Maximising Goals Achievement through Abstract Argumentation Frameworks: An Optimal Approach

Andrea Cohen[a], Sebastian Gottifredi[a], Mauro Vallati[b], Alejandro J. García[a], Grigoris Antoniou[b]

[a] *Institute for Computer Science and Engineering (CONICET-UNS)*
Department of Computer Science and Engineering, Universidad Nacional del Sur
San Andrés 800 - Campus de Palihue, (8000) Bahía Blanca, Argentina
[b] *School of Computing and Engineering, University of Huddersfield*
Huddersfield, HD1 3DH, United Kingdom

Abstract

Argumentation is a prominent AI research area, focused on approaches and techniques for performing common-sense reasoning, that is of paramount importance in a wide range of real-world applications, such as decision support and recommender systems. In this work we introduce an approach for updating an abstract Argumentation Framework (AF) so that achievement with respect to a given set of goals is maximised. The set of goals identifies arguments for which a specific acceptability status (a labelling) will be pursued, distinguishing between "`in`" and "`out`" goals. Given an AF, a set of goals and a set of available actions allowing to add or remove arguments and attacks from the AF, our approach will select the strategy (set of actions) that should be applied in order to obtain a new AF where the goals achievement is maximised. Moreover, the selected strategy will be optimal with respect to the number of actions to be applied. In the context of argumentation-based expert and intelligent systems, our approach will provide tools allowing the user to interact with the argumentative reasoning process carried out by the system, learning how the strategy she undertakes will affect the recommendations she receives. For that, we propose an encoding of the AF, the available actions and goals as weighted Boolean formulas, and rely on MaxSAT techniques for selecting the optimal strategy. We provide an experimental analysis of our approach, and formally show that the results we obtain correspond to the optimal strategy.

Keywords: abstract argumentation, argumentation dynamics, goals achievement, MaxSAT

1. Introduction

Argumentation is an attractive and effective paradigm for conceptualising common-sense reasoning (Bench-Capon and Dunne, 2007; Besnard and Hunter, 2008; Simari and Rahwan, 2009). Briefly, it is a form of reasoning where a piece of information (claim) is accepted or rejected after considering the reasons for and against that acceptance providing a reasoning mechanism capable of handling contradictory, incom-

Email addresses: `ac@cs.uns.edu.ar` (Andrea Cohen), `sg@cs.uns.edu.ar` (Sebastian Gottifredi), `m.vallati@hud.ac.uk` (Mauro Vallati), `ajg@cs.uns.edu.ar` (Alejandro J. García), `g.antoniou@hud.ac.uk` (Grigoris Antoniou)

plete and/or uncertain information. The argumentation process has been employed in various applications and domains such as decision making and negotiation (Black and Hunter, 2009; Ferretti et al., 2017), multi-agent systems (Thimm, 2014; Atkinson et al., 2016), and has led to the development of argumentation-based recommender and decision support systems (Chesñevar et al., 2009; Briguez et al., 2014; Bedi and Vashisth, 2014; Teze et al., 2015a; Gómez et al., 2016). There exist different approaches to argumentation-based reasoning, among which we can distinguish abstract (Dung, 1995) and structured (Besnard et al., 2014) ones. In particular, the interest in studying abstract Argumentation Frameworks (AFs) (Dung, 1995) has greatly increased over the years and several extensions of AFs have been proposed, incorporating support relations (Cayrol and Lagasquie-Schiex, 2013; Cohen et al., 2014) and recursive interactions (Baroni et al., 2011; Gottifredi et al., 2018), among the others.

Recently, the argumentation community has invested great effort in addressing efficiency issues surrounding the computation of extensions of AFs under different semantics, as witnessed by the *International Competition on Computational Models of Argumentation (ICCMA)*[1]. However, it is worth noting that those works mainly appear in the context of static argumentation, where the considered AFs are not allowed to change over time. Even though, in the last decade, there have been works considering the dynamics of AFs with regards to the arguments' acceptance, they were somewhat limited. For instance, in (Cayrol et al., 2008) the authors addressed the problem of updating the sets of extensions of an AF after an argument has been added; nevertheless, their proposal is restricted to the case of adding just one argument. As another example, Amgoud and Vesic (2009) studied the evolution in the acceptability of arguments when new arguments are incorporated, without requiring the computation of the whole extensions; however, again, only the situation were one argument is added was considered.

