Solving Set Optimization Problems by Cardinality Optimization with an Application to Argumentation

Wolfgang Faber[1] and **Mauro Vallati**[1] and **Federico Cerutti**[2] and **Massimiliano Giacomin**[3]

Abstract.
Optimization—minimization or maximization—in the lattice of subsets is a frequent operation in Artificial Intelligence tasks. Examples are subset-minimal model-based diagnosis, nonmonotonic reasoning by means of circumscription, or preferred extensions in abstract argumentation. Finding the optimum among many admissible solutions is often harder than finding admissible solutions with respect to both computational complexity and methodology. This paper addresses the former issue by means of an effective method for finding subset-optimal solutions. It is based on the relationship between cardinality-optimal and subset-optimal solutions, and the fact that many logic-based declarative programming systems provide constructs for finding cardinality-optimal solutions, for example maximum satisfiability (MaxSAT) or weak constraints in Answer Set Programming (ASP). Clearly each cardinality-optimal solution is also a subset-optimal one, and if the language also allows for the addition of particular restricting constructs (both MaxSAT and ASP do) then all subset-optimal solutions can be found by an iterative computation of cardinality-optimal solutions. As a showcase, the computation of preferred extensions of abstract argumentation frameworks using the proposed method is studied.

1 INTRODUCTION

In Artificial Intelligence, the task of set optimization, in the sense of finding a set that is minimal or maximal with respect to set inclusion, frequently occurs. There are famous examples such as Circumscription [20] or Model-based Diagnosis that involve set minimization. Computing preferred extensions of abstract argumentation frameworks [12] is an example that involves set maximization.

Often, set optimization is an element that creates difficulties in implementation and representation. For example, McCarthy had to resort to Second-order logic for defining Circumscription of First-order theories [20]. Also for computing preferred extensions, relatively sophisticated techniques are required, for instance QBFs rather than propositional formulas [2].

There is another notion of set optimization, finding a set that has minimal or maximal cardinality, which has more readily available system support nowadays. The most prominent examples of languages and systems that support cardinality optimization are MaxSAT, Constraint Programming, and Answer Set Programming (ASP).

[1] School of Computing and Engineering,University of Huddersfield, Huddersfield, UK. email: n.surname@hud.ac.uk

[2] Cardiff University, School of Computer Science & Informatics, Cardiff, UK. email: ceruttif@cardiff.ac.uk

[3] Università degli Studi di Brescia, Brescia, Italy. email: massimiliano.giacominb@unibs.it

In this paper, we show how set optima can be computed by a general algorithm that employs cardinality optimizing subroutines, provided that the underlying languages allow for expressing simple constraints. Algorithm MCSes in Figure 2 of [19], which computes Minimal Correction Sets[4] of a propositional formula, bears a few similarities to our algorithms. However, it is also different in several respects, most notably it is formulated for solving one particular problem only, assumes propositional formulas as the representation formalism, and does not employ a cardinality optimization oracle explicitly. To the best of our knowledge a general method applicable in a variety of representation formalisms and for unspecific set optimization problems settings has not been proposed in the literature before. We develop two instantiations of the general method, for MaxSAT and for ASP, and show that they are suitable.

There are two more recent software tools that also support computing set optimization problems when the underlying language is ASP: **asprin** [9] and **D-FLAT^2** [8]. The scope of the tool **asprin** is actually reasoning with preferences, but as a special case one can express preferences such that only set optimal solutions remain. The required preferences come with the predefined library of **asprin**. The underlying algorithms of **asprin** are very different from those presented in this paper. **D-FLAT^2** builds on dynamic programming and exploits tree decomposition in order to solve set optimization problems and is therefore also very different from the method that we will present in this paper. In the realm of ASP, the tool **metasp** [17] can be seen as a predecessor of **asprin**, which does not seem to be maintained any longer. It relies on reification of rules and exploits a programming pattern known as saturation for set optimization, which is also very different from the method described in this paper.

We then turn our attention to a showcase application, computing preferred extensions of abstract argumentation frameworks. Dung's theory of abstract argumentation [12] is a unifying framework able to encompass a large variety of specific formalisms in the areas of nonmonotonic reasoning, logic programming and computational argumentation. It is based on the notion of argumentation framework (AF), consisting of a set of *arguments* and a binary *attack* relation between them. Arguments can thus be represented by nodes of a directed graph, and attacks by arcs. The nature of arguments is left unspecified: it can be anything from logical statements to informal natural language text. For instance, [24] shows how argumentation can be efficiently used for supporting critical thinking and intelligence analysis in military-sensitive contexts.

Different *argumentation semantics* declare the criteria to deter-

[4] It is worth noticing that algorithms exploiting minimal correction sets have been proposed for computing argumentation semantics extensions, in particular for semi-stable, ideal, and eager semantics [25], but not for preferred semantics which is our main test-case in this paper.

mine which arguments emerge as "justified" among conflicting ones, by identifying a number of *extensions*, i.e. sets of arguments that can "survive the conflict together". In [12] four "traditional" semantics were introduced, namely *complete*, *grounded*, *stable*, and *preferred* semantics. For a complete overview of subsequently proposed alternative semantics, the interested reader is referred to [3].

The main computational problems in abstract argumentation include *decision*—e.g. determine if an argument is in all the extensions prescribed by a semantics—and *construction* problems, and turn out to be computationally intractable for most argumentation semantics [13]. In this paper we focus on the *extension enumeration* problem, i.e. constructing *all* extensions for a given AF: its solution provides complete information about the justification status of arguments and allows for solving the other problems as well.

Our general method allowed for the definition and implementation of two novel algorithms for enumerating preferred extensions: **prefMaxSAT** (based on a MaxSAT solver) and **prefASP** (based on an ASP solver). Both are evaluated using benchmarks from the International Competition on Computational Models of Argumentation (ICCMA2015). We report on a variety of experiments: the first focuses on **prefASP** and starts with comparing the use of different solver configurations for **prefASP**, followed by a comparison of **prefASP** to **asprin** and **D-FLAT^2**, and eventually comparing **prefASP** to the dedicated argumentation solver **ASPARTIX-V**. Eventually we compare **prefASP** and **prefMaxSAT** to the ICCMA2015 competition winner **Cegartix**. The experiments show that despite their conceptual simplicity, our software tools are competitive with the best available ones.

2 ABSTRACT METHODOLOGY

The proposed methodology applies to a variety of knowledge representation formalisms, we therefore consider an abstract setting. We define a knowledge base $\mathcal{K}$ to be associated with a set $\sigma(\mathcal{K})$ of solutions, and we assume that each $s \in \sigma(\mathcal{K})$ is a set. We also use a set restriction operator $_{\downarrow O}$ such that $s_{\downarrow O} = s \cap O$, the idea being that $s_{\downarrow O}$ identifies the solution elements that are relevant for optimization. We also assume a composition operator $\mathcal{K}_1 \circ \mathcal{K}_2$ to be present that allows to compose two knowledge bases $\mathcal{K}_1$ and $\mathcal{K}_2$. We next define a few optimization criteria for solutions of knowledge bases.

