint satisfaction problem instance.

Definition 5 (Generalized backbones)**.** *A variable x with domain $D(x)$ in a constraint satisfaction problem instance F is a* backbone *under the domain $D'(x) \subsetneq D(x)$ if and only if $D'(x)$ is the largest subset of $D(x)$ s.t. for each $v \in D'(x)$, F has a solution with $x = v$.*

When restricted to Boolean domains (i.e., $D(x) = \{\text{true}, \text{false}\}$), this definition is equivalent to the notion of backbone literals in the context of SAT (recall Definition 4) with $D'(x) = \text{true}$ or $D'(x) = \text{false}$.

A. Instantiations in SMT

While Definition 5 also captures a range of other classes of CSPs, we will in the following instantiate the definition and the general algorithms within the realm of SMT. To exemplify

how the algorithms work—and to empirically evaluate instantiations of the general algorithms in the context of central SMT theories—we will focus on the theory of quantifier-free linear integer arithmetic (LIA) and the theory of quantifier-free bit-vectors (BV). The theory of LIA is used to represent boolean combination of linear inequalities of the form $a_1x_1+...+a_nx_n+c \leq 0$ over the domain of integer. The theory of fixed sized bit-vectors (BV) represents a natural way for high-level reasoning about circuits and programs. Bit-vectors are fixed-length sequences of binary bits which are interpreted as unsigned or signed integers. The only predicate symbols in the BV theory are $\leq_u$ and $\leq_s$, interpreted as inequality of the unsigned and signed integer encoding, respectively. Function symbols include $+, \times, \div, \ \& \ , |, \ll, \gg$, and are intrepreted as addition, multiplication, unsigned division, bit-wise and, bit-wise or, left-shift, and right-shift, respectively.

Definition 6. *A variable var is a backbone variable for an SMT formula F if $\forall var \exists Y.F$ is false, where $Y = Var(F) \setminus \{var\}$, or alternatively, when $\exists var \forall Y.\neg F$ is true.*

Notice that since $\exists Y \forall var.F \rightarrow \forall var \exists Y.F$, whenever $\forall var \exists Y.F$ is false, also $\exists Y \forall var.F$ is false. This clearly shows that backbone variables cannot appear as *don't-care*. In this sense, backbone variables can be referred to as *necessary* variables, since without assigning them it is not possible to satisfy the formula. To be more precise they have to be part of every partial satisfying assignment. This relates to the case of propositional logic where backbone literals are part of every model. However, differently from the case of propositional logic, where backbone literals can be assigned just one value, here backbone variables can have a potentially infinite range of values.

Proposition 1. *If var is a backbone variable, then $\forall Var\ (F) \setminus \{var\}.\neg F$ is satisfiable and any value assigned to var is an MFA of F.*

This relates to the definition of backbone literal as unary prime implicate given in the context of Boolean satisfiability [21].

Example 1. *As an example, backbones in the context of LIA can take the form*

1) $x \geq l$,
2) $x \leq l$,

or a combination of them, i.e.,

1) $x \geq l \wedge x \leq m$ with $m > l$,
2) $x \leq l \vee x \geq m$ with $m > l$.

We can also have more than one interval, like $(x \geq l \wedge x \leq m) \vee (x \geq h \wedge x \leq n)$, with $m > l$, $n > h$ and $h > m$. As a special case we have x = l when $x \leq l \wedge x \geq l$. This kind of backbone variables are referred to as frozen/fixable variables [5], [22], [23] within the CSP community. A formula is unsatisfiable when for one variable we have $x \geq l \wedge x \leq m$ with $m < l$. ■

IV. Computing Generalized Backbones

We present two algorithms, BB-OPT and BB-Q, for computing backbone variables and the bounds of their domain.