Within the past decade, works like (Liao et al., 2011; Baroni et al., 2014), (Doutre et al., 2014, 2017) and (Alfano et al., 2017) have gone further by considering a set of updates on the framework and the efficient re-computation of its extensions. Related to this line of work, there has also been growing interest in investigating the notion of extension enforcement (Doutre and Mailly, 2018). Briefly, works like (Coste-Marquis et al., 2015), (Niskanen et al., 2016) and (Baumeister et al., 2018) study how changes in an AF can help enforcing that a given set of arguments is (respectively, is contained in) an extension of the framework. Since these works have some common ground with our proposal, we will discuss them further in Section 6.

In this work we propose an approach for updating an AF driven by a set of goals, with the aim of maximising achievement with respect to those goals. Given an AF, goals identify arguments of the AF for which a specific acceptability status is wanted. In particular, we will distinguish between "`in`" and "`out`" goals, where the type of goal can be associated with the labelling of the corresponding arguments under a particular semantics; in other words, "`in`" and "`out`" goals can be associated with accepted and rejected arguments, respectively.

It should be noted that our approach is general, and agnostic with regards to the specific domain. It is therefore in the best position to serve a variety of application domains where argumentation is suitable for carrying out knowledge representation and reasoning tasks. Motivated by the developments on argumentation-based recommender systems (Briguez et al., 2014; Bedi and Vashisth, 2014; Teze et al., 2015b; Rodríguez

[1]`http://argumentationcompetition.org`

et al., 2017) and the ongoing need for providing users with ways to interact with the systems and their recommendations (He et al., 2016), in this paper we will consider an argumentation-based recommender system setting. These systems usually encode the knowledge used for building arguments through rules of the form premises-conclusion (*e.g.*, (Briguez et al., 2014)). In that way, the user obtains recommendations based on the set of accepted and rejected arguments of the system at a given state. As explained below, there is an inherently dynamic component in this kind of systems.

The inference rules used for building the arguments involved in the system reasoning process will not be applicable at every moment. Rather, the system will only be able to use a rule when it is capable of retrieving every piece of information present in the rule's premises; the rules that can be used at a given state can be identified as the active rules. Similarly, an argument built using a set of active rules is identified as active. As a result, in a given state of the system there will be a specific set of active arguments, and this set may vary from state to state. Moreover, since the set of accepted and rejected arguments is determined by considering only the set of active arguments, the sets of accepted and rejected arguments of the system may also vary from state to state.

Given the above mentioned setting, let us suppose that the user of such a system is not interested in obtaining a recommendation given the current state. Rather, she is interested in finding out what changes should be made to the current state in order to obtain the recommendations she desires. In other words, the user wants to know what actions she should undertake for the system to provide the recommendations she expects, which are specified in terms of the sets of arguments she wants to have as accepted or rejected. In that setting, we will abstract from the internal representation of arguments and assume that the knowledge handled by the system is represented through the use of AFs; briefly, an AF is defined by a set of arguments and an attack or conflict relation among them. On the other hand, the user goals will be expressed as arguments that will be tried to be made "`in`" or "`out`" in the corresponding AF. Then, in a particular state (represented as an AF), the user can check what goals are achieved by looking at the accepted arguments of the corresponding AF. Furthermore, in order to induce a change of state, the user has a set of actions she may perform. As a consequence, the application of actions will result in a new AF (obtained by adding/removing arguments and their associated attacks, as specified by the actions) that encodes the new state the system is in. To exemplify this, we consider the following scenario, which will serve as a running example throughout the rest of the paper.

Example 1. *Suppose that a hotel H is not being recommended by an expert system (consequently, the hotel is not receiving many guests) and the hotel administrator has to do something about it. Guests have stated that the hotel has uncomfortable beds, and that it is expensive. Also, it seems to be the case that the hotel yields no revenue. In order to achieve her goal of making the hotel H be considered as a good choice for guests, the administrator has to undertake some actions, while also avoiding to obtain zero revenue.*

