A Preference-Based Approach to Backbone Computation with Application to Argumentation

Alessandro Previti
HIIT, Dept. Computer Science, University of Helsinki
alessandro.previti@helsinki.fi

Matti Järvisalo
HIIT, Dept. Computer Science, University of Helsinki

ABSTRACT

The backbone of a constraint satisfaction problem consists of those variables that take the same value in all solutions. Algorithms for determining the backbone of propositional formulas, i.e., Boolean satisfiability (SAT) instances, find various real-world applications. From the knowledge representation and reasoning (KRR) perspective, one interesting connection is that of backbones and the so-called ideal semantics in abstract argumentation. In this paper, we propose a new backbone algorithm which makes use of a "SAT with preferences" solver, i.e., a SAT solver which is guaranteed to output a most preferred satisfying assignment w.r.t. a given preference over literals of the SAT instance at hand. We also show empirically that the proposed approach is specifically effective in computing the ideal semantics of argumentation frameworks, noticeably outperforming an other state-of-the-art backbone solver as well as the winning approach of the recent ICCMA 2017 argumentation solver competition in the ideal semantics track.

ACM Reference Format:
Alessandro Previti and Matti Järvisalo. 2018. A Preference-Based Approach to Backbone Computation with Application to Argumentation. In *Proceedings of ACM SAC Conference (SAC'18).* ACM, New York, NY, USA, 7 pages. https://doi.org/https://doi.org/10.1145/3167132.3167230

1 INTRODUCTION

The so-called *backbone* is an established concept [33, 34, 44] in constraint satisfaction problems (CSPs), and finds a wide range of applications [5, 22, 23, 48, 54–56]. The backbone of a CSP consists of variables that take the same value in all solutions, together with their respective values. Backbones have been studied in the contexts of various combinatorial problems [8, 10, 21, 28, 29, 32–34, 45, 46, 49–53]. In terms of CSPs, backbones have been studied for general finite-domain constraint satisfaction, as well as in the context of Boolean satisfiability (SAT) [9, 11, 24, 30, 38]—as focused on also in this work. The problem of determining the backbone of a given propositional formula is a notably hard problem and surpasses the complexity of deciding satisfiability.

The applicability of backbones arises from the fact that they enable expressing various kinds of interesting information. The existence of backbone variables precludes the existence of super-solutions [20]. Perhaps most interestingly, in various application domains of SAT solvers, it is possible to develop SAT encodings in which the inclusion of specific variables in the backbone of the SAT-encoded instance provides an indicator for a property of interest—the exact property depending on the problem domain at hand. Concretely, backbones have been used to represent faults in fault localization in integrated circuits [54, 56]; in interactive product configuration, an identified backbone variable provides users important information on unavailable configurations [23]; and in causal structure discovery, backbone variables have been used to represent equivalence classes of causal structures [22], to mention a few examples. In the topical area of KR research of abstract argumentation—and importantly in terms of this paper—the so-called ideal semantics [13] can be realized through determining the backbone of a propositional formula encoding so-called admissible sets [14, 48].

Motivated by the widening range of applications, developing efficient algorithms for backbone computation, i.e., for determining the backbone of a given propositional formula, is important. Indeed, several algorithmic approaches to backbone computation have been recently developed [24], based on iterative applications of SAT solvers as natural practical NP oracles.

In this work, we propose a new backbone algorithm for propositional formulas. While earlier state-of-the-art algorithms apply standard conflict driven clause learning (CDCL) SAT solvers iteratively, the algorithm proposed here is based on the use of a CDCL SAT solver extended to provide a best solution with respect to a given preference (ordering) on the literals in an input SAT instance. While this "SAT with preferences" approach [16, 17, 42] has been employed for other beyond NP problems [3, 18, 40–42], to the best of our understanding the use of SAT with preferences has not been previously proposed in the context of backbone computation.

In terms of practical performance, we present results from an empirical evaluation of the proposed approach. Specifically, in the context of abstract argumentation, the approach noticeably outperforms both a state-of-the-art generic backbone algorithm as well as Pyglaf [1], the winner of the 2017 *International Competition of Computational Models of Argumentation* (ICCMA 2017) argumentation solver competition, on the problem of computing the ideal semantics over argumentation frameworks via determining the backbone of a SAT encoding for the admissible semantics.

The rest of this paper is organized as follows. We start with necessary background on Boolean satisfiability (Section 2) and backbones with a short overview of backbone algorithms in the context of SAT (Section 3). The main contribution of the paper, the preference-based backbone algorithm, is described in Section 4. As we will

SAC'18, April 9-13, 2018, Pau,France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/https://doi.org/10.1145/3167132.3167230

empirically show, the algorithm is noticeably efficient for computing the ideal semantics on argumentation framework. We give background on this application domain in Section 5. An overview of results from an empirical evaluation of the proposed backbone algorithm is presented in Section 6. Finally, before conclusions, we give an overview of related work (Section 7).

2 BOOLEAN SATISFIABILITY

Propositional formulas are built from Boolean variables by repeated application of logical connectives such as $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence). Any propositional formula can be represented in *conjunctive normal form* (CNF) using a standard linear-size encoding [47].[1] For a Boolean variable x, there are two literals, the positive literal x and the negative literal $\neg x$. A *clause* is a disjunction of literals, and a CNF formula is a conjunction of clauses. A clause can be represented as a set of literals, and a CNF formula as a set of clauses. Given a formula F, $\mathrm{Var}(F)$ denotes the set of variables of F and $\mathrm{Lit}(F)$ denotes the set of literals of F. Moreover, for a literal l, $\mathrm{Var}(l)$ gives its variable.