Algorithm 1: GETBBVAR: computing all backbone variables

```
1 GETBBVAR(F)
2 bbvar = ∅
3 for var ∈ Var(F) do
4     Y ← Var(F) \ {var}
5     st ← SOLVE(∃var∀Y.¬F)
6     if st then
7         bbvar = bbvar ∪ {var}
8 return bbvar
```

Algorithm 2: BB-OPT: backbone extraction using an optimizer

```
1 BB-OPT(F)
2 bbvar ← GETBBVAR(F)
3 for var ∈ bbvar do
4     lb = GETLB(F)
5     ub = GETUB(F)
6     var_domain = GETBOUNDS(F, var, lb, ub)
7     print var_domain
```

We present here the two algorithms BB-OPT and BB-Q (Algorithm 2 and Algorithm 4, respectively) in the context of SMT theories without quantifiers. The approach can be generalized to quantified formulas, where the variables *var*, X and Y range over the free, non-quantified variables.

The first phase (Algorithm 1) is shared by the two algorithms. The first phase aims at identifying all the backbone variables. This is shown in Algorithm 1 which makes use of Definition 6. The second phase depends on the theory taken into consideration, since its aim is to compute the set of admissible values of each variable. Nevertheless, the second phase can be used with any constraint language respecting Assumption 1. BB-OPT and BB-Q (Algorithm 2 and Algorithm 4, respectively) work by computing the lower and upper bounds of each admissible interval. In what follows, lower and upper bounds may refer to the smallest and highest value of an interval of admissible values without gaps in-between, i.e., not necessarily of the entire domain. Although similar in spirit, the algorithms differ in the way each bound is identified. While the ability of BB-OPT (Algorithm 2) to compute lower and upper bounds relies on solving an optimization problem, BB-Q (Algorithm 4) only requires a theory solver, although it still requires quantification support. This makes BB-Q (Algorithm 4) more widely usable for theories for which solvers with optimization capabilities are not readily available.

A. BB-OPT

BB-OPT (Algorithm 2) starts by computing the set of all backbone variables (line 2). Then for each backbone variable the lower and upper bound are computed. This is done by solving an optimization problem of the form

Algorithm 3: GETBOUNDS subfunction: extracts the set of intervals containing admissible solutions

```
1  GetBounds(F, var, lb, ub)
2  Y ← Var(F) \ {var}
3  bounds ← ∅
4  st, m ← Solve(∀Y.¬F ∧ var > lb ∧ var < ub)
5  if st then
6      fv ← m[var]    // F falsified by var = fv
7      ub1 ← GetUB(F, var < fv)
8      b1 ← GetBounds(F, var, lb, ub1)
9      lb1 ← GetLB(F, var > fv)
10     b2 ← GetBounds(F, var, lb1, ub)
11     bounds ← b1 ∪ b2
12 else
13     interval ← {lb, ub}
14     bounds ← {interval}
15 return bounds
```

$$\min / \max var \text{ subject to } F, \tag{1}$$

where F is the input formula and var is a backbone variable. Note that a backbone variable may not have a lower or an upper bound. As an example, consider the formula $F = x < 5 \vee x > 10$. The lower and upper bound are then given as input to the function GETBOUNDS(F, var, lb, ub), which returns the set of admissible values. If no lower or upper bound exist for a variable, then $lb = -\infty$ and $ub = \infty$, respectively. Algorithm 3 works by interleaving the computation of MFAs with the extraction of lower and upper bounds. At each step, Algorithm 3 splits the interval received in input in two parts and calls recursively the function GETBOUNDS on each of them. The value that splits the interval is returned by the call SOLVE$(\forall Y.\neg F \wedge var > lb \wedge var < ub)$, where $Y = Var(F) \setminus \{var\}$. If the formula $\forall Y.\neg F \wedge var > lb \wedge var < ub$ is satisfiable, the value assigned to var is an MFA of F. The corresponding falsifying value is then stored in the variable fv (line 6 of Algorithm 3). This proves that not all the values between lb and ub are part of an admissible solution and that the interval received as input has to be refined. This is done through a call to GETLB (GETUB) that returns a lower (upper) bound subject to the additional constraint $var > fv$ ($var < fv$). The two functions GETLB and GETUB make use of an optimizer in order to provide a value. Otherwise, if $\forall Y.\neg F \wedge var > lb \wedge var < ub$ is unsatisfiable, it means that the set of values between lb and ub is an interval of admissible values for var.