Definition 1 *Let $\mathcal{K}$ be a knowledge base and R be a set (of elements occuring in solutions of $\mathcal{K}$). Define:*

$$\begin{aligned}
S_R^{max}(\mathcal{K}) &= \{s \mid s \in \sigma(\mathcal{K}), \nexists s' \in \sigma(\mathcal{K}) : s'_{\downarrow R} \supset s_{\downarrow R}\}\\
S_R^{min}(\mathcal{K}) &= \{s \mid s \in \sigma(\mathcal{K}), \nexists s' \in \sigma(\mathcal{K}) : s'_{\downarrow R} \subset s_{\downarrow R}\}\\
C_R^{max}(\mathcal{K}) &= \{s \mid s \in \sigma(\mathcal{K}), \nexists s' \in \sigma(\mathcal{K}) : |s'_{\downarrow R}| > |s_{\downarrow R}|\}\\
C_R^{min}(\mathcal{K}) &= \{s \mid s \in \sigma(\mathcal{K}), \nexists s' \in \sigma(\mathcal{K}) : |s'_{\downarrow R}| < |s_{\downarrow R}|\}
\end{aligned}$$

While $S_R^{min}(\mathcal{K})$ and $S_R^{max}(\mathcal{K})$ occur in diverse applications of knowledge representation and reasoning, it often happens that the computational complexity of these tasks increases (under standard assumptions) compared to $S_R^{min}(\mathcal{K})$ and $S_R^{max}(\mathcal{K})$. For example, deciding whether $\sigma(\mathcal{K}) = \emptyset$ is co-NP-complete (in the size of $\mathcal{K}$) if $\mathcal{K}$ is represented using a propositional formula and $\sigma(\mathcal{K})$ is the set of satisfying assignments, where each assignment is represented as the set of true variables. In this setting, computing $S_R^{max}(\mathcal{K})$ is then Σ_2^P-hard, as showing the optimality of a solution may require exponentially many co-NP checks, while $C_R^{max}(\mathcal{K})$ is in Δ_2^P, requiring at most a polynomial number of co-NP checks. Note that this does not necessarily have practical consequences, because at the moment all known algorithms to solve these problems require at least exponential time.

There are also representational repercussions. Still assuming $\mathcal{K}$ to be a propositional formula, one cannot find a propositional formula of polynomial size that encodes any of $S_R^{min}(\mathcal{K})$, $S_R^{max}(\mathcal{K})$, $C_R^{max}(\mathcal{K})$, and $C_R^{min}(\mathcal{K})$ (if $NP \neq \Sigma_2^P$, which is currently unknown, but often conjectured). If $\mathcal{K}$ has been modelled using ASP, it is easy to encode $C_R^{max}(\mathcal{K})$ and $C_R^{min}(\mathcal{K})$ because of the availability of weak constraints (or optimization constructs). In fact, one can use ASP also for encoding $S_R^{min}(\mathcal{K})$ and $S_R^{max}(\mathcal{K})$, because ASP can express all problems in Σ_2^P. We will discuss this further in Section 3.

In this paper, we relate $S_R^{max}(\mathcal{K})$ to $C_R^{max}(\mathcal{K})$ (and $S_R^{min}(\mathcal{K})$ to $C_R^{min}(\mathcal{K})$). We first observe that each cardinality optimal solution is also subset optimal.

Observation 1 *For any knowledge base $\mathcal{K}$ and set R, $C_R^{max}(\mathcal{K}) \subseteq S_R^{max}(\mathcal{K})$ and $C_R^{min}(\mathcal{K}) \subseteq S_R^{min}(\mathcal{K})$.*

This observation holds because if any $s \in C_R^{max}(\mathcal{K})$ were not in $S_R^{max}(\mathcal{K})$, then there would be some $s' \in \sigma(\mathcal{K})$ such that $s'_{\downarrow R} \supset s_{\downarrow R}$ and clearly $|s'_{\downarrow R}| > |s_{\downarrow R}|$ then holds (and symmetrically for minimization).

This implies that when the task is to compute one subset optimal solution, one can instead safely compute one cardinality optimal solution. When, however, the computational task involves an enumeration of all subset optimal solutions, one is faced with incompleteness, as not all subset optima are cardinality optimal.

Example 1 *Let $\mathcal{K}_1$ be such that $\sigma(\mathcal{K}_1) = \{\{a,b\},\{b\},\{c\}\}$ and let $R_1 = \{a,b,c\}$. Then $S_{R_1}^{max}(\mathcal{K}_1) = \{\{a,b\},\{c\}\}$ while $C_{R_1}^{max}(\mathcal{K}_1) = \{\{a,b\}\}$.*

This can be overcome by an iterative approach, in which first cardinality optimal solutions are computed. In the next stage, the knowledge base is extended in a way that it no longer admits the solutions already found or any subsets (for maximization) or supersets (for minimization) thereof.

Definition 2 *Given a knowledge base $\mathcal{K}$, a set R, and a set $S \subseteq \sigma(\mathcal{K})$, let $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$ denote a knowledge base such that*

$$\sigma(\mathcal{K} \circ \mathcal{N}^{\subseteq}(\mathcal{K}, R, S)) = \sigma(\mathcal{K}) \setminus \{s' \mid s'_{\downarrow R} \subseteq s_{\downarrow R} \wedge s \in S\}.$$

Symmetrically, let $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$ denote a knowledge base such that

$$\sigma(\mathcal{K} \circ \mathcal{N}^{\supseteq}(\mathcal{K}, R, S)) = \sigma(\mathcal{K}) \setminus \{s' \mid s'_{\downarrow R} \supseteq s_{\downarrow R} \wedge s \in S\}.$$

It depends on the formalism used for the knowledge base, whether $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$ and $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$ can be created, and in particular whether they can be represented in a concise way. It also depends on the formalism whether there is a uniform way of encoding $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$ and $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$, or whether one has to rely on a representation that depends on the structure of $\mathcal{K}$.

The iterative approach is formalized for subset maximality in Algorithm 1 and for subset minimality in Algorithm 2. Note that practical algorithms will usually not collect all solutions in the output because of space considerations, but rather output them immediately as they are computed.