A truth assignment is a function τ from Boolean variables to $\{0, 1\}$. A truth assignment is complete if $\tau(x) \in \{0, 1\}$ for all $x \in \mathrm{Var}(F)$. An assignment τ satisfies a literal ($\tau(l) = 1$) if $l = x$ is a positive literal and $\tau(x) = 1$, or if $l = \neg x$ is a negative literal and $\tau(x) = 0$. An assignment τ satisfies a clause C ($\tau(C) = 1$) if $\tau(l) = 1$ for some literal l in C. A CNF formula F is satisfiable if there is an assignment that satisfies all clauses in F, and unsatisfiable otherwise. An assignment that satisfies all the clauses in F is referred to as a satisfying assignment, or a *model*, of F. The NP-complete Boolean satisfiability (SAT) problem asks whether a given CNF formula F is satisfiable.

A truth assignment τ can also be represented as the set $\{l \mid \tau(l) = 1\}$ of literals it satisfies. We will make use of this representation when detailing the backbone algorithm proposed in this work, and will write, e.g., $l \in \tau$ (resp., $l \notin \tau$) to denote that $\tau(l) = 1$ (resp., $\tau(l) = 0$).

Implementations of decision procedures for SAT, so-called SAT solvers, can in practice not only determine satisfiability of CNF formulas, but also produce a satisfying truth assignment for satisfiable formulas. Various hard computational problems have been successfully approached by first encoding the problem at hand as a propositional formula, and then calling an off-the-shelf SAT solver to find a satisfying truth assignment to the formula, representing a solution of the original problem at hand. The most efficient SAT solvers are based on the complete conflict-driven clause learning (CDCL) search algorithm [15, 31, 35]. Central to CDCL is the ability to derive lemmas (in terms of new CNF clauses) based on non-solutions detected during search, which makes the search performed by CDCL SAT solvers differ from standard depth-first backtracking search. In many cases, the state-of-the-art CDCL SAT solvers can solve SAT instances consisting of millions of clauses and variables [26].

[1] The standard linear-size "Tseitin" CNF encoding introduces a fresh Boolean variable x_ϕ for each subformula ϕ, and represents the logical equivalence $x_\phi \Leftrightarrow \phi$ with clauses.

3 BACKBONES AND COMPUTATION

If a Boolean variable x takes the same value in all satisfying truth assignments of a given CNF formula F, x is called a *backbone variable* of F; the value x is assigned to in all satisfying assignments is called the polarity of x. If $x = 1$ ($x = 0$) in all satisfying assignments, then x ($\neg x$) is a *backbone literal* of F. The backbone of F consists of the backbone literals of F, or equivalently, of its backbone variables together with their respective truth values.

The following simple observation is central to backbone computation. In particular, given a variable x such that $\tau_1(x) = 0$ and $\tau_2(x) = 1$, where τ_1 and τ_2 are two models of a CNF formula F, neither of the literals x and $\neg x$ are backbone literals of F.

Proposition 1. *For any satisfiable CNF formula F, literal $l \in \mathrm{Lit}(F)$, and model τ of F, if $\tau(l) = 0$ (resp., $\tau(l) = 1$), then l (resp., $\neg l$) is not a backbone literal of F.*

Algorithms for determining the backbone of a given propositional formula are generally based on iterative applications of a SAT solver.

The most intuitive and straightforward way for determining the backbone works by making a linear number of calls (in the number of variables in F) to a SAT solver and follows Proposition 1: if exactly one of $F \wedge x$ and $F \wedge \neg x$ is satisfiable, then x is in the backbone of F.

Several techniques for improving the practical efficiency of the "straightforward" approach to backbone computation have been previously proposed [24, 56]. Many of the resulting more refined backbone algorithms rely on the connection between *implicants* (i.e., partial satisfying assignments to the formula F at hand) and the backbone of F. Since a backbone literal is contained in every model, the intersection of all implicants of F corresponds to the backbone of F. While the number of prime implicants can be exponential in the worst case ([24]), various techniques have been proposed with the aim of avoiding the worst-case scenario of enumerating all the prime implicants. Common to the resulting algorithms is the idea of first computing a model of F, and then proceeding by iteratively flipping variable assignments in the model and checking the satisfiability of F under the modified assignments. Techniques refining this general scheme include *chunking*, i.e., testing a subset of literals instead of all literals at once; employment of *unsatisfiable cores* obtained from unsatisfiable SAT solver calls for inferring inclusion and exclusion of literals from the backbone; and *backbone filtering* via the concept of so-called *rotatable literals*.

4 A PREFERENCE-BASED APPROACH TO BACKBONE COMPUTATION

In this section we present a new approach for backbone computation. Compared to previous approaches for backbone computation, a key difference in our approach is the following. Given a CNF formula F as input, previously proposed approaches (such as the ones proposed in [24, 56]) determine the inclusion and exclusion of literals in the backbone of F by iteratively calling a (standard) CDCL SAT solver. In contrast, we propose to employ a SAT solver *extended with preferences* [36, 42] (a "pref-SAT" solver) instead of a standard SAT solver. A pref-SAT solver allows for finding a best satisfying assignment (model) with respect to a preference ordering over

the literals of F [36, 43]. We will show how such preferences can be harnessed for backbone computation. While pref-SAT solvers have been recently employed for various other beyond-NP problems [3, 17, 42]), to the best of our understanding this is the first time their use for backbone computation is proposed.

In the context of this discussion, a preference specifies a preferred value for a specific variable ([7]) over other individual value assignments to variables. Given a CNF formula and such a preference relation $\succ$, a pref-SAT solver is guaranteed to return the most preferred model of F in terms of $\succ$ (or report *unsatisfiable* in case F is unsatisfiable). A preference relation $\succ$ imposes a partial order among the literals in $\text{Lit}(F)$. More formally, a preference relation imposes a partial ordering over $\text{Lit}(F)$ satisfying the following properties.