It is worth mentioning that the user goals may be conflicting. For instance, given the above example, the administrator might choose to lower the rates so that the hotel becomes a good choice; however, in that case, the hotel revenue could decrease. As a result, in order to achieve some goal, the user might be forced to leave others aside. Furthermore, it could be the case that a user is not able to perform the course of action leading to achieve a particular goal, due to some external constraints. In this context, the user is in need of a strategy for selecting the actions leading her to perform her tasks, while maximising the achievement of her goals. To this end, we will exploit a weighted MaxSAT approach (Li and Manya, 2009), where the knowledge in the current state of the system (represented as an AF), and the available actions and goals of the user are encoded as weighted Boolean formulas. Thus, the proposed encoding for selecting the accepted arguments of the AF will account for the effect of the user actions. As part of the theoretical contribution of this paper, we will formally show the optimality of generated strategies, both in terms of the number of actions to be applied and in terms of the number of goals being achieved. Furthermore, we will provide an experimental analysis of the performance of our approach.

Our approach will contribute to expand the reasoning capabilities of expert and intelligent systems, particularly, argumentation-based recommender systems. As recalled in (He et al., 2016), the effectiveness of recommender systems goes beyond recommendation accuracy, and thus different methods and techniques facilitating the interaction between the user and the system have become of great relevance. Consequently, our contribution is to provide users of argumentation-based recommender systems with additional tools, indicating the set of actions that will have to be pursued in order to get the desired recommendations, so that they align with the users' goals. In that sense, our approach has the novelty and utility of allowing the user to interact with the argumentative reasoning process carried out by the recommender system, learning how the actions she undertakes will affect the recommendations she receives.

The rest of this work is organised as follows. In Section 2 we will start by introducing some background notions on argumentation that are required for developing our approach. Then, in Section 3 we will provide a formal characterisation of a system specification, its dynamics, and define the conditions under which the user goals are achieved. Section 4 introduces the MaxSAT approach for maximising goals achievement and minimising actions, including the specification of the algorithm we developed and formal results. Later, in Section 5 we present and discuss the results obtained in a large empirical analysis. Finally, in Section 6 we provide a discussion of related work and then move to Section 7 to draw some conclusions and comment on future lines of work.

2. Argumentation Background

As mentioned before, in this work we will consider that knowledge is represented through abstract Argumentation Frameworks (AFs) (Dung, 1995). Briefly, an AF is defined in terms of a set of arguments and a set of attacks among them.

Definition 1. *An abstract* Argumentation Framework (AF) *is a pair* $\langle \mathcal{A}, \mathcal{R} \rangle$ *where* $\mathcal{A}$ *is a finite and non-empty set of arguments and* $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ *is an attack relation.*

Arguments in an AF are abstract entities (their origin is not specified), and will be denoted using bold lower-case letters. The attack relation between two arguments $\mathbf{a}$ and $\mathbf{b}$ denotes the fact that these arguments cannot be simultaneously accepted, since they are conflicting. An argument $\mathbf{a}$ attacks an argument $\mathbf{b}$ iff $(\mathbf{a}, \mathbf{b}) \in \mathcal{R}$, noted as $\mathbf{a} \to \mathbf{b}$. The set of attackers of an argument $\mathbf{a}$ will be noted as $\mathbf{a}^- \triangleq \{\mathbf{b} \mid \mathbf{b} \to \mathbf{a}\}$. To illustrate this, let us consider the following example.

Example 2. *Let us consider the situation described in Example 1. In addition, suppose the administrator can lower the hotel rates, buy new beds to replace the old ones, buy slot machines to be placed in the hotel, or declare bankruptcy. To represent all the*

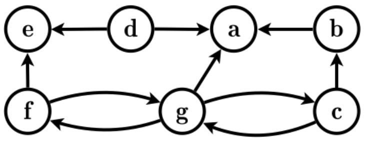

Figure 1: AF of Example 2

potential knowledge in the system, let us consider the following arguments and their conclusions:

- $\mathbf{a}$*: "Hotel H is a good choice "*
- $\mathbf{b}$*: "Hotel H has uncomfortable beds"*
- $\mathbf{c}$*: "Hotel H has brand-new beds"*
- $\mathbf{d}$*: "Hotel H is expensive"*
- $\mathbf{e}$*: "Hotel H yields no revenue"*
- $\mathbf{f}$*: "Hotel H has slot machines"*
- $\mathbf{g}$*: "Hotel H is bankrupt"*