Theorem 1 *For a knowledge base $\mathcal{K}$ and set R, Algorithm 1 computes $S_R^{max}(\mathcal{K})$, and Algorithm 2 computes $S_R^{min}(\mathcal{K})$.*

Algorithm 1 Enumerating $S_R^{max}(\mathcal{K})$ by means of $C_R^{max}(\mathcal{K})$

```
 1: Input: $\mathcal{K}, R$
 2: Output: $S_R^{max}(\mathcal{K})$
 3: $\mathcal{K}_i := \mathcal{K}$
 4: $S := \emptyset$
 5: $S_i := C_R^{max}(\mathcal{K}_i)$
 6: while $S_i \mathrel{!=} \emptyset$ do
 7:     $S := S \cup S_i$
 8:     $\mathcal{K}_i := \mathcal{K}_i \circ \mathcal{N}^{\subseteq}(\mathcal{K}_i, R, S_i)$
 9:     $S_i := C_R^{max}(\mathcal{K}_i)$
10: end while
11: return $S$
```

Algorithm 2 Enumerating $S_R^{min}(\mathcal{K})$ by means of $C_R^{min}(\mathcal{K})$

```
 1: Input: $\mathcal{K}, R$
 2: Output: $S_R^{min}(\mathcal{K})$
 3: $\mathcal{K}_i := \mathcal{K}$
 4: $S := \emptyset$
 5: $S_i := C_R^{min}(\mathcal{K}_i)$
 6: while $S_i \mathrel{!=} \emptyset$ do
 7:     $S := S \cup S_i$
 8:     $\mathcal{K}_i := \mathcal{K}_i \circ \mathcal{N}^{\supseteq}(\mathcal{K}_i, R, S_i)$
 9:     $S_i := C_R^{min}(\mathcal{K}_i)$
10: end while
11: return $S$
```

Proof 1 (Sketch) *We show that Algorithm 1 computes $S_R^{max}(\mathcal{K})$. Symmetric arguments prove that Algorithm 2 computes $S_R^{min}(\mathcal{K})$.*

We first observe that each assignment of variable S_i contains only elements of $S_R^{max}(\mathcal{K})$. When variable S_i is first initialized in line 5 of Algorithm 1, Observation 1 guarantees the claim. For each later assignment, by construction only $s \in \sigma(\mathcal{K})$ are assigned, and any such s is such that $\nexists s' \in \sigma(\mathcal{K}) : s'_{\downarrow R} \supset s_{\downarrow R}$ (otherwise $|s'_{\downarrow R}| > |s_{\downarrow R}|$ would hold). It is also clear that the algorithm terminates (if $\sigma(\mathcal{K})$ is finite).

Now observe that each $s \in S_R^{max}(\mathcal{K})$ is assigned once to S_i in Algorithm 1. Indeed, the first assignment contains all elements in $S_R^{max}(\mathcal{K})$ that are of maximum cardinality, the next iteration contains all elements in $S_R^{max}(\mathcal{K})$ of the next-highest cardinality, and so forth down to the elements of $S_R^{max}(\mathcal{K})$ of least cardinality in the last assignment. In this way, all elements of $S_R^{max}(\mathcal{K})$ will be contained in S when Algorithm 1 terminates.

It should be pointed out that Algorithms 1 and 2 also work when instead of $C_R^{max}(\mathcal{K}_i)$ (or $C_R^{min}(\mathcal{K}_i)$) any non-empty subset thereof is assigned to S_i in lines 5 and 9. In particular, for the instantiation of the framework using MaxSAT that will be discussed in Section 3.1, many MaxSAT solvers calculate only one solution, rather than all.

Let us note that the number of subcalls to $C_R^{max}(\mathcal{K}_i)$ (resp., $C_R^{min}(\mathcal{K}_i)$) is at most $|S_R^{max}(\mathcal{K})|$ (resp., $|S_R^{min}(\mathcal{K})|$). The cardinality of these sets can be exponential in $\mathcal{K}$ in the worst case. That also means that in the worst case an exponential number of knowledge bases $\mathcal{N}^{\supseteq}(\mathcal{K}_i, R, S_i)$ (or $\mathcal{N}^{\subseteq}(\mathcal{K}_i, R, S_i)$) are composed to $\mathcal{K}$, which could lead to a use of exponential space. Note however, that this only occurs if there is an exponential number of solutions to be generated by the algorithm. This only occurs if $s_{1\downarrow R} \nsubseteq s_{2\downarrow R}$ and $s_{2\downarrow R} \nsubseteq s_{1\downarrow R}$ holds for almost all solutions s_1 and s_2 of $\sigma(\mathcal{K})$.

Note that there is also a contrast to more traditional algorithms that invoke a co-NP oracle call for each $s \in \sigma(\mathcal{K})$, especially if they run a test on each subset of a found solution. In that setting, the number of subcalls that take exponential time will usully be much greater than $|S_R^{min}(\mathcal{K})|$ (resp., $|S_R^{min}(\mathcal{K})|$). We view this feature of our algorithm as one of the main advantages over more traditional methods.

Algorithm 1 bears some similarities to Algorithm MCSes in Figure 2 of [19]. Algorithm MCSes computes Minimal Correction Sets of a propositional formula. Apart from the fact that is solves a much more specific problem and assumes a specific knowledge representation formalism. In fact, it iteratively increases the cardinality of the (relevant portion of the) solution to be computed and enforces the cardinality by means of formulas thus imitating the behaviour of a MaxSAT. The clauses that Algorithm MCSes adds follow the same idea of $\mathcal{N}^{\subseteq}(\mathcal{K}_i, R, S_i)$, and correspond to $\mathcal{N}^{\subseteq}_{sat}(\mathcal{K}_i, R, S_i)$ that will be defined in Section 3.1.

3 CONCRETIZATIONS USING MaxSAT AND ASP

We now show how to instantiate the abstract method described in Section 2 using MaxSAT and ASP.

3.1 SAT and MaxSAT

In Boolean satisfiability (SAT) [7], one asks for satisfying assignments of a propositional formula φ. We will assume that the propositional variables are taken from the infinite set $\mathcal{L}$ and use the standard connectives $\neg, \wedge, \vee, \supset$. SAT solvers usually assume that φ is represented in conjunctive normal form, as a set of clauses.

In this setting, the terminology of Section 2 is instantiated as follows: $\mathcal{K}$ is a propositional formula φ over $\mathcal{L}$, or a set of clauses over $\mathcal{L}$. The solutions $\sigma(\mathcal{K})$ are the satisfying assignments of $\mathcal{K}$, represented as sets of propositional variables that are true in the respective assignment. The operation $\mathcal{K}_1 \circ \mathcal{K}_2$ is either $\mathcal{K}_1 \wedge \mathcal{K}_2$ if $\mathcal{K}_1, \mathcal{K}_2$ are formulae, or $\mathcal{K}_1 \cup \mathcal{K}_2$ if $\mathcal{K}_1, \mathcal{K}_2$ are sets of clauses. In the sequel, we will assume the clause notation.

In order to encode $C_R^{max}(\mathcal{K})$ and $C_R^{min}(\mathcal{K})$, one can make use of weighted MaxSAT [18]. In weighted MaxSAT, one can assign a numerical weight to some of the clauses (known as soft clauses), and these clauses do not necessarily have to be satisfied. The solutions of a MaxSAT problem are determined among all assignments that satisfy the non-weighted clauses (hard clauses), and are those that maximize the sum of weights of the soft clauses.

Definition 3 *Given a set of clauses $\mathcal{K}$ and $R \subset \mathcal{L}$, we define the following MaxSAT problem*

$$C_{R,sat}^{max}(\mathcal{K}) \equiv \mathcal{K} \cup \{1 : \{r\} \mid r \in R\}$$

consisting of hard clauses $\mathcal{K}$ and soft clauses $\{r\}$ (unit clauses) for each $r \in R$, all with the same positive weight 1 (any other weight could be used, as long it is the same for all soft clauses).