(1) *Irreflexivity*: $l \not\succ l$.
(2) *Transitivity*: if $l_1 \succ l_2$ and $l_2 \succ l_3$, then $l_1 \succ l_3$.

Intuitively, $\succ$ expresses the relative importance of the preferences. If we have $l_1 \succ l_2$, a literal l_1 is preferred to literal l_2. Furthermore, if $l_1 \succ l_2 \succ l_3$, then $l_1 \succ l_2$ is more preferred than $l_2 \succ l_3$, and $l_2 \succ l_3$ less preferred than $l_1 \succ l_2$.

A preference relation hence also imposes a preference on the models of a CNF formula. Given two models τ and τ' (viewed as sets of literals from now on), we say that τ is preferred to τ', denoted by $\tau \succ \tau'$, if and only if

(1) τ satisfies at least one preference that is not satisfied by τ', and
(2) the preferences satisfied by τ' and not by τ are less preferred to those satisfied by τ and not by τ'.

The preferred value of a variable x is 1 (resp., 0) if the literal x (resp., $\neg x$) is preferred to $\neg x$ (resp., x).

A pref-SAT solver can be implemented on top of a (standard) modern CDCL SAT solver by controlling the order in which decisions on variables are made during search. Modern SAT solvers use a heuristic in order to select the next variable to be assigned. In order to take into account a given preference $x_1 \succ x_2$, a pref-SAT solver is forced to decide on the variable x_1 before x_2. When the variable x_1 is decided on, the value 1 is assigned to it. This is possible unless the value $x_1 = 1$ is implied by the current partial assignment.

With the necessary background on pref-SAT in place, we are ready to present our preference-based approach to backbone computation. Algorithm 1 outlines the approach in pseudocode. The intuitive idea is to discard a *maximal* number of non-backbone literals at each iteration. Recall that a backbone literal is a literal that is contained in every model. If we find two models τ_1 and τ_2 such that $x \in \tau_1$ and $\neg x \in \tau_2$, then neither x nor $\neg x$ is a backbone literal (Proposition 1). In the context of our algorithm, we use this observation together with preferences in order to discard non-backbone literals from consideration. More specifically, the algorithm maintains a set of backbone literal candidates $\mathcal{B}$. At any stage during search, literal l is in $\mathcal{B}$ if we have not seen a model with $\neg l$.

The search begins (Algorithm 1, line 2) by computing an arbitrary model τ of the input formula F; i.e., at this stage, no preferences are imposed, and the pref-SAT solver acts like a standard SAT solver. The set of candidate backbone literals $\mathcal{B}$ is initialized to τ (line 3).

Algorithm 1: BB-pref: Backbone computation using pref-SAT

```
1  Function BB-PREF(F)
2      τ ← pref-SAT(F)
3      B ← τ
4      for l ∈ B do
5          setPreference(¬l)
6      while true do
7          τ ← pref-SAT(F)
8          C ← B \ τ
9          if C = ∅ then
10             return B
11         for l ∈ C do
12             removePreference(¬l)
13             B ← B \ {l}
```

Then, for each $l \in \mathcal{B}$ the algorithm sets the preference $\neg l \succ l'$ for each $l' \in \text{Lit}(F) \setminus \mathcal{B}'$, where $\mathcal{B}' = \{\neg l \mid l \in \mathcal{B}\}$, via the *setPreference* function (line 5). The idea here is to force a maximal set of literals in $\mathcal{B}$ to be flipped. For each literal l in $\mathcal{B}$ that we are able to flip (in terms of obtaining a model under the modified $\mathcal{B}$), we know by Proposition 1 that l and $\neg l$ are not backbone literals. During the main loop, pref-SAT is called to obtain the most preferred model τ w.r.t. the modified $\mathcal{B}$ (line 7). On line 8 information of the flipped literals are extracted and stored in C. If C is not empty, we know for each literal $l \in C$ that neither l nor $\neg l$ is a backbone literal. So for each $l \in C$ we remove the preferences on l via the *removePreference* function (line 12), and further, we remove l from the set of backbone literal candidates $\mathcal{B}$ (line 13). Otherwise, if C is empty, it is no more possible to flip any literals in $\mathcal{B}$. This means that all the literals in $\mathcal{B}$ are backbone literals and the set $\mathcal{B}$ is returned (line 10).

Finally, we will discuss more implementation-level details on how pref-SAT is instantiated for the approach. On line 5 we impose for each literal $l \in \mathcal{B}$ to prefer a model containing $\neg l$. More precisely, let $\mathcal{B}' = \{\neg l \mid l \in \mathcal{B}\}$ be the set of preferred literals. We impose the preferences

$$\neg l \succ l' \quad \forall l \in \mathcal{B} \text{ and } \forall l' \in \text{Lit}(F) \setminus \mathcal{B}'.$$

This means in practice that the solver has to assign all the variables referring to the literals in $\mathcal{B}$ before selecting any other variable. All the literals within $\mathcal{B}$ and $\text{Lit}(F) \setminus \mathcal{B}'$ can be selected according to the solver heuristic. This is very important in order to not affect performance, since imposing a fixed order could have a noticeable negative impact on the efficiency of the SAT solver in the worst case [25].

Modern SAT solvers use a heap H for selecting the next variable to decide on. In our implementation, we split the variables in two heaps, H_1 and H_2, such that all the variables in H_1 are selected before the variables in H_2. The heap H_1 contains all the variables whose literals are in $\mathcal{B}$. The second contains all the remaining variables. When a variable in H_1 is selected, the corresponding preferred value is assigned to it. For all the other variables the choice of their values is left to the solver. The variables in $\mathcal{B}$ are decided on first. When no preference is specified, pref-SAT acts as a standard SAT solver. The key modifications to a SAT solver come

in the form of implementing the two functions *setPreference* and *removePreference*.