We can characterise these arguments and the conflicts between them as an AF $\Omega_{Ex} = \langle\{\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{e},\mathbf{f},\mathbf{g}\},\{\mathbf{b}\to\mathbf{a},\mathbf{c}\to\mathbf{b},\mathbf{d}\to\mathbf{a},\mathbf{d}\to\mathbf{e},\mathbf{f}\to\mathbf{e},\mathbf{f}\to\mathbf{g},\mathbf{g}\to\mathbf{f},\mathbf{g}\to\mathbf{a},\mathbf{g}\to\mathbf{c},\mathbf{c}\to\mathbf{g}\}\rangle$*, depicted in Figure 1. In particular, arguments* $\mathbf{b}$ *and* $\mathbf{d}$ *attack argument* $\mathbf{a}$ *since they provide reasons against the hotel H being a good choice. Then, argument* $\mathbf{d}$ *also attacks argument* $\mathbf{e}$ *because if the hotel is expensive, then there are reasons to believe that it would be profitable (i.e., it would not be the case that it yields no revenue). Similarly, argument* $\mathbf{f}$ *attacks argument* $\mathbf{e}$ *because slot machines are known to be profitable. On the other hand, argument* $\mathbf{c}$ *attacks argument* $\mathbf{b}$ *since new beds are not supposed to be uncomfortable. Finally, note that arguments* $\mathbf{c}$*,* $\mathbf{f}$ *and* $\mathbf{g}$ *are linked to the effect of performing some of the administrator's actions (respectively, buy new beds, buy slot machines, and declare bankruptcy). Moreover, the attacks between* $\mathbf{c}$ *and* $\mathbf{g}$*, and between* $\mathbf{f}$ *and* $\mathbf{g}$ *express that the conflicting arguments cannot be simultaneously accepted, since the hotel would not have been able to buy any new goods if it were bankrupt.*

In abstract argumentation, the formal definition of methods ruling the arguments evaluation process corresponds to the characterisation of argumentation semantics. A semantics definition specifies how to obtain a set of extensions, where an extension is a set of arguments that can survive together or are considered to be collectively acceptable (Baroni and Giacomin, 2009). In (Dung, 1995), the semantics are defined in terms of the following notions.

Definition 2. *Let* $\Gamma = \langle\mathcal{A},\mathcal{R}\rangle$ *be an abstract Argumentation Framework and* $S \subseteq \mathcal{A}$*. Then:*

- *S is* conflict-free *iff* $\nexists\mathbf{a},\mathbf{b}\in S$ *s.t.* $(\mathbf{a},\mathbf{b})\in\mathcal{R}$*.*
- *An argument* $\mathbf{a}\in\mathcal{A}$ *is* acceptable *w.r.t. S iff* $\forall\mathbf{b}\in\mathcal{A}$ *s.t.* $(\mathbf{b},\mathbf{a})\in\mathcal{R}$*,* $\exists\mathbf{c}\in S$ *s.t.* $(\mathbf{c},\mathbf{b})\in\mathcal{R}$*.*

- *S is* admissible *iff S is conflict-free and every argument in S is acceptable w.r.t. S.*

Intuitively, an argument $\mathbf{a}$ is acceptable with regards to a set of arguments S if for every argument $\mathbf{b}$ that attacks $\mathbf{a}$, there is an argument $\mathbf{c}$ in S that attacks $\mathbf{b}$ (in which case $\mathbf{c}$ is said to defend $\mathbf{a}$). An admissible set of arguments S can then be interpreted as a coherent defensible position. For instance, given the AF Ω_{Ex} of Example 2, the sets $\{\mathbf{d}\}$ and $\{\mathbf{c}, \mathbf{f}\}$ are admissible whereas the set $\{\mathbf{a}, \mathbf{e}\}$ is not. Regarding the notions of admissibility and acceptability, let us consider the admissible set $\{\mathbf{c}, \mathbf{f}\}$ of Ω_{Ex}. It can be noted that adding argument $\mathbf{d}$ to that set will result in another admissible set. Then, we can keep adding arguments that are acceptable w.r.t. the resulting set, until every argument that can be defended by the set is in that set. Such a set will hold arguments that can stand on their own, and are associated to the notion of *complete extension* proposed in (Dung, 1995). In addition, if we look for maximal complete extensions of an AF, we obtain its *preferred extensions*.