In a similar way,

$$C_{R,sat}^{min}(\mathcal{K}) \equiv \mathcal{K} \cup \{1 : \{\neg r\} \mid r \in R\}$$

consists of hard clauses $\mathcal{K}$ and soft clauses $\{\neg r\}$ (unit clauses) for each $r \in R$, again all with the same weight.

It is clear that $C_R^{max}(\mathcal{K})$ corresponds to the solutions of $C_{R,sat}^{max}(\mathcal{K})$ and $C_R^{min}(\mathcal{K})$ corresponds to the solutions of $C_{R,sat}^{min}(\mathcal{K})$.

For obtaining $S_R^{max}(\mathcal{K})$ and $S_R^{min}(\mathcal{K})$, one would have to resort to Quantified Boolean Formulae (QBFs), which we do not discuss in detail in this paper.

Example 2 *An instantiation of Example 1 would be the* $\mathcal{K}_1^{sat} = \{\{\neg a, \neg c\}, \{\neg b, \neg c\}, \{b, c\}\}$*, with* $\sigma(\mathcal{K}_1^{sat}) = \{\{a, b\}, \{b\}, \{c\}\}$*. Then, with* $R_1 = \{a, b, c\}$*,* $C^{max}_{R_1,sat}(\mathcal{K}_1^{sat}) = \{\{\neg a, \neg c\}, \{\neg b, \neg c\}, \{b, c\}, 1 : \{a\}, 1 : \{b\}, 1 : \{c\}\}$*, and the only optimal solution of* $C^{max}_{R_1,sat}(\mathcal{K}_1^{sat})$ *is* $\{a, b\}$*, which is equal to* $C^{max}_{R_1}(\mathcal{K}_1^{sat})$.

Finally, we consider how to encode $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$ and $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$ in SAT. For $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$, the idea is to require for each solution in S that at least an element of R not in that solution should be true. This inhibits the solution and any subset (restricted to R) of it. For $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$, we require for each solution in S that at least one variable in R should be false. In this way the solution itself and any superset is inhibited.

Definition 4 *Given a set of clauses* $\mathcal{K}$*,* $R \subset \mathcal{L}$*, and* $S \subseteq \sigma(\mathcal{K})$*, let*

$$\mathcal{N}^{\subseteq}_{sat}(\mathcal{K}, R, S) \equiv \bigwedge_{s \in S} \bigvee_{v \in R \setminus s} v$$

and in a similar way

$$\mathcal{N}^{\supseteq}_{sat}(\mathcal{K}, R, S) \equiv \bigwedge_{s \in S} \bigvee_{v \in R \cap s} \neg v \ .$$

Note that these formulas are in clause form. It is clear that $\mathcal{N}^{\subseteq}_{sat}(\mathcal{K}, R, S)$ and $\mathcal{N}^{\supseteq}_{sat}(\mathcal{K}, R, S)$ restrict the solutions in the way required by $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$ and $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$.

Example 3 *Continuing Example 3, if* $S_1 = \{\{a, b\}\}$*, then* $\mathcal{N}^{\subseteq}_{sat}(\mathcal{K}_1^{sat}, R_1, S_1) = \{\{c\}\}$*, therefore the only optimal solution of* $C^{max}_{R_1,sat}(\mathcal{K}_1^{sat}) \cup \mathcal{N}^{\subseteq}_{sat}(\mathcal{K}_1^{sat}, R_1, S_1)$ *is* $\{c\}$.

3.2 Answer Set Programming

In Answer Set Programming (ASP) one asks for the answer sets (often also called stable models) of a logic program. The full language specification can be found at `https://www.mat.unical.it/aspcomp2013/ASPStandardization`, below we provide a brief overview of the concepts relevant to this work.

The basic constructs in ASP logic programs are of the form

$$\mathtt{h_1} \mid \ldots \mid \mathtt{h_k} \ :-\ \mathtt{b_1},\ \ldots,\ \mathtt{b_m},\ \mathtt{not}\ \mathtt{b_{m+1}},\ \ldots,\ \mathtt{not}\ \mathtt{b_n}.$$

where $0 \leq \mathtt{k}$, $0 \leq \mathtt{m} \leq \mathtt{n}$ and the $\mathtt{h_i}$ and $\mathtt{b_j}$ are function-free first-order atoms. When $\mathtt{k} > 0$, it is called a rule, otherwise a constraint. If $\mathtt{k} = 1$ and $\mathtt{m} = \mathtt{n} = 0$, the rule is called a fact. The part left of the construct $:-$ is called head, the part right of it is called body. Sets of rules and constraints are called programs.

Programs with variables are thought of as shorthand for their ground (variable-free) versions with respect to the Herbrand universe of the program. Answer sets are defined on ground programs: they are Herbrand models of the program, which satisfy an additional stability condition.

The language of ASP consists of quite a lot more constructs. One relevant for this paper is the weak constraint, which takes the form

$$:\sim\ \mathtt{b_1},\ \ldots,\ \mathtt{b_m},\ \mathtt{not}\ \mathtt{b_{m+1}},\ \ldots,\ \mathtt{not}\ \mathtt{b_n}.$$

An interpretation that satisfies all literals to the right of $:\sim$ will incur a (uniform) penalty. Answer sets of programs with weak constraints are then those answer sets of the weak-constraint-free portion that minimize the penalties incurred by weak constraints.

For ASP, the terminology of Section 2 is instantiated as follows: $\mathcal{K}$ in this setting is a weak-constraint-free program, and $\sigma(\mathcal{K})$ is the set of its answer sets. The operation $\mathcal{K}_1 \circ \mathcal{K}_2$ simply is the set union of the two programs $\mathcal{K}_1$ and $\mathcal{K}_2$.

In ASP, it is possible to encode $C^{max}_R(\mathcal{K})$ and $C^{min}_R(\mathcal{K})$ by means of weak constraints, similar to the MaxSAT approach described earlier.

Definition 5 *Given a set of clauses* $\mathcal{K}$ *and* $R \subset \mathcal{L}$*, we define*

$$C^{max}_{R,asp}(\mathcal{K}) = \mathcal{K} \cup \{:\sim\ \mathtt{not}\ r. \mid r \in R\}$$

and in a similar way

$$C^{min}_{R,asp}(\mathcal{K}) = \mathcal{K} \cup \{:\sim r. \mid r \in R\} \ .$$

It is easy to verify that $C^{max}_R(\mathcal{K})$ corresponds to the answer sets of $C^{max}_{R,sat}(\mathcal{K})$ and $C^{min}_R(\mathcal{K})$ corresponds to the answer sets of $C^{min}_{R,sat}(\mathcal{K})$.

ASP also allows for encoding $S^{max}_R(\mathcal{K})$ and $S^{min}_R(\mathcal{K})$, but this requires rather involved, and often ad-hoc programs. A general approach has been presented in [17], but it relies on reification techniques, which is often detrimental for performance.

Let us now consider how to obtain $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$ and $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$ in ASP. For $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$, one requires for each solution in S that not all elements of R outside that solution should be false. This inhibits the solution itself and any subset (restricted to R) of it. For $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$, we require for each solution in S that not all of its elements in R should be true. In this way the solution itself and any superset is inhibited.