- *setPreference*(l) sets the preference for l to be satisfied and in addition adds $l \succ l'$ for each $l' \in \text{Lit}(F) \setminus \mathcal{B}'$. On the implementation level this means that the variable referring to the literal l is put in H_1 and the algorithm makes sure to assign the preferred value when the variable is selected.
- *removePreference*(l) removes the preference for l to be satisfied and in addition removes $l \succ l'$ for each $l' \in \text{Lit}(F) \setminus \mathcal{B}'$ and adds $l' \succ l$ for each $l' \in \mathcal{B}'$. On the implementation level this means that the variable referring to the literal l is removed from H_1 and put in H_2. Moreover, the preferred value associated to the variable is removed.

When no preference is expressed on the two literals of a variable, the variable is assumed to be in H_2.

Example 1. *Consider the formula* $F = x_1 \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$. *Suppose that we obtain as the first model* $\tau_1 = \{x_1, x_2, \neg x_3, x_4\}$. *Then, for each literal* $l \in \mathcal{B} = \{x_1, x_2, \neg x_3, x_4\}$, *we add the variable* $\text{Var}(l)$ *to the first heap* H_1. *At this stage the heap* H_2 *is empty. When the second model* $\tau_2 = \{x_1, x_2, x_3, x_4\}$ *is found, we have* $C = \mathcal{B} \setminus \tau_2 = \{\neg x_3\}$. *So, we remove* $\text{Var}(\neg x_3)$ *from* H_1 *and add* $\text{Var}(\neg x_3)$ *to the second heap* H_2. *We also remove* $\neg x_3$ *from* $\mathcal{B}$. *At the next call to pref-SAT, the model* $\tau_3 = \{x_1, x_2, \neg x_3, x_4\}$ *is returned. At this point* $C = \mathcal{B} \setminus \tau_3 = \emptyset$ *and* $\mathcal{B} = \{x_1, x_2, x_4\}$ *is the set of backbone literals of the formula.*

5 IDEAL SEMANTICS AS BACKBONE COMPUTATION

We recall concepts related to argumentation frameworks [12] and their semantics [4, 13].

Definition 1. *An* argumentation framework (AF) *is a pair* $F = (A, R)$, *where* A *is a finite set of arguments and* $R \subseteq A \times A$ *is the attack relation. The pair* $(a, b) \in R$ *means that* a attacks b.

Definition 2. *An argument* $a \in A$ *is* defended *(in* F*) by a set* $S \subseteq A$ *if, for each* $b \in A$ *such that* $(b, a) \in R$, *there exists* $c \in S$ *such that* $(c, b) \in R$.

Example 2. *Let* $F = (A, R)$ *be an AF with* $A = \{a, b, c, d, e\}$ *and* $R = \{(a, b), (b, c), (c, d), (d, c), (d, e), (e, e)\}$. *The corresponding graph representation is shown in Figure 1.*

Semantics for argumentation frameworks are defined through a function σ which assigns to each AF $F = (A, R)$ a set $\sigma(F) \subseteq 2^A$ of *conflict-free* extensions.

Definition 3. *Let* $F = (A, R)$ *be an AF. A set* $S \subseteq A$ *is* conflict-free (in F), *if there are no* $a, b \in S$ *such that* $(a, b) \in R$. *We denote the collection of conflict-free sets of* F *by* $cf(F)$.

We consider for σ the functions *adm*, *pref*, and *ideal* which stand for admissible, preferred and ideal extensions, respectively.

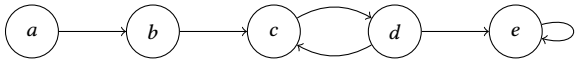

Figure 1: Argumentation framework in Example 2.

Definition 4. *Let* $F = (A, R)$ *be an AF and* $S \in cf(F)$. *Now* S *is an admissible extension (*$S \in adm(F)$*) if and only if*

$$S \subseteq \{x \in A \mid x \text{ is defended by } S\}.$$

Preferred extensions are the subset-maximal admissible extensions.

Definition 5. *Let* $F = (A, R)$ *be an AF and* $S \in adm(F)$. *Now* S *is a preferred extension (*$S \in pref$*) if and only if there is no* $S' \supsetneq S$ *such that* $S' \in adm$.

Finally, as the main AF semantics of interest in this work, the ideal semantics, which is a so-called unique-status semantics, i.e., there is a unique ideal extension for any AF. Informally, the ideal extension is the maximal-admissible set with respect to subset-inclusion that is composed only of skeptically accepted arguments under the preferred semantics.

Definition 6. *Let* $F = (A, R)$ *be an AF and, furthermore,* $S \subseteq \bigcap pref(F)$, *i.e., S is a subset of the set of* skeptically accepted *arguments under the preferred semantics. Now* S *is the ideal extension of* F (*$S \in ideal(F)$*) if and only if there is no* $S' \in adm(F)$ *with* $S' \supsetneq S$ *such that* $S' \subseteq \bigcap pref(F)$.

As explained in [14, 48], the ideal extension of a given AF $F = (A, R)$ can be determined via computing the backbone of a propositional encoding of admissible sets, and afterwards applying straightforward postprocessing to the backbone. Specifically, the main computational task (in terms of computational complexity) is to determine the set of *credulously accepted arguments* of F with respect to admissible sets, i.e., the set of arguments $\bigcup adm(F)$. This is achieved by first computing the backbone B of the standard propositional encoding

$$\bigwedge_{(a,b)\in R} (\neg a \vee \neg b) \wedge \bigwedge_{(b,c)\in R} \Big(\neg c \vee \bigvee_{(a,b)\in R} a\Big)$$

of $adm(F)$. It then holds that $\bigcup adm(F) = A \setminus \{a \mid \neg a \in B\}$. As detailed in [48], the ideal extension is then easy to determine from $\bigcup adm(F)$ via a fast polynomial-time algorithm.[2]

6 EXPERIMENTS

In this section, we provide results from an empirical evaluation on the performance of the proposed backbone algorithm using a "SAT with preferences" solver. Our evaluation is focused on the task of computing the ideal semantics for a given argumentation framework.