Example 2. *Suppose we are given the LIA formula* $F = (x \leq 7 \vee (x \geq 11 \wedge x \leq 13) \vee (x \geq 16 \wedge x \leq 20)) \wedge (x + y \leq 0)$ *as input. The execution of Algorithm 3 on this input is summarized in Table I. The column* ***Input*** *gives the values received in input by the functions* SOLVE, GETLB *and* GETUB. *When the input has the form* $[lb, ub]$, *the function being called is* SOLVE. *When the input has the form* $var < val$ *or* $var > val$, *the function called is* GETUB *or* GETLB, *respectively. The column* ***Result*** *gives the value returned by the functions. In the case of* SOLVE, *when the formula is satisfiable, we write directly the value assigned to the* var. *The third column* ***Intervals to Test*** *contains the intervals that we still have to test, and the fourth column* ***Admissible Intervals*** *lists the set of intervals containing admissible solutions.* ■

B. BB-Q

In contrast to BB-OPT presented in Algorithm 2, BB-Q (see Algorithm 4) does not rely on an optimizer. Algorithm 4 is based on the observation that if an assignment $v = l$ is an upper bound (lower bound, resp.) of an interval of admissible values for variable v, then $v = l + 1$ ($l - 1$, resp.) is an MFA for F (i.e., it is an MSA for $\neg F$), which enables us to consider the formula

$$F' = F \wedge \forall Var(F_r) \setminus \{var_r\}.\neg F_r \wedge (var_r = var + 1), \tag{2}$$

where formula F_r is a duplicate of F with all variables renamed, i.e., for every variable var in formula F, there is the corresponding variable var_r in F_r. The idea of Algorithm 4 is to decide the formula F' for every backbone variable var individually. Observe that if F' is satisfiable, then each of its satisfying assignments returned by SOLVE(F') satisfies the following.

1) The assignment contains an assignment to variables $Var(F)$ satisfying F, which sets $var = l$.
2) The partial assignment $var_r = l + 1$ is an MSA of $\neg F^r$.

This means that no assignment assigning $var = l + 1$ satisfies F, since all the assignments with $var = l + 1$ falsify F (see condition 2 above and recall that an MSA of $\neg F$ is an MFA of F, and that F_r is a duplicate of F). Therefore, l is an upper bound of an interval of admissible values for var. The same technique can be used to find a lower bound by considering $var_r = var - 1$ in (2).

Observe that Algorithm 4 first computes a set of all backbone variables by calling Algorithm 1. For every backbone variable var, Algorithm 4 enumerates exhaustively all upper and lower bounds of the interval of admissible values for var (see line 4 and line 5). Once this is done, the lists of lower $lb\_lst = (lb_1, lb_2, \ldots)$ and upper $ub\_lst = (ub_1, ub_2, \ldots)$ bounds are *zipped*[1] into one list of pairs $((lb_1, ub_1), (lb_2, ub_2), \ldots)$ (see the corresponding subroutine call on line 6 detailed as Algorithm 6), each defining an interval $[lb_i, ub_i]$. The computed domain of the backbone variable comprises the union of such intervals.

The GETALLUB subroutine is detailed in Algorithm 5. It makes an extensive use of formula (2) (see line 3 of Algorithm 5) and makes a sequence of SMT oracle calls (line 6). All upper bound values computed are then blocked (line 11) and Algorithm 5 returns as soon as no more upper bound values can be extracted (line 8).

[1]For doing this, the standard *convolution* operation is applied used in a number of programming languages, e.g. in Ruby, Python, Haskell, etc. Given a tuple of sequences, convolution maps it into a sequence of tuples.

Table I: Example execution of Algorithm 3 on the formula $F = (x \leq 7 \vee (x \geq 11 \wedge x \leq 13) \vee (x \geq 16 \wedge x \leq 20)) \wedge (x + y \leq 0)$. The run is limited to the variable x.

