Argpref:
A SAT-with-Preferences Approach to Ideal Semantics

Alessandro Previti and Matti Järvisalo

HIIT, Department of Computer Science, University of Helsinki, Finland
apreviti.research@gmail.com,matti.jarvisalo@helsinki.fi

Abstract

We provide a short overview of the Argpref solver submitted to the ICCMA 2019 competition. Argpref focuses on computation of the ideal semantics. The solver implements a recently proposed SAT-with-preferences approach to computing the backbone of a propositional encoding of admissible sets in order to construct the ideal extension of a given argumentation framework from the backbone.

1 Introduction

We provide a short overview of the Argpref solver submitted to the ICCMA 2019 competition. Argpref focuses on computation of the ideal semantics. The solver implements a recently-proposed SAT-with-preferences [5, 7] based approach [6] to computing the backbone of a propositional encoding of admissible sets of a given argumentation framework, and applies polynomial-time postprocessing [3, 9] to construct the ideal extension from the backbone.

2 Backbone Computation by SAT-with-Preferences

We shortly overview the SAT-with-preferences approach to backbone computation; for more details, see [6].

If a Boolean variable x takes the same value in all satisfying truth assignments of a given conjunctive normal form (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. 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.

Algorithm 1 outlines in pseudocode the BB-pref approach to computing the backbone of a given CNF formula. A pref-SAT solver allows for finding a best satisfying assignment (model) with respect to a preference ordering over the literals of F [8, 5]. 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. 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

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}
```

a standard SAT solver. The set of candidate backbone literals $\mathcal{B}$ is initialized to τ (line 3). 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 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 $\mathcal{C}$. If $\mathcal{C}$ is not empty, we know for each literal $l \in \mathcal{C}$ that neither l nor $\neg l$ is a backbone literal. So for each $l \in \mathcal{C}$ 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 $\mathcal{C}$ 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).

3 Postprocessing to Obtain the Ideal Extension

As explained in [3, 9], the ideal extension of a given argumentation framework (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 [2]

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

of $adm(F)$, i.e., the collection of admissible sets of F. It then holds that $\bigcup adm(F) = A \setminus \{a \mid \neg a \in B\}$.

As detailed in [9], the ideal extension is then easy to determine from $\bigcup adm(F)$ via a fast polynomial-time algorithm. 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 [3, 9].

4 Implementation

The SAT-with-preferences approach is implemented on top of the MiniSAT 2.2.0 SAT solver [4]. The postprocessing and input-output interface is also integrated into the code of MiniSAT. The approach was shown in [6] to perform well on the ICCMA 2017 benchmarks, outperforming the first-place Pyglaf solver [1].

5 Availability

The solver can be found under the repository **elsandp/argpre** at

`https://hub.docker.com/r/elsandp/argpref`.

The tasks supported by Argpref are: **DC-ID, SE-ID**.

References

[1] Mario Alviano. The pyglaf argumentation reasoner. In *ICCMA 2017 Solver Descriptions*, 2017. `http://www.dbai.tuwien.ac.at/iccma17/files/pyglaf.pdf`.

[2] Philippe Besnard and Sylvie Doutre. Checking the acceptability of a set of arguments. In *Proc. NMR*, pages 59–64, 2004.

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

[4] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[5] Davy Van Nieuwenborgh, Stijn Heymans, and Dirk Vermeir. On programs with linearly ordered multiple preferences. In *Proc. ICLP*, volume 3132 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2004.

[6] Alessandro Previti and Matti Järvisalo. A preference-based approach to backbone computation with application to argumentation. In *Proc. SAC*, pages 896–902. ACM, 2018.

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

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

[9] Johannes Peter Wallner, Georg Weissenbacher, and Stefan Woltran. Advanced SAT techniques for abstract argumentation. In *Proc. CLIMA*, volume 8143 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2013.