We implemented our approach (Algorithm 1) on top of the widely-used MiniSAT [15] CDCL SAT solver, version 2.2.0. We will refer to the implementation as PrefBones.

We will compare the performance of PrefBones to that of MiniBones [24], a state-of-the-art solver for backbone computation, as well as Pyglaf [1], the winner of the ICCMA 2017 competition track on computing the ideal extension (see http://www.dbai.tuwien.ac.at/iccma17/). Following the suggestion of the authors of MiniBones, we used the following parameter values in the experiments: `-e -i`

[2] In short, starting from $S = A \setminus \bigcup adm(F)$, first add to S arguments $x \in \bigcup adm(F)$ such that all arguments adjacent to x are in $A \setminus \bigcup adm(F)$. Then, considering the AF $F' = (S, R_S)$, where R_S is R restricted to S, iteratively remove from S argument which are not defended by S in F'. After at most $|S|$ iterations, this yields the ideal extension of F [14, 48].

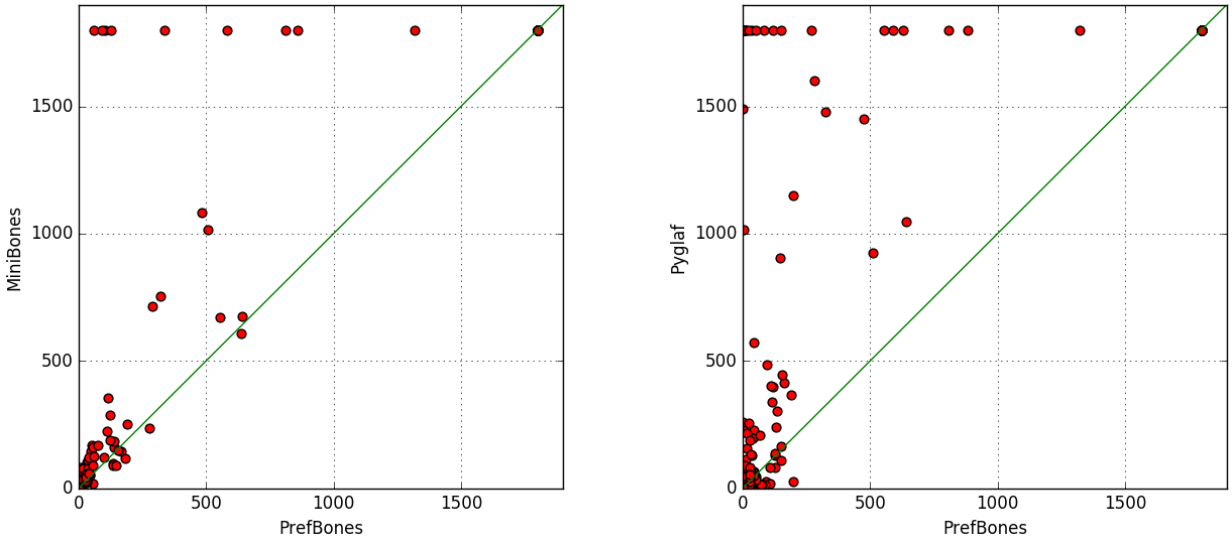

Figure 2: Comparison of the approaches on the ICCMA'17 benchmarks. Left: PrefBones vs MiniBones. Right: PrefBones vs Pyglaf.

`-c 100`. It should be noted that MiniBones also uses MiniSAT 2.2.0 as the underlying SAT solver.

For the evaluation, we considered two types of argumentation frameworks as benchmarks. Firstly, we used the whole ICCMA 2017 competition benchmark set D, which was used in the competition for benchmarking solvers on the ideal semantics. Secondly, we generated random AFs using `afgen` generator from [37] under the Erdös-Renýi random graph model, generating a total of 50 AFs using 630 and 1% as the number of arguments and the edge probability, respectively.[3] For both types of AFs, we used the `cnfgen` tool from [37] to generate the CNF instances for PrefBones and MiniBones.

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 were set to 1800 seconds and 4 GB, respectively. When comparing the relative performance of PrefBones and MiniBones, we do not include the CNF generation times in the results, as these do not differ between the solvers. However, when comparing PrefBones and Pyglaf, we include the CNF generation times and the postprocessing times to the per-instance running times of PrefBones for a fair comparison.

The results from the evaluation are shown in Figure 2 and Figure 3 for the ICCMA 2017 and the randomly generated benchmarks, respectively. We observe that PrefBones scales clearly better on the ICCMA 2017 instances than both MiniBones and Pyglaf. There were no instances on which PrefBones would timeout and one of the competing approaches could compute the backbone. Compared to MiniBones, PrefBones is always at least on par with its competitor, and there are several instances on which PrefBones can determine the backbone clearly faster, and PrefBones also solves considerably more instances. Compared to Pyglaf, the solvers are essentially on par on the easier benchmarks. However, PrefBones again solves a considerable number of instances noticeably faster than Pyglaf, as well as considerably more instances. In fact, this would have made PrefBones a clear winner of the ideal track of ICCMA 2017, as Pyglaf was the winner of that track.

Finally, turning to the results on the randomly generated benchmark set (Figure 3), we observe that PrefBones clearly and consistently outperforms both MiniBones and Pyglaf, whereas as the performance of MiniBones and Pyglaf on these instances is essentially on par with each other.