Definition 6 *Given a set of clauses* $\mathcal{K}$*,* $R \subset \mathcal{L}$*, and* $S \subseteq \sigma(\mathcal{K})$*, let*

$$\mathcal{N}^{\subseteq}_{asp}(\mathcal{K}, R, S) = \{:-\ \mathtt{not}\ \mathtt{a_1},\ \ldots,\ \mathtt{not}\ \mathtt{a_n}. \mid s \in S, R \setminus s = \{\mathtt{a_1}, \ldots, \mathtt{a_n}\}\}$$

and in a similar way

$$\mathcal{N}^{\supseteq}_{asp}(\mathcal{K}, R, S) = \{:-\ \mathtt{a_1},\ \ldots,\ \mathtt{a_n}. \mid s \in S, R \cap s = \{\mathtt{a_1}, \ldots, \mathtt{a_n}\}\} \ .$$

Again, it is easy to verify that $\mathcal{N}^{\subseteq}_{asp}(\mathcal{K}, R, S)$ and $\mathcal{N}^{\supseteq}_{asp}(\mathcal{K}, R, S)$ restrict the answer sets in the way required by $\mathcal{N}^{\subseteq}(\mathcal{K}, R, S)$ and $\mathcal{N}^{\supseteq}(\mathcal{K}, R, S)$.

4 METHODOLOGY SHOWCASE: ABSTRACT ARGUMENTATION

In this section we show how to use the proposed methodology for enumerating all preferred extensions of abstract argumentation frameworks. After a short background on abstract argumentation, we introduce a MaxSAT-based solver (**prefMaxSAT**) and an ASP-based solver (**prefASP**) that employ the methods described in Section 3.

4.1 Background on Abstract Argumentation

An argumentation framework [12] consists of a set of arguments[5] and a binary attack relation between them.

[5] In this paper we consider only *finite* sets of arguments: see [4] for a discussion on infinite sets of arguments.

$$\begin{tikzcd}\mathbf{c} \arrow[r] & \mathbf{b} \arrow[r, bend right] & \mathbf{a} \arrow[l, bend right]\end{tikzcd}$$

Figure 1. The $AF\ \Gamma_M$ for the hypertension problem

Definition 7 *An* argumentation framework *(AF) is a pair* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ *where* $\mathcal{A}$ *is a set of arguments and* $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$*. We say that* $\boldsymbol{b}$ attacks $\boldsymbol{a}$ *iff* $\langle \boldsymbol{b}, \boldsymbol{a} \rangle \in \mathcal{R}$*, also denoted as* $\boldsymbol{b} \rightarrow \boldsymbol{a}$*. The set of attackers of an argument* $\boldsymbol{a}$ *will be denoted as* $\boldsymbol{a}^- \triangleq \{\boldsymbol{b} : \boldsymbol{b} \rightarrow \boldsymbol{a}\}$*, the set of arguments attacked by* $\boldsymbol{a}$ *will be denoted as* $\boldsymbol{a}^+ \triangleq \{\boldsymbol{b} : \boldsymbol{a} \rightarrow \boldsymbol{b}\}$*.*

Each AF has an associated directed graph where the vertices are the arguments, and the edges are the attacks.

As an intuitive example from [6], let **a** be the argument "Patient has hypertension so prescribe diuretics;" **b**: "Patient has hypertension so prescribe betablockers;" and **c**: "Patient has emphysema which is a contraindication for betablockers." Intuitively, assuming that only one treatment is possible at the very same time, **a** attacks **b** and vice versa, while **c** suggests that **b** should not be the case (**c** attacks **b**). Therefore, let $\Gamma_M = \langle \mathcal{A}_M, \mathcal{R}_M \rangle$ such that, $\mathcal{A}_M = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and $\mathcal{R}_M = \{\langle \mathbf{c}, \mathbf{b} \rangle, \langle \mathbf{b}, \mathbf{a} \rangle, \langle \mathbf{a}, \mathbf{b} \rangle\}$. Γ_M is depicted in Fig. 1.

The basic properties of conflict–freeness, acceptability, and admissibility of a set of arguments are fundamental for the definition of argumentation semantics.

Definition 8 *Given an AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*:*

- *a set* $S \subseteq \mathcal{A}$ *is a* conflict–free *set of* Γ *if* $\nexists\, \boldsymbol{a}, \boldsymbol{b} \in S$ *s.t.* $\boldsymbol{a} \rightarrow \boldsymbol{b}$*;*
- *an argument* $\boldsymbol{a} \in \mathcal{A}$ *is* acceptable *with respect to a set* $S \subseteq \mathcal{A}$ *of* Γ *if* $\forall \boldsymbol{b} \in \mathcal{A}$ *s.t.* $\boldsymbol{b} \rightarrow \boldsymbol{a}$*,* $\exists\, \boldsymbol{c} \in S$ *s.t.* $\boldsymbol{c} \rightarrow \boldsymbol{b}$*;*
- *the function* $\mathrm{F}_\Gamma : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$ *such that* $\mathrm{F}_\Gamma(S) = \{\boldsymbol{a} \mid \boldsymbol{a}$ *is acceptable w.r.t.* $S\}$ *is called the* characteristic function *of* Γ*;*
- *a set* $S \subseteq \mathcal{A}$ *is an* admissible set *of* Γ *if* S *is a conflict–free set of* Γ *and every element of* S *is acceptable with respect to* S *of* Γ*.*

In the AF Γ_M of Fig. 1, $\{\mathbf{a}\}$ is an admissible set because it is conflict–free (there is no such attack $\langle \mathbf{a}, \mathbf{a} \rangle$) and each element of the set (i.e. **a**) is defended against the attack it receives (i.e. **a** is attacked by **b**, but, in turn, **b** is attacked by **a**).

An argumentation semantics S prescribes for any $AF\ \Gamma$ a set of *extensions*, denoted as $\mathcal{E}_{\mathsf{S}}(\Gamma)$, namely a set of sets of arguments satisfying the conditions dictated by S. Here we recall the definition of the grounded semantics, denoted as $\mathcal{GR}$, and of the preferred semantics, denoted as PR.

Definition 9 *Given an AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*:*

- *a set* $S \subseteq \mathcal{A}$ *is the* grounded extension *of* Γ*, if* S *is the least (w.r.t. set inclusion) fixed point of the characteristic function* F_Γ*;*
- *a set* $S \subseteq \mathcal{A}$ *is a* preferred extension *of* Γ*, i.e.* $S \in \mathcal{E}_{\mathsf{PR}}(\Gamma)$*, if* S *is a maximal (w.r.t.* $\subseteq$*) admissible set of* Γ*.*

While $\{\mathbf{a}\}$ is an admissible set for Γ_M, it is not a preferred extension. In fact, $\{\mathbf{c}, \mathbf{a}\}$ is also an admissible set which contains $\{\mathbf{a}\}$. Since there are no admissible supersets of $\{\mathbf{c}, \mathbf{a}\}$, it is therefore maximal and thus a preferred extension, the only one for Γ_M.

4.2 Admissible Extensions in SAT and ASP

As discussed in [5, 11] the search for admissible sets can be encoded using propositional logic formulae.