Input	Result	Intervals to Test	Admissible Intervals
$(-\infty, 20]$	FV: 14	$[14, 20]$, $(-\infty, 14]$	$\emptyset$
$x < 14$	UB: 13	$[14, 20]$, $(-\infty, 13]$	$\emptyset$
$(-\infty, 13]$	FV: 8	$[14, 20]$, $[8, 13]$, $(-\infty, 8]$	$\emptyset$
$x < 8$	UB: 7	$[14, 20]$, $[8, 13]$, $(-\infty, 7]$	$\emptyset$
$(-\infty, 7]$	UNSAT	$[14, 20]$, $[8, 13]$	$(-\infty, 7]$
$x > 8$	LB: 11	$[14, 20]$, $[11, 13]$	$(-\infty, 7]$
$[11, 13]$	UNSAT	$[14, 20]$	$(-\infty, 7]$, $[11, 13]$
$x > 14$	LB: 16	$[16, 20]$	$(-\infty, 7]$, $[11, 13]$
$[16, 20]$	UNSAT	$\emptyset$	$(-\infty, 7]$, $[11, 13]$, $[16, 20]$

Algorithm 4: BB-Q: backbone extraction using quantifiers

```
1  BB-Q(F)
2  bbvar ← GetBBVar(F)
3  for var ∈ bbvar do
4      lb_lst = GetAllLB(F)
5      ub_lst = GetAllUB(F)
6      var_domain = ZipBounds(lb_lst, ub_lst)
7      print var_domain
```

Algorithm 5: GetAllUB subfunction: returns all the upper bounds of a backbone variable

```
1  GetAllUB(F, var)
2  F_r ← RenameVar(F)
3  F' = F ∧ ∀Var(F_r) \ {var_r}.¬F_r ∧ (var_r = var + 1)
4  ub_lst ← ∅
5  while true do
6      (st, m) ← Solve(F')
7      if not st then
8          break
9      val = m[var]
10     ub_lst = ub_lst ∪ {val}
11     F' = F' ∧ (var ≠ val)
12 return ub_lst
```

It is interesting to point out that our algorithms are able to identify intervals that are not possible to compute when the theory is encoded as a propositional formula and backbone literals are extracted from that. Suppose, for example, that we have a backbone variable x such that $x \geq 6$ and $x \leq 8$. So the set of admissible values for x is $\{6, 7, 8\}$. This can be represented in binary as $\{0110, 0111, 1000\}$, that clearly contains no backbone literals.

V. Experimental Results

We empirically evaluate instantiations of the two proposed algorithms, BB-OPT and BB-Q, for computing generalized backbones in the context of the quantifier-free SMT theories of linear integer arithmetic and bit vectors. For this, we

Algorithm 6: ZipBounds subfunction: convolves the lists of computed lower and upper bounds into a list of intervals

```
1  if lb_lst[0] > ub_lst[0] then
2      lb_lst ← {−∞} ∪ lb_lst
3  if lb_lst[−1] > ub_lst[−1] then
4      ub_lst ← ub_lst ∪ {∞}
5  intervals ← ∅
6  for i ∈ {1, ..., |lb_lst|} do
7      intervals ← intervals ∪ {(lb_i, ub_i)}
8  return intervals
```

Here and following the standard notation of most scripting programming languages (e.g. Bash, Perl, Ruby, Python, etc.), index $[-1]$ is used to denote the last element of a list.

implemented BB-OPT and BB-Q for both of these theories, using z3 [40] as the backend SMT solver. Here we present a comparison of the running time performance of the two proposed approaches. The experiments were run under Ubuntu Linux on Intel Xeon E5540 2.53-GHz processors with 32 GB of RAM. The per-instance time limit and memory limit was set to 1800 seconds and 4 GB, respectively.

A. Results on LIA Formulas

We evaluated the performance of the two algorithms on a total of 489 LIA instances. The instances were obtained from two sources: from the benchmark set proposed in [41], and from the benchmark set QF_LIA from the SMT-LIB benchmark library (http://smtlib.cs.uiowa.edu/benchmarks.shtml). The benchmark set proposed in [41] was generated by the program analysis tool Compass [42]. The instances considered are those on which at least one of the two algorithms, BB-OPT and BB-Q, was able to finish.