All in all, we conclude that the algorithm proposed in this work and implemented in PrefBones offers currently a very competitive approach to computing the ideal semantics of argumentation frameworks.

7 RELATED WORK

In terms of previously proposed algorithms for computing the backbone of a given propositional formulas, essentially all approaches (including those proposed in [24, 56]) rely on using standard complete conflict-driven clause learning (CDCL) SAT solvers [15, 31, 35]. While the algorithm proposed in this work bears resemblance to the idea of enumerating implicants [24], our approach differs crucially in terms of our use of a SAT with preferences solver—instead of a standard SAT solver—and by using preferences for avoiding the enumeration of all implicants before termination. Due to harnessing SAT with preferences, our approach has the potential of converging faster than the approach enumerating implicants to the final intersection representing the backbone.

In terms of the approaches using standard SAT solvers, in [24] various ideas have been presented on how to force the SAT solver to return models with flipped literals. One of the best-performing approaches tries to flip all the literals at once, using assumptions,

[3] This gives rise to AFs in which each argument is part of approximately 7 attacks. The number of arguments was chosen via experimentation so that meaningful benchmark instances were obtained considering the per-instance time limit enforced on the solvers.

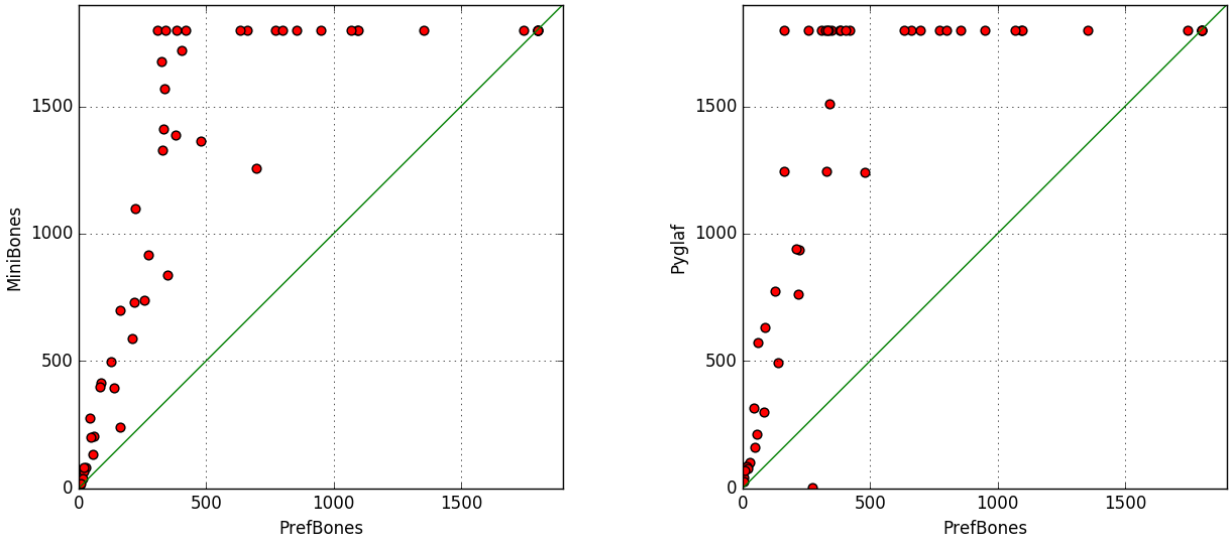

Figure 3: Comparison of the approaches on randomly generated AFs. Left: PrefBones vs MiniBones. Right: PrefBones vs Pyglaf.

and then analyzes the returned *unsatisfiable core*. Another approach, firstly introduced in [56], tries to flip at least one literal for each SAT call by adding a new clause of the form $\bigvee_{l \in \mathcal{B}} \neg l$, where $\mathcal{B}$ is the set of backbone candidates we want to try to flip.

The backbone algorithm proposed in this work relies heavily on the "SAT with preference" approach. The SAT with preferences approach was proposed in [16, 17, 42], with applications to maximal/minimal model computation and maximum satisfiability, automated planning [18], model enumeration [40], qualitative preferences [41], and relaxation search [3]. However, to the best of our knowledge the approach has not been previously considered as a basis for backbone computation.

Backbone computation is a central approach to implementing the ideal semantics in abstract argumentation [4, 12, 13], i.e., for computing the ideal set of a given argumentation framework [14, 48]. The very recently organized 2nd International Competition on Computational Models of Argumentation (ICCMA 2017) included a competition track on computing the ideal extension, where a majority of the competing approaches (10 in total) were based on backbone computation (or, equivalently, on computing the cautious consequences of a constraint declaration of the ideal semantics), including the winner of the track, Pyglaf.

Beyond SAT [9, 11, 24, 30, 38] and SAT-based applications [5, 22, 23, 48, 54–56], backbones play a role in a wide range of combinatorial problems and in CSPs in general [8, 10, 21, 28, 29, 32–34, 45, 46, 49–53]. Variants and generalizations have also been proposed. In the context of CSPs, a notion of backbones has been proposed under the name of frozen/fixable variables [6, 10, 27], defined as variables that take the same value in all solutions. A more general notion is that of generalized backbones [39], which extends backbones to arbitrary variable domains and thereby to, e.g., the satisfiability modulo theories approach (SMT). Backbones also have connections to minimal constraint networks [19] and minimal labelings in qualitative constraints networks [2].

8 CONCLUSIONS

Backbone algorithms for propositional SAT instances have applications in a widening range of real-world problems. We proposed a new approach to backbone computation based on iterative application of a "SAT with preferences" solver; in contrast, other current state-of-the-art algorithms typically rely on standard SAT solvers. In practice, the proposed approach noticeably outperforms both a state-of-the-art generic backbone algorithm as well as Pyglaf, the winner of the ICCMA 2017 argumentation solver competition, on the problem of computing the ideal semantics over argumentation frameworks via determining the backbone of a SAT encoding for the ideal semantics.