A plethora of semantics for abstract argumentation frameworks has been proposed in the literature (Baroni and Giacomin, 2009; Charwat et al., 2015). In this paper, we will focus on the complete and preferred semantics proposed in (Dung, 1995), since they are comprehensive enough for us to study how the addition/removal of arguments (and the attacks they are involved in) affects the goals achievement. Among the four traditional semantics proposed by Dung (1995), preferred semantics is one of the most widely used in the literature of argumentation as it is less sceptical than grounded semantics, because the existence of extensions is always guaranteed (as opposed to stable semantics), and no extension is a proper subset of another extension (differently from complete semantics).

Definition 3. *Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an abstract Argumentation Framework and $S \subseteq \mathcal{A}$. Then:*

- *S is a* complete extension *of Γ iff S is an admissible set of Γ and all arguments that are acceptable w.r.t. S belong to S.*

- *S is a* preferred extension *of Γ iff S is a maximal (w.r.t. $\subseteq$) complete extension of Γ.*

Given the AF Ω_{Ex} of Example 2, we can obtain the following preferred extensions: $\mathcal{E}^1_{\Omega_{Ex}} = \{\mathbf{c}, \mathbf{d}, \mathbf{f}\}$ and $\mathcal{E}^2_{\Omega_{Ex}} = \{\mathbf{b}, \mathbf{d}, \mathbf{g}\}$. Note that none of the extensions simultaneously contains $\mathbf{f}$ and $\mathbf{g}$ or $\mathbf{c}$ and $\mathbf{g}$, capturing the conflicting nature of the administrator's actions. Then, for instance, extension $\mathcal{E}^2_{\Omega_{Ex}}$ containing argument $\mathbf{g}$ would correspond to a scenario where the administrator declared the bankruptcy of the hotel. Alternatively, extension $\mathcal{E}^1_{\Omega_{Ex}}$ containing arguments $\mathbf{c}$ and $\mathbf{f}$ would correspond to a scenario where the administrator acquired new goods for the hotel.

As shown in (Caminada and Gabbay, 2009), an extension of an AF Γ characterises a three-valued labelling of every argument in Γ. Using such a characterisation, argumentation semantics can be equivalently defined in terms of labellings (see *e.g.*, (Baroni et al., 2013)). In particular, the notion of complete labelling (Caminada and Gabbay, 2009) provides an equivalent characterisation of the complete semantics, establishing a one-to-one correspondence between complete labellings and complete extensions.

Definition 4. *Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an abstract Argumentation Framework. A total* *function* $\mathcal{L}ab : \mathcal{A} \mapsto \{\texttt{in}, \texttt{out}, \texttt{nondec}^2\}$ *is a* complete labelling *for* Γ *iff for every* $\mathbf{a} \in \mathcal{A}$ *it holds that:*

- $\mathcal{L}ab(\mathbf{a}) = \texttt{in}$ *iff* $\forall \mathbf{b} \in \mathbf{a}^-$, $\mathcal{L}ab(\mathbf{b}) = \texttt{out}$*;*
- $\mathcal{L}ab(\mathbf{a}) = \texttt{out}$ *iff* $\exists \mathbf{b} \in \mathbf{a}^-$ *s.t.* $\mathcal{L}ab(\mathbf{b}) = \texttt{in}$*; and*
- $\mathcal{L}ab(\mathbf{a}) = \texttt{nondec}$ *iff* $\nexists \mathbf{b} \in \mathbf{a}^-$ *s.t.* $\mathcal{L}ab(\mathbf{b}) = \texttt{in}$ *and* $\exists \mathbf{c} \in \mathbf{a}^-$ *s.t.* $\mathcal{L}ab(\mathbf{c}) = \texttt{nondec}$

It has been also shown in (Caminada and Gabbay, 2009) that the preferred extensions of an AF Γ are in one-to-one correspondence with those complete labellings of Γ that maximise the set of arguments labelled `in`. For instance, if we consider the preferred extension $\mathcal{E}^1_{\Omega_{Ex}}$ of Ω_{Ex}, the corresponding labelling is $\mathcal{L}ab(\mathbf{a}) = \texttt{out}$, $\mathcal{L}ab(\mathbf{b}) = \texttt{out}$, $\mathcal{L}ab(\mathbf{c}) = \texttt{in}$, $\mathcal{L}ab(\mathbf{d}) = \texttt{in}$, $\mathcal{L}ab(\mathbf{e}) = \texttt{out}$, $\mathcal{L}ab(\mathbf{f}) = \texttt{in}$, and $\mathcal{L}ab(\mathbf{g}) = \texttt{out}$.