The plot in Figure 2, with the number of instances solved by the individual algorithms under different per-instance time limits, shows that BB-OPT outperforms BB-Q: BB-Q timeouts on 12 instances while BB-OPT is able to determining the backbone of each of the instance. The advantage of BB-OPT on BB-Q can be attributed to the hardness of the quantified formulas that BB-Q has to deal with. Or, at least in the context of LIA, the optimization algorithm implemented in z3

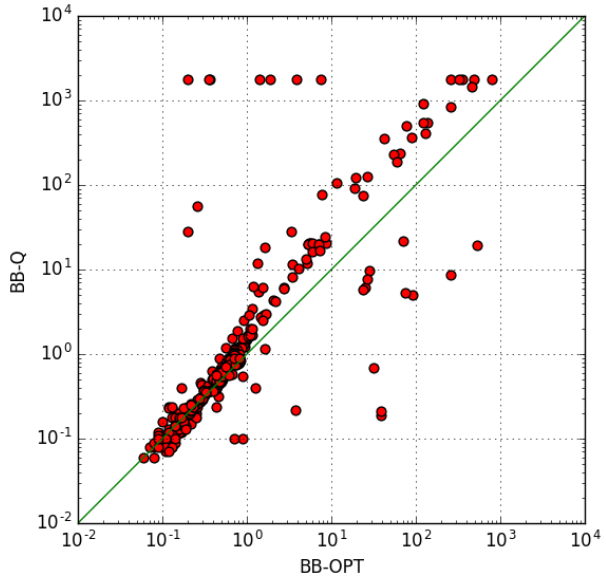

Figure 1: Comparison of the running times of BB-OPT and BB-Q on LIA formulas

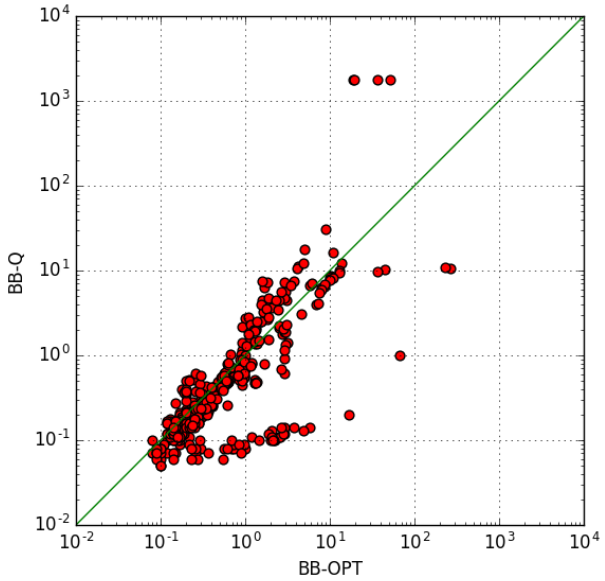

Figure 3: Comparison of the running times of BB-OPT and BB-Q on BV formulas

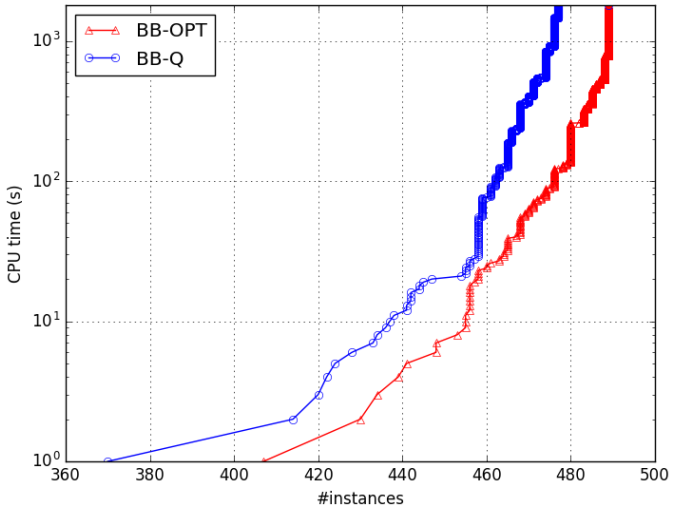

Figure 2: Number of instances solved by BB-OPT and BB-Q under different per-instance time limits on LIA formulas