ACKNOWLEDGMENTS

The authors thank Mikolas Janota for advice on MiniBones, and Andreas Niskanen for discussion and advice on the AFGen and CNFGen generators. The work has been financially supported by Academy of Finland (grants 251170 COIN, 276412, 284591, and 312662) and the Research Funds of the University of Helsinki.

REFERENCES

[1] Mario Alviano. 2017. The pyglaf argumentation reasoner. In *ICCMA 2017 Solver Descriptions*. http://www.dbai.tuwien.ac.at/iccma17/files/pyglaf.pdf.
[2] Nouhad Amaneddine, Jean-François Condotta, and Michael Sioutis. 2013. Efficient Approach to Solve the Minimal Labeling Problem of Temporal and Spatial Qualitative Constraints. In *Proc. IJCAI*. IJCAI/AAAI, 696–702.
[3] Fahiem Bacchus, Jessica Davies, Maria Tsimpoukelli, and George Katsirelos. 2014. Relaxation Search: A Simple Way of Managing Optional Clauses. In *Proc. AAAI*. AAAI Press, 835–841.
[4] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. 2011. An introduction to argumentation semantics. *Knowledge Engingeering Review* 26, 4 (2011), 365–410.
[5] Anton Belov, Mikolás Janota, Inês Lynce, and João Marques-Silva. 2014. Algorithms for computing minimal equivalent subformulas. *Artificial Intelligence* 216 (2014), 309–326.
[6] Lucas Bordeaux, Marco Cadoli, and Toni Mancini. 2005. Exploiting Fixable, Removable, and Implied Values in Constraint Satisfaction Problems. In *Proc. LPAR 2004 (Lecture Notes in Computer Science)*, Vol. 3452. Springer, 270–284.

[7] Thierry Castell, Claudette Cayrol, Michel Cayrol, and Daniel Le Berre. 1996. Using the Davis and Putnam Procedure for an Efficient Computation of Preferred Models. In *Proc. ECAI*. John Wiley and Sons, 350–354.

[8] Sharlee Climer and Weixiong Zhang. 2002. Searching for Backbones and Fat: A Limit-Crossing Approach with Applications. In *Proc. AAAI*. AAAI Press, 707–712.

[9] Michael Codish, Yoav Fekete, and Amit Metodi. 2013. Backbones for Equality. In *Proc. HVC (Lecture Notes in Computer Science)*, Vol. 8244. Springer, 1–14.

[10] Joseph C. Culberson and Ian P. Gent. 2001. Frozen development in graph coloring. *Theoretical Computer Science* 265, 1-2 (2001), 227–264.

[11] Olivier Dubois and Gilles Dequen. 2001. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proc. IJCAI*. Morgan Kaufmann, 248–253.

[12] Phan Minh Dung. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence* 77, 2 (1995), 321–358.

[13] Phan Minh Dung, Paolo Mancarella, and Francesca Toni. 2007. Computing ideal sceptical argumentation. *Artificial Intelligence* 171, 10-15 (2007), 642–674. https://doi.org/10.1016/j.artint.2007.05.003

[14] Paul E. Dunne, Wolfgang Dvorák, and Stefan Woltran. 2013. Parametric properties of ideal semantics. *Artificial Intelligence* 202 (2013), 1–28.

[15] Niklas Eén and Niklas Sörensson. 2004. An Extensible SAT-solver. In *Proc. SAT 2003 (Lecture Notes in Computer Science)*, Vol. 2919. Springer, 502–518.

[16] Enrico Giunchiglia and Marco Maratea. 2006. optsat: A Tool for Solving SAT Related Optimization Problems. In *Proc. JELIA (Lecture Notes in Computer Science)*, Vol. 4160. Springer, 485–489.

[17] Enrico Giunchiglia and Marco Maratea. 2006. Solving Optimization Problems with DLL. In *Proc. ECAI (Frontiers in Artificial Intelligence and Applications)*, Vol. 141. IOS Press, 377–381.

[18] Enrico Giunchiglia and Marco Maratea. 2007. Planning as Satisfiability with Preferences. In *Proc. AAAI*. AAAI Press, 987–992.

[19] Georg Gottlob. 2012. On minimal constraint networks. *Artificial Intelligence* 191-192 (2012), 42–60.

[20] Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. 2004. Super Solutions in Constraint Programming. In *Proc. CPAIOR (Lecture Notes in Computer Science)*, Vol. 3011. Springer, 157–172.

[21] Eric I. Hsu, Christian J. Muise, J. Christopher Beck, and Sheila A. McIlraith. 2008. Probabilistically Estimating Backbones and Variable Bias: Experimental Overview. In *Proc. CP (Lecture Notes in Computer Science)*, Vol. 5202. Springer, 613–617.

[22] Antti Hyttinen, Patrik Hoyer, Frederick Eberhardt, and Matti Järvisalo. 2013. Discovering Cyclic Causal Models with Latent Variables: A General SAT-Based Procedure. In *Proc. UAI*. AUAI Press, 301–310.

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

[24] Mikolás Janota, Inês Lynce, and Joao Marques-Silva. 2015. Algorithms for computing backbones of propositional formulae. *AI Communications* 28, 2 (2015), 161–177.

[25] Matti Järvisalo and Tommi Junttila. 2009. Limitations of Restricted Branching in Clause Learning. *Constraints* 14, 3 (2009), 325–356.

[26] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. 2012. The International SAT Solver Competitions. *AI Magazine* 33, 1 (2012), 89–92.