Definition 10 *Given an AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*,* $\mathcal{L}$ *a set of propositional variables, and* $v : \mathcal{A} \rightarrow \mathcal{L}$*, let* $admsat_\Gamma$ *be*

$$\bigwedge_{\boldsymbol{a} \in \mathcal{A}} \left(\left(v(\boldsymbol{a}) \supset \bigwedge_{\boldsymbol{b} \in \boldsymbol{a}^-} \neg v(\boldsymbol{b}) \right) \wedge \left(v(\boldsymbol{a}) \supset \bigwedge_{\boldsymbol{b} \in \boldsymbol{a}^-} \bigvee_{\boldsymbol{c} \in \boldsymbol{b}^-} v(\boldsymbol{c}) \right) \right)$$

The models of $admsat_\Gamma$ *correspond to the admissible sets of* Γ*.*

For the example AF Γ_M, $admsat_{\Gamma_M}$ is

$$\begin{array}{l}(v(\mathbf{a}) \supset \neg v(\mathbf{b})) \wedge (v(\mathbf{a}) \supset (v(\mathbf{a}) \vee v(\mathbf{c}))) \wedge \\ (v(\mathbf{b}) \supset (\neg v(\mathbf{a}) \wedge \neg v(\mathbf{c}))) \wedge (v(\mathbf{b}) \supset (v(\mathbf{b}) \wedge \bot)) \wedge \\ (v(\mathbf{c}) \supset \top) \wedge (v(\mathbf{c}) \supset \top)\end{array}$$

For ASP, an encoding for admissible extensions is rather straightforward, see [14, 11].

Definition 11 *Given an AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*, for each* $a \in \mathcal{A}$ *a fact*

```
arg(a).
```

is created and for each $(a, b) \in \mathcal{R}$ *a fact*

```
att(a, b).
```

is created (this corresponds to the apx file format in the ICCMA competition). Together with the program

```
in(X) :−not out(X), arg(X).
out(X) :−not in(X), arg(X).
:−in(X), in(Y), att(X, Y).
defeated(X) :−in(Y), att(Y, X).
not_defended(X) :−att(Y, X), not defeated(Y).
:−in(X), not_defended(X).
```

we form $admasp_\Gamma$ *and there is a one-to-one correspondence between answer sets of* $admasp_\Gamma$ *and admissible extensions.*

Different from the SAT encoding, the ASP encoding only changes facts for different AFs, the main program remains equal.

4.3 Preferred Extensions via Algorithm 1

We can now use our methodology in order to step from admissible to preferred extensions. Indeed, if $\mathcal{K}$ encodes admissible extensions of an AF and R is the language part encoding the extensions, then $S_R^{max}(\mathcal{K})$ encodes preferred extensions. We can then use Algorithm 1 to compute them. The two concretizations then give rise to two solvers, **prefMaxSAT** and **prefASP**, described in the following.

For **prefMaxSAT**, given an $AF\ \Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ and $v : \mathcal{A} \rightarrow \mathcal{L}$, we use Algorithm 1 with input $\mathcal{K}$ being $admsat_\Gamma$ and input R being the image of v, in the following referred to as $img(v)$. In lines 5 and 9 of Algorithm 1, we use a MaxSAT solver for obtaining one solution of $C^{max}_{img(v),sat}(\mathcal{K}_i)$. In line 8, $\mathcal{N}^{\subseteq}_{sat}(\mathcal{K}_i, img(v), S_i)$ is used.

For **prefASP**, given an $AF\ \Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, we use Algorithm 1 with input $\mathcal{K}$ being $admasp_\Gamma$ and input R being $\{\mathtt{in(a)} \mid \mathtt{a} \in \mathcal{A}\}$, in the following referred to as $I(\mathcal{A})$. In lines 5 and 9 of Algorithm 1, we use an ASP solver for obtaining all answer sets of $C^{max}_{I(\mathcal{A}),asp}(\mathcal{K}_i)$. In line 8, $\mathcal{N}^{\subseteq}_{asp}(\mathcal{K}_i, I(\mathcal{A}), S_i)$ is used.

Apart from the encodings and underlying solvers, there is also a difference in the fact that **prefASP** computes all cardinality-optimal solutions in one go, while **prefMaxSAT** computes one at a time.

5 EXPERIMENTAL ANALYSIS

In order to evaluate the efficiency of the introduced algorithms, we have carried out an experimental analysis where performance is analyzed from different perspectives. After describing the general setup, we first report on an experiment for choosing a parameter setting in the backend solver of **prefASP**. Next we report on a comparison of **prefASP** with **asprin** and **D-FLAT^2**. As discussed in the Introduction, these systems also support set optimization in an ASP setting, but use very different underlying algorithms. This is followed by a comparison to dedicated argumentation systems, in which we compare **prefASP** with **ASPARTIX-V**, since **ASPARTIX-V** is also based on ASP. Finally, we compare both **prefASP** and **prefMaxSAT** with **Cegartix**, which is the state-of-the-art solver, in the sense that it won the ICCMA2015 competition. These comparisons are deliberately split up into small pairings, in order to have a crisper picture of the relative performance measure. Indeed, the IPC score depends on the solvers considered in the comparison, while PAR10 and coverage are specific to a single solver. Experiments do not need to be re-run for presenting results together: however, comparisons would be less informative given the changes in the IPC score.

5.1 Experimental Settings

prefMaxSAT has been implemented in C++, and exploits the ASPino MaxSAT solver [1]. It should be noted that **prefMaxSAT** can be used with any MaxSAT system supporting the DIMACS format. The ASP-based algorithm **prefASP** has been implemented as a bash script using basic tools like sed and grep and exploits clingo 4.5.2 [16] as ASP solver.

The experiments were conducted on a cluster with computing nodes equipped with 2.5 GHz Intel$^{\text{TM}}$ Core 2 Quad Processors, 4 GB of RAM and Linux operating system. A cutoff of 900 seconds—wallclock time—was imposed to compute the preferred extensions for each AF. For each involved solver we recorded the overall result: success (if it finds each preferred extension), crashed, timed-out or ran out of memory. In fact, in our experimental evaluation all the unsuccessful runs are due to time-out. Experiments have been conducted on the ICCMA2015 benchmarks [23], which is a set of randomly generated 192 AFs. They have been generated considering three different graph models, in order to provide different levels of complexity. More details can be found on the ICCMA website.[6]

The performance measures reported in this paper are the Penalised Average Runtime and the International Planning Competition (IPC) score.

The Penalised Average Runtime (PAR score) is a real number which counts (i) runs that fail to find all the preferred extensions as ten times the cutoff time (PAR10) and (ii) runs that succeed as the actual runtime. PAR scores are commonly used in automated algorithm configuration, algorithm selection, and portfolio construction, because using them allows runtime to be considered while still placing a strong emphasis on high instance set coverage.

The IPC score, borrowed from the Planning community, is defined as follows:

- For each test case (in our case, each test AF) let T^* be the best execution time among the compared systems (if no system produces the solution within the time limit, the test case is not considered valid and ignored).
- For each valid case, each system gets a score of $1/(1 + \log_{10}(T/T^*))$, where T is its execution time, or a score of 0 if it fails in that case. Runtimes below 1 second get by default the maximal score of 1.