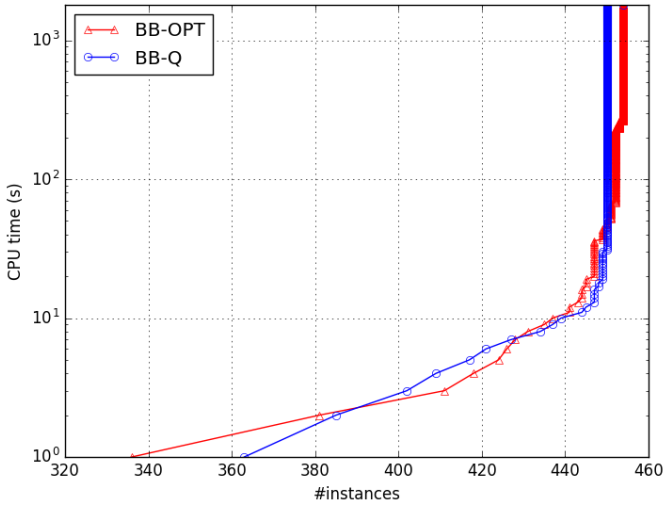

Figure 4: Number of instances solved by BB-OPT and BB-Q under different per-instance time limits on BV formulas

appears to perform more efficiently. The scatter plot depicted in Figure 1 highlights how on most of the instances BB-OPT outperforms BB-Q, apart from a few exceptions.

B. Results on BV Formulas

For evaluating the relative performance of BB-OPT and BB-Q on bit-vector formulas, we used a total of 454 instances from the benchmark set QF_BV from the SMT-LIB benchmark library (http://smtlib.cs.uiowa.edu/benchmarks.shtml). The final benchmark set includes those instances for which at least one of the two solvers was able to compute all the backbone variables within 1800 seconds. Figure 3 shows a comparison between BB-Q and BB-OPT. As one can see, the two solvers perform quite similarly on most of the instances. However, on a non negligible set of instances BB-Q is able to output the backbone in less time. On the other hand, BB-OPT exhibits a more robust and stable behaviour in terms of performance. This is particularly evident when looking at Figure 4. In particular, BB-Q timeouts on 4 instances, while BB-OPT is able to determine the backbone of each of them.

VI. Conclusions

We proposed a generalized notion of backbone variables, covering a range of constraint satisfaction languages, and proposed two generic algorithms for computing such generalized backbones. For concreteness, we illustrated the algorithms in the realm of SMT, implemented them for two central SMT theories, the quantified-free theories of linear integer arith-

metic and bit vectors, and empirically evaluated the relative performance of the algorithms on LIA and BV benchmarks.

Acknowledgments

This work was financially supported in part by Academy of Finland (grants 251170 COIN, 276412, 284591, 312662), Research Funds of the University of Helsinki, FCT funding of post-doctoral grant SFRH/BPD/120315/2016, and LASIGE Research Unit, ref. UID/CEC/00408/2013.

References

[1] J. Schneider, C. Froschhammer, I. Morgenstern, T. Husslein, and J. Singer, "Searching for backbones – an efficient parallel algorithm for the traveling salesman problem," *Computer Physics Communications*, vol. 96, pp. 173–188, 1996.

[2] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansk, "Determining computational complexity from characteristic 'phase transitions'," *Nature*, vol. 400, pp. 133–137, July 1999.

[3] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky, "2+p-SAT: Relation of typical-case complexity to the nature of the phase transition," *Random Structures and Algorithms*, vol. 15, no. 3-4, pp. 414–435, 1999.