[27] Peter Jonsson and Andrei A. Krokhin. 2004. Recognizing frozen variables in constraint satisfaction problems. *Theoretical Computer Science* 329, 1-3 (2004), 93–113.

[28] Philip Kilby, John K. Slaney, Sylvie Thiébaux, and Toby Walsh. 2005. Backbones and Backdoors in Satisfiability. In *Proc. AAAI*. AAAI Press, 1368–1373.

[29] Philip Kilby, John K. Slaney, and Toby Walsh. 2005. The Backbone of the Travelling Salesperson. In *Proc. IJCAI*. Professional Book Center, 175–180.

[30] João Marques-Silva, Mikolás Janota, and Inês Lynce. 2010. On Computing Backbones of Propositional Theories. In *Proc. ECAI (Frontiers in Artificial Intelligence and Applications)*, Vol. 215. IOS Press, 15–20.

[31] João P. Marques-Silva and Karem A. Sakallah. 1999. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Comput.* 48, 5 (1999), 506–521.

[32] Mohamed El-bachir Menai. 2005. A Two-Phase Backbone-Based Search Heuristic for Partial MAX-SAT - An Initial Investigation. In *Proc. IEA/AIE (Lecture Notes in Computer Science)*, Vol. 3533. Springer, 681–684.

[33] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansk. 1999. Determining computational complexity from characteristic 'phase transitions'. *Nature* 400 (1999), 133–137.

[34] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. 1999. 2+p-SAT: Relation of typical-case complexity to the nature of the phase transition. *Random Structures and Algorithms* 15, 3-4 (1999), 414–435.

[35] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proc. DAC*. ACM, 530–535.

[36] Davy Van Nieuwenborgh, Stijn Heymans, and Dirk Vermeir. 2004. On Programs with Linearly Ordered Multiple Preferences. In *Proc. ICLP (Lecture Notes in Computer Science)*, Vol. 3132. Springer, 180–194.

[37] Andreas Niskanen, Johannes Peter Wallner, and Matti Järvisalo. 2016. Pakota: A System for Enforcement in Abstract Argumentation. In *Proc. JELIA (Lecture Notes in Computer Science)*, Vol. 10021. Springer, 385–400.

[38] Andrew J. Parkes. 1997. Clustering at the Phase Transition. In *Proc. AAAI/IAAI*. AAAI Press, 340–345.

[39] Alessandro Previti, Alexey Ignatiev, Matti Järvisalo, and Joao Marques-Silva. 2017. On Computing Generalized Backbones. In *Proc. ICTAI*. IEEE Computer Society.

[40] Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea. 2008. Computing All Optimal Solutions in Satisfiability Problems with Preferences. In *Proc. CP (Lecture Notes in Computer Science)*, Peter J. Stuckey (Ed.), Vol. 5202. Springer, 603–607.

[41] Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea. 2008. A new Approach for Solving Satisfiability Problems with Qualitative Preferences. In *Proc. ECAI (Frontiers in Artificial Intelligence and Applications)*, Vol. 178. IOS Press, 510–514.

[42] Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea. 2010. Solving satisfiability problems with preferences. *Constraints* 15, 4 (2010), 485–515.

[43] Chiaki Sakama and Katsumi Inoue. 2000. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence* 123, 1-2 (2000), 185–222.

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

[45] Josh Singer, Ian P. Gent, and Alan Smaill. 2000. Backbone Fragility and the Local Search Cost Peak. *Journal of Artificial Intelligence Research* 12 (2000), 235–270.

[46] John K. Slaney and Toby Walsh. 2001. Backbones in Optimization and Approximation. In *Proc. IJCAI*. Morgan Kaufmann, 254–259.

[47] Grigori S. Tseitin. 1983. On the Complexity of Derivation in Propositional Calculus. In *Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970*, J. Siekmann and G. Wrightson (Eds.). Springer, 466–483.

[48] Johannes Peter Wallner, Georg Weissenbacher, and Stefan Woltran. 2013. Advanced SAT Techniques for Abstract Argumentation. In *Proc. CLIMA (Lecture Notes in Computer Science)*, Vol. 8143. Springer, 138–154.

[49] Weixiong Zhang. 2001. Phase Transitions and Backbones of 3-SAT and Maximum 3-SAT. In *Proc. CP (Lecture Notes in Computer Science)*, Vol. 2239. Springer, 153–167.

[50] Weixiong Zhang. 2004. Configuration landscape analysis and backbone guided local search: Part I: Satisfiability and maximum satisfiability. *Artificial Intelligence* 158, 1 (2004), 1–26.

[51] Weixiong Zhang. 2004. Phase Transitions and Backbones of the Asymmetric Traveling Salesman Problem. *Journal of Artificial Intelligence Research* 21 (2004), 471–497.

[52] Weixiong Zhang and Moshe Looks. 2005. A Novel Local Search Algorithm for the Traveling Salesman Problem that Exploits Backbones. In *Proc. IJCAI*. Professional Book Center, 343–350.

[53] Weixiong Zhang, Ananda Rangan, and Moshe Looks. 2003. Backbone Guided Local Search for Maximum Satisfiability. In *Proc. IJCAI*. Morgan Kaufmann, 1179–1186.

[54] Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. 2011. Post-silicon fault localisation using maximum satisfiability and backbones. In *Proc. FMCAD*. FMCAD Inc., 63–66.

[55] Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. 2014. Silicon fault diagnosis using sequence interpolation with backbones. In *Proc. ICCAD*. IEEE, 348–355.

[56] Charlie Shucheng Zhu, Georg Weissenbacher, Divjyot Sethi, and Sharad Malik. 2011. SAT-based techniques for determining backbones for post-silicon fault localisation. In *Proc. HLDVT*. IEEE Computer Society, 84–91.