3. System Characterisation

As discussed in the introduction, our approach is general and thus, it can be applied to a variety of application domains where argumentation has proved being useful.

For illustration purposes, in this paper we will consider an expert or recommender system setting, where the system's knowledge and the user goals and actions are represented by means of an AF and its components. In particular, a system will be characterised by a specification and a state. The system specification holds static information: an AF containing all conceivable arguments and the attacks between them, the available actions that the user may perform, and the goals the user has.

Definition 5. *A* specification *of a system for a user* $\mathtt{U}$ *is a tuple* $\mathsf{S}_\mathtt{U} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *where* $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$ *is an AF representing the potential knowledge stored within the system,* $\mathsf{Add}, \mathsf{Rem} \subseteq \mathcal{P}$ *are the sets of arguments that can be added and removed as the effect of actions performed by* $\mathtt{U}$*, and* $\mathcal{I}, \mathcal{O} \subseteq \mathcal{P}$ *are sets of arguments denoting the "*`in`*" and "*`out`*" goals the user* $\mathtt{U}$ *has.*

The first component in a system specification corresponds to the *potential knowledge*; that is, all the information that could be stored within the system. We refer to this knowledge as potential since it may be the case that the system has stored a piece of information (represented by a given argument) that is not currently available, for instance, because one of its premises is not available in the current state. Then, the user goals, and actions (represented in terms of the arguments they add or remove) are specified with respect to such potential knowledge.

Note that Definition 5 does not place any restriction on the sets of user goals and actions in a system specification. Therefore, it could be the case that, for instance, an argument $\mathbf{a}$ is simultaneously considered as an `in` and `out` goal. Similarly, it could be the case that the user has available actions for both adding and removing an argument $\mathbf{b}$. Even though situations like these do not introduce technical complications (for instance, in the case of having $\mathbf{a}$ as an `in` and `out` goal, the user would just have to choose one goal to achieve — as they cannot be simultaneously satisfied), for simplicity, we

[2] `nondec` corresponds to the `undec` labelling used in the literature of argumentation. We have renamed it in order to avoid misunderstandings with the notation of some concepts that will be introduced in Section 4.

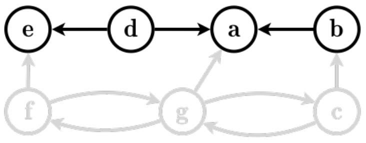

Figure 2: AF illustrating the system state Σ_{Ex} from Example 3

will assume that a system specification is coherent in the sense that $\mathsf{Add} \cap \mathsf{Rem} = \emptyset$ and $\mathcal{I} \cap \mathcal{O} = \emptyset$.

In a particular *state*, the system's active knowledge will be a subset of the potential knowledge; this is formalised as follows.

Definition 6. *Let* $\mathtt{S}_{\mathtt{U}} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *be the specification of a system for a user* $\mathtt{U}$*, with* $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$*. A* state *of* $\mathtt{S}_{\mathtt{U}}$ *is an AF* $\Sigma = \langle \mathcal{A}_{\Sigma}, \mathcal{R}_{\Sigma} \rangle$*, where* $\mathcal{A}_{\Sigma} \subseteq \mathcal{P}$ *is the set of available arguments in the state* Σ *and* $\mathcal{R}_{\Sigma} = \{(\mathbf{a}, \mathbf{b}) \mid \mathbf{a}, \mathbf{b} \in \mathcal{A}_{\Sigma},\ (\mathbf{a}, \mathbf{b}) \in \mathcal{R}\}$ *is the set of attacks between them.*

In other words, a state holds the information (arguments and attacks) that is available for the system in a particular moment. Then, given a system specification and a state, an argument from the system's potential knowledge is said to be `available` if it belongs to the set of arguments of that state; otherwise, it is said to be `unavailable`.