[4] J. Singer, I. P. Gent, and A. Smaill, "Backbone fragility and the local search cost peak," *J. Artif. Intell. Res. (JAIR)*, vol. 12, pp. 235–270, 2000. [Online]. Available: http://dx.doi.org/10.1613/jair.711

[5] J. C. Culberson and I. P. Gent, "Frozen development in graph coloring," *Theoretical Computer Science*, vol. 265, no. 1-2, pp. 227–264, 2001. [Online]. Available: http://dx.doi.org/10.1016/S0304-3975(01)00164-5

[6] J. K. Slaney and T. Walsh, "Backbones in optimization and approximation," in *Proc. IJCAI*, 2001, pp. 254–259.

[7] W. Zhang, "Phase transitions and backbones of 3-SAT and maximum 3-SAT," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 2239. Springer, 2001, pp. 153–167.

[8] S. Climer and W. Zhang, "Searching for backbones and fat: A limit-crossing approach with applications," in *Proc. AAAI*. AAAI Press / The MIT Press, 2002, pp. 707–712.

[9] W. Zhang, A. Rangan, and M. Looks, "Backbone guided local search for maximum satisfiability," in *Proc. IJCAI*. Morgan Kaufmann, 2003, pp. 1179–1186.

[10] W. Zhang, "Configuration landscape analysis and backbone guided local search: Part I: satisfiability and maximum satisfiability," *Artificial Intelligence*, vol. 158, no. 1, pp. 1–26, 2004. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2004.04.001

[11] ——, "Phase transitions and backbones of the asymmetric traveling salesman problem," *Journal of Artificial Intelligence Research*, vol. 21, pp. 471–497, 2004. [Online]. Available: http://dx.doi.org/10.1613/jair.1389

[12] P. Kilby, J. K. Slaney, S. Thiébaux, and T. Walsh, "Backbones and backdoors in satisfiability," in *Proc. AAAI*. AAAI Press / The MIT Press, 2005, pp. 1368–1373.

[13] P. Kilby, J. K. Slaney, and T. Walsh, "The backbone of the travelling salesperson," in *Proc. IJCAI*, 2005, pp. 175–180.

[14] W. Zhang and M. Looks, "A novel local search algorithm for the traveling salesman problem that exploits backbones," in *Proc. IJCAI*. Professional Book Center, 2005, pp. 343–350.

[15] M. E. Menai, "A two-phase backbone-based search heuristic for partial MAX-SAT - an initial investigation," in *Proc. IEA/AIE*, ser. Lecture Notes in Computer Science, vol. 3533. Springer, 2005, pp. 681–684.

[16] E. I. Hsu, C. J. Muise, J. C. Beck, and S. A. McIlraith, "Probabilistically estimating backbones and variable bias: Experimental overview," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 5202. Springer, 2008, pp. 613–617.

[17] O. Dubois and G. Dequen, "A backbone-search heuristic for efficient solving of hard 3-SAT formulae," in *Proc. IJCAI*, 2001, pp. 248–253.

[18] J. Marques-Silva, M. Janota, and I. Lynce, "On computing backbones of propositional theories," in *Proc. ECAI*, ser. Frontiers in Artificial Intelligence and Applications, vol. 215. IOS Press, 2010, pp. 15–20.

[19] M. Codish, Y. Fekete, and A. Metodi, "Backbones for equality," in *Proc. HVC*, ser. Lecture Notes in Computer Science, vol. 8244. Springer, 2013, pp. 1–14.

[20] M. Janota, I. Lynce, and J. Marques-Silva, "Algorithms for computing backbones of propositional formulae," *AI Communications*, vol. 28, no. 2, pp. 161–177, 2015. [Online]. Available: http://dx.doi.org/10.3233/AIC-140640

[21] A. J. Parkes, "Clustering at the phase transition," in *Proc. AAAI/IAAI*. AAAI Press / The MIT Press, 1997, pp. 340–345.

[22] P. Jonsson and A. A. Krokhin, "Recognizing frozen variables in constraint satisfaction problems," *Theoretical Computer Science*, vol. 329, no. 1-3, pp. 93–113, 2004. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2004.08.006

[23] L. Bordeaux, M. Cadoli, and T. Mancini, "Exploiting fixable, removable, and implied values in constraint satisfaction problems," in *Proc. LPAR 2004*, ser. Lecture Notes in Computer Science, vol. 3452. Springer, 2005, pp. 270–284.