The IPC score for a system is the sum of its scores over all the valid test cases.

It should be noted that the IPC score depends on the ensemble of tested systems, that is, it is a relative measure, which depends on the experimental context. Indeed, in the following, the IPC scores of **prefASP** vary depending on the different experimental settings. In contrast to this, the PAR10 score of **prefASP** remains equal, as this is an absolute score.

[6] http://argumentationcompetition.org

5.2 Comparison of prefASP Using Different Solver Configurations

Clingo offers several different solver configurations (inherited from the clasp solver used in clingo) which correspond to different heuristic and search setups, see [15] for detailed explanations. As a first analysis, we investigated how robust **prefASP** is with respect to these configurations, and, as a by-product, we determined the best-performing configuration to be used for comparison with to other systems.

Table 1. Comparison of different Clingo solver configurations, that can be exploited by **prefASP**, on the ICCMA2015 benchmarks. Results are shown in terms of IPC score (maximum achievable is 192.0), percentages of success (% Success) and PAR10.

	IPC score	% Success	PAR10
Crafty	183.7	100.0	23.1
Frumpy	178.6	99.5	75.0
Jumpy	172.0	99.5	82.3
Many	177.4	99.5	84.8

Table 1 shows a comparison of the different solver configurations, in terms of IPC score, percentage of successfully analysed AFS and PAR10, on the ICCMA2015 benchmarks. It can be observed that all configurations perform relatively similar to each other, implying that the particular chosen configuration is not critical for the performance of **prefASP**.

However, there is one winning configuration: the **Crafty** configuration allows **prefASP** to enumerate preferred extensions of all the considered AFs, and to provide solutions faster. This configuration is geared towards "crafted" problems, which also makes sense in the context of the considered benchmarks. Therefore, in the rest of the experimental analysis the **Crafty** configuration will be considered for **prefASP**.

5.3 Comparison with Existing General Algorithms

We now turn to general tools that allow for easy representation and effective solution of subset optimization problems. To the best of our knowledge, the only tools of this kind are **asprin** [9] (with its predecessor **metasp**) and **D-FLAT^2** [8], which we have discussed in the Introduction.

Actually, **D-FLAT^2** uses the computation of preferred extensions as an example. When we followed the instruction provided by the authors[7], we were able to compute the preferred extensions of small AFs, but the system was already struggling with resource consumption on medium sized instances. On the ICCMA2015 benchmarks, the system ran out of memory very quickly, and we did not obtain any solutions for any of the ICCMA2015 benchmarks. This is probably due to the fact that ICCMA instances have large tree-width and **D-FLAT^2** relies heavily on tree decomposition. For this reason, in the remainder of this section we focus our comparison on **asprin**.

As **asprin** is based on ASP, the most natural comparison is against the ASP implementation of the proposed approach, namely **prefASP**. For **asprin**, we used clingo 4.5.2, the same version that is used as a backend for **prefASP**.

As input to **asprin** we use the program $admasp_\Gamma$ in Definition 11 together with the following preference definition:

```
#preference(p1, superset){
    in(X) : arg(X)
}.
#optimize(p1).
```

which makes **asprin** compute those answer sets, which are subset maximal for atoms with the predicate `in`.

Table 2. Comparison of **prefASP** and **asprin**, on the ICCMA2015 benchmarks. Results are shown in terms of IPC score (maximum achievable is 192.0), percentages of success (% Success) and PAR10.

	IPC score	% Success	PAR10
prefASP	191.2	100.0	23.1
asprin	157.8	100.0	44.9

The results of comparison between **asprin** and **prefASP** performed on the ICCMA2015 benchmarks for enumerating preferred extensions are shown in Table 2. Results indicate that the proposed **prefASP** system is significantly faster: **prefASP** achieves an IPC score of 191.2 versus 157.8 of **asprin**. According to the results, **prefASP** is the fastest system on 187 of the considered AFs. This is also confirmed by the PAR10 scores; on average **asprin** is about 20 seconds slower than **prefASP**, while in terms of coverage, both the considered systems are able to successfully analyse all the 192 AFs of the benchmark set. With regard to the observed performance difference, our conjecture is that **asprin** proves properties for more subsets of each answer set/extension than **prefASP** has to.

5.4 Comparison with Abstract Argumentation Algorithms Based on the Same Approaches

According to the results of ICCMA2015 [23], **ASPARTIX-V** [22] is the ASP-based abstract argumentation solver that showed the best performance in the preferred enumeration track.[8] Table 3 presents the results of a comparison between **prefASP** and **ASPARTIX-V** performed on the ICCMA2015 benchmarks. Presented results indicate that **prefASP** is faster, both in terms of IPC and PAR10 scores. Remarkably, **prefASP** is able to successfully analyse a larger number of AFs (100.0% against 94.0%).

Table 3. Comparison of **prefASP** and **ASPARTIX-V**, the ASP-based abstract argumentation solver that showed the best performance in the preferred enumeration track, on the ICCMA2015 benchmarks. Results are shown in terms of IPC score (maximum achievable is 192.0), percentages of success (% Success) and PAR10.

	IPC score	% Success	PAR10
prefASP	171.3	100.0	23.1
ASPARTIX-V	148.5	94.0	630.9

At a closer look, it is noticeable that—among ICCMA2015 frameworks—the AFs with a very large grounded extension and many nodes in general[9] are very challenging for **ASPARTIX-V**, while the proposed **prefASP** solver is able to quickly and effectively analyse also such large frameworks.

5.5 Comparison with the State of the Art Solver

In this analysis we compare **prefASP** and **prefMaxSAT** with the winner of the the ICCMA2015 track on enumerating preferred extensions, **Cegartix** [10]. Table 4 shows the performance of considered solvers in terms of IPC score, percentage of successfully analysed AFs and PAR10. **prefASP** performs significantly better than **prefMaxSAT**. This is possibly due to the fact that each preferred extension results from an execution of the MaxSAT solver; and a final run is needed in order to demonstrate that no other extensions exist. Therefore, the number of MaxSAT calls is exactly the number of preferred extensions plus one. The generated MaxSAT formulae tend to be large on the considered benchmarks; therefore, the added overhead can be remarkable.

Table 4. Comparison of **prefMaxSAT** and **prefASP** with the winner of the track of ICCMA2015 on enumerating preferred extensions, **Cegartix**. Results are shown in terms of IPC score (maximum 192.0), percentages of success and PAR10.

	IPC score	% Success	PAR10
prefASP	161.7	100.0	23.1
prefMaxSAT	115.3	85.0	1423.8
Cegartix	188.9	100.0	15.2

Interestingly, the performance of **prefASP** is comparable to the performance of **Cegartix**; according to PAR10, **prefASP** needs on average 8 seconds more to enumerate the preferred extensions. Moreover, by re-running the top solvers that took part in this track of ICCMA2015, we observed that **prefASP** would have been ranked second. This is an impressive achievement, considering that the described algorithm: (i) is very general, in the sense that it does not exploit any argumentation-specific knowledge; (ii) is very easy to implement, particularly in the ASP configuration; and (iii) has been implemented as a prototype, without attention on software engineering techniques for improving performance.