[24] ——, "A unifying framework for structural properties of CSPs: Definitions, complexity, tractability," *Journal of Artificial Intelligence Research*, vol. 32, pp. 607–629, 2008.

[25] ——, "Generalizing consistency and other constraint properties to quantified constraints," *ACM Transactions on Computational Logic*, vol. 10, no. 3, pp. 17:1–17:25, 2009. [Online]. Available: http://doi.acm.org/10.1145/1507244.1507247

[26] C. S. Zhu, G. Weissenbacher, and S. Malik, "Post-silicon fault localisation using maximum satisfiability and backbones," in *Proc. FMCAD*. FMCAD Inc., 2011, pp. 63–66.

[27] C. S. Zhu, G. Weissenbacher, D. Sethi, and S. Malik, "SAT-based techniques for determining backbones for post-silicon fault localisation," in *Proc. HLDVT*. IEEE Computer Society, 2011, pp. 84–91.

[28] M. Janota, "SAT solving in interactive configuration," Ph.D. dissertation, University College Dublin, November 2010.

[29] C. S. Zhu, G. Weissenbacher, and S. Malik, "Silicon fault diagnosis using sequence interpolation with backbones," in *Proc. ICCAD*. IEEE, 2014, pp. 348–355.

[30] A. Belov, M. Janota, I. Lynce, and J. Marques-Silva, "Algorithms for computing minimal equivalent subformulas," *Artificial Intelligence*, vol. 216, pp. 309–326, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2014.07.011

[31] E. Hebrard, B. Hnich, and T. Walsh, "Super solutions in constraint programming," in *Proc. CPAIOR*, ser. Lecture Notes in Computer Science, vol. 3011. Springer, 2004, pp. 157–172.

[32] A. Hyttinen, P. Hoyer, F. Eberhardt, and M. Järvisalo, "Discovering cyclic causal models with latent variables: A general SAT-based procedure," in *Proc. UAI*. AUAI Press, 2013, pp. 301–310.

[33] J. P. Wallner, G. Weissenbacher, and S. Woltran, "Advanced SAT techniques for abstract argumentation," in *Proc. CLIMA*, ser. Lecture Notes in Computer Science, vol. 8143. Springer, 2013, pp. 138–154.

[34] G. Gottlob, "On minimal constraint networks," *Artif. Intell.*, vol. 191-192, pp. 42–60, 2012.

[35] N. Amaneddine, J. Condotta, and M. Sioutis, "Efficient approach to solve the minimal labeling problem of temporal and spatial qualitative constraints," in *Proc. IJCAI*, 2013, pp. 696–702.

[36] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*. IOS Press, 2009.

[37] D. Kroening and O. Strichman, *Decision Procedures - An Algorithmic Point of View, Second Edition*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-662-50497-0

[38] T. Walsh, "SAT v CSP," in *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, ser. Lecture Notes in Computer Science, R. Dechter, Ed., vol. 1894. Springer, 2000, pp. 441–456. [Online]. Available: https://doi.org/10.1007/3-540-45349-0_32

[39] M. Fitting, *First-Order Logic and Automated Theorem Proving, Second Edition*, ser. Graduate Texts in Computer Science. Springer, 1996. [Online]. Available: https://doi.org/10.1007/978-1-4612-2360-3

[40] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Proc. TACAS*, ser. Lecture Notes in Computer Science, vol. 4963. Springer, 2008, pp. 337–340.

[41] I. Dillig, T. Dillig, K. L. McMillan, and A. Aiken, "Minimum satisfying assignments for SMT," in *Proc. CAV*, ser. Lecture Notes in Computer Science, vol. 7358. Springer, 2012, pp. 394–409.

[42] I. Dillig, T. Dillig, and A. Aiken, "Automated error diagnosis using abductive inference," in *Proc. PLDI*. ACM, 2012, pp. 181–192.