6 CONCLUSIONS AND FUTURE WORK

We have proposed a general methodology for solving subset optimality problems by means of iteratively solving cardinality optimality problems. This approach is motivated by the availability of efficient

[7] **D-FLAT^2** software and instructions have been retrieved from `https://github.com/hmarkus/dflat-2` in March 2016.

[8] We are not aware of any existing solver able to enumerate preferred extensions in a way directly comparable to **prefMaxSAT**.

[9] http://argumentationcompetition.org/2015/results.html

systems that support finding cardinality optimal solutions, namely MaxSAT solvers and ASP solvers supporting weak constraints.

As a methodology showcase we have produced two prototype systems, **prefMaxSAT** and **prefASP**, for enumerating preferred extensions of abstract argumentation frameworks. While the algorithms are general and easy to implement, an experimental analysis showed that they are competitive with the state-of-the-art system, which is specialized for this particular problem. On this showcase, our methods also prove higher performance than the existing general methods for computing subset optimal solutions of answer set programs, viz. **asprin** and **D-FLAT^2**.

There are quite many opportunities for future work. As discussed in [19], retain information between calls could help the performance, and we might investigate it in future work. However, this impact on the modularity of the concrete approach—i.e., a solver must be integrated in the framework—and can potentially reduce the generality of the overall framework.

Apart from tuning the prototype implementations of **prefMaxSAT** and **prefASP** to improve their performance, we intend to apply the methodology also to other application domains. Diagnosis or minimal model computation are immediate candidates. Another possibility is integrating our algorithm into a system like **asprin**, or one that supports the same input language.

The methodology would also allow for computing Σ_3^P-hard problems when using ASP, which would be interesting to explore, as it would give rise to alternatives to implementations relying on QBFs. It would also be worthwhile to explore whether the general methodology can be used also with formalisms different from MaxSAT and ASP, which would open entirely new avenues.

Given that there are a few similarities to Algorithm MCSes in [19], a comparison with the CAMUS system described in that paper could be attempted. Looking at the other direction, evaluating whether some implementation techniques in this area, e.g. in [21], can be generalized to our less specialized setting could be pursued as well.

Acknowledgements

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

The authors also thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] Mario Alviano, Carmine Dodaro, and Francesco Ricca, 'A maxsat algorithm using cardinality constraints of bounded size', in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pp. 2677–2683, (2015).

[2] Ofer Arieli and Martin W.A. Caminada, 'A QBF-based formalization of abstract argumentation semantics', *Journal of Applied Logic*, **11**(2), 229–252, (2013).

[3] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin, 'An introduction to argumentation semantics', *Knowledge Engineering Review*, **26**(4), 365–410, (2011).

[4] Pietro Baroni, Federico Cerutti, Paul E. Dunne, and Massimiliano Giacomin, 'Automata for Infinite Argumentation Structures', *Artificial Intelligence*, **203**(0), 104–150, (2013).

[5] Philippe Besnard and Sylvie Doutre, 'Checking the acceptability of a set of arguments', in *Proc. of the 10th Int. Workshop on Non-Monotonic Reasoning (NMR 2004)*, pp. 59–64, (2004).

[6] Philippe Besnard and Anthony Hunter, 'Constructing argument graphs with deductive arguments: a tutorial', *Argument & Computation*, **5**(1), 5–30, (2014).

[7] *Handbook of Satisfiability*, eds., Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009.

[8] Bernhard Bliem, Günther Charwat, Markus Hecher, and Stefan Woltran, 'D-FLAT^2: Subset minimization in dynamic programming on tree decompositions made easy', in *Eighth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2015)*, (2015).

[9] Gerhard Brewka, James P. Delgrande, Javier Romero, and Torsten Schaub, 'asprin: Customizing answer set preferences without a headache', in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, pp. 1467–1474, (2015).

[10] Rémi Brochenin, Thomas Linsbichler, Marco Maratea, Johannes Peter Wallner, and Stefan Woltran, 'Abstract solvers for Dung's argumentation frameworks', in *Proceedings of the International Workshop on Theory and Applications of Formal Argument (TAFA)*, (2015).

[11] Günther Charwat, Wolfgang Dvorák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran, 'Methods for solving reasoning problems in abstract argumentation A survey', *Artificial Intelligence*, **220**, 28–63, (2015).

[12] Phan M Dung, 'On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming, and n-Person Games', *Artificial Intelligence*, **77**(2), 321–357, (1995).

[13] Paul E. Dunne and Michael Wooldridge, 'Complexity of abstract argumentation', in *Argumentation in AI*, eds., I Rahwan and G Simari, chapter 5, 85–104, Springer-Verlag, (2009).

[14] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran, 'Answer-set programming encodings for argumentation frameworks', *Argument & Computation*, **1**(2), 147–177, (2010).

[15] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Javier Romero, and Torsten Schaub, 'Progress in clasp series 3', in *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2015)*, Lecture Notes in Artificial Intelligence, pp. 368–383, (2015).

[16] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub, '*Clingo* = ASP + control: Preliminary report', in *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP 2014)*, (2014).

[17] Martin Gebser, Roland Kaminski, and Torsten Schaub, 'Complex optimization in answer set programming', *Theory and Practice of Logic Programming*, **11**(4-5), 821–839, (2011).

[18] Chu Min Li and Felip Manya, 'Maxsat, hard and soft constraints.', *Handbook of Satisfiability*, **185**, 613–631, (2009).

[19] Mark H Liffiton and Karem A Sakallah, 'Algorithms for computing minimal unsatisfiable subsets of constraints', *Journal of Automated Reasoning*, **40**(1), 1–33, (2008).

[20] John McCarthy, 'Circumscription - A form of non-monotonic reasoning', *Artificial Intelligence*, **13**(1–2), 27–39, (1980).

[21] Antonio Morgado, Mark Liffiton, and Joao Marques-Silva, 'Maxsat-based mcs enumeration', in *Hardware and Software: Verification and Testing*, 86–101, Springer Berlin Heidelberg, (2012).

[22] Matthias Thimm and Serena Villata, 'System descriptions of the first international competition on computational models of argumentation (ICCMA-15)', *arXiv preprint arXiv:1510.05373*, (2015).

[23] Matthias Thimm, Serena Villata, Federico Cerutti, Nir Oren, Hannes Strass, and Mauro Vallati, 'Summary report of the first international competition on computational models of argumentation', *AI Magazine*, **37**(1), 102, (2016).

[24] Alice Toniolo, Timothy J. Norman, Anthony Etuk, Federico Cerutti, Robin Wentao Ouyang, Mani Srivastava, Nir Oren, Timothy Dropps, John A. Allen, and Paul Sullivan, 'Agent Support to Reasoning with Different Types of Evidence in Intelligence Analysis', in *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, (2015).

[25] Johannes Peter Wallner, Georg Weissenbacher, and Stefan Woltran, 'Advanced sat techniques for abstract argumentation', in *Proceedings of Computational Logic in Multi-Agent Systems: 14th International Workshop (CLIMA XIV)*, pp. 138–154, (2013).