Example 3. *Consider the scenario presented in Examples 1 and 2. The administrator (user) has the goal of making the hotel H a good choice for guests. Therefore, her set of* `in`*-goals will be* $\mathcal{I} = \{\mathbf{a}\}$*, whereas her set of* `out`*-goals will be* $\mathcal{O} = \{\mathbf{e}\}$*. As shown in Example 2, the administrator can lower the hotel rates (which has the effect of removing argument* $\mathbf{d}$*), buy new beds to replace the old ones (add argument* $\mathbf{c}$*), buy slot machines to be placed in the hotel (add argument* $\mathbf{f}$*) or declare the hotel bankrupt (add argument* $\mathbf{g}$*). Thus, the system specification for the administrator in such a scenario will be* $\mathtt{S}_{\mathtt{U}_{Ex}} = (\Omega_{Ex}, \{\mathbf{c}, \mathbf{f}, \mathbf{g}\}, \{\mathbf{d}\}, \{\mathbf{a}\}, \{\mathbf{e}\})$*. In addition, if we consider that the current state of the system reflects the situation described in Example 1, the corresponding state in* $\mathtt{S}_{\mathtt{U}_{Ex}}$ *will be* $\Sigma_{Ex} = \langle \{\mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{e}\}, \{\mathbf{b} \to \mathbf{a}, \mathbf{d} \to \mathbf{a}, \mathbf{d} \to \mathbf{e}\} \rangle$*.*

The system state Σ_{Ex} from Example 3 is depicted in Figure 2. In particular, note that the available knowledge in Σ_{Ex} is a proper subset of the universal knowledge; the available arguments and attacks are highlighted in Figure 2.

Clearly, system inferences might change from state to state, depending on the available knowledge. Furthermore, the recommendations a user gets from the system in a given state will be determined by the arguments that end up accepted after an argumentative analysis carried out over the available knowledge in that state. Then, since the user goals are represented as arguments that will be tried to be made "`in`" or "`out`", their achievement will depend on the state the system is in. Briefly, the system will achieve a user `in`-goal $\mathbf{i}$ in a given state under a preferred extension if $\mathbf{i}$ belongs to the extension in that state. In contrast, a user `out`-goal $\mathbf{o}$ will be achieved by the system in a given state under a preferred extension if $\mathbf{o}$ is available in that state but does not belong to the extension.

Definition 7. *Let* $\mathtt{S}_{\mathtt{U}} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *be the specification of a system for a user* $\mathtt{U}$*,* Σ *a state of* $\mathtt{S}_{\mathtt{U}}$*, and* $\mathcal{E}$ *a preferred extension of* Σ*, with* $\mathcal{L}ab$ *its corresponding labelling:*

- $\mathtt{S_U}$ achieves an $\mathtt{in}$-goal $\mathbf{i} \in \mathcal{I}$ *in the state* Σ *under the extension* $\mathcal{E}$ *iff* $\mathcal{L}ab(\mathbf{i}) = \mathtt{in}$*; and*

- $\mathtt{S_U}$ achieves an $\mathtt{out}$-goal $\mathbf{o} \in \mathcal{O}$ *in the state* Σ *under the extension* $\mathcal{E}$ *iff* $\mathcal{L}ab(\mathbf{o}) = \mathtt{out}$.

Example 4. *Let us consider the specification* $\mathtt{S}_{\mathtt{U}_{Ex}}$ *and the state* Σ_{Ex} *from Example 3, corresponding to the situation described in Examples 1 and 2. In state* Σ_{Ex}*, the system's only preferred extension is* $\{\mathbf{d}, \mathbf{b}\}$*, under which the* $\mathtt{out}$*-goal* $\mathbf{e}$ *is achieved but the* $\mathtt{in}$*-goal* $\mathbf{a}$ *is not.*

As shown in Definition 5, a system specification for a user establishes, among other things, the actions (sets Add and Rem) the user is able to perform. These actions allow the user to induce a state change in the system, modifying the set of available arguments; however, it must be noted that the set of actions does not change from state to state. Hence, if the user chooses to apply an action that adds an argument $\mathbf{a}$, the system will reach a state where argument $\mathbf{a}$ is available. Furthermore, if $\mathbf{a}$ was already available, then the new state will coincide with the previous one (*i.e.*, the action will have no effect).

In our approach, we are abstracting from the internal structure of actions; hence, they are simply specified in terms of the arguments (and attacks) they affect. As a result, the presence of an argument in Add (respectively, Rem) represents that the user is able to perform an action that will render such argument as `available` (respectively, `unavailable`). Next, we characterise the effect of applying a set of actions in a given state the system is in.

Definition 8. *Let* $\mathtt{S_U} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *be the specification of system for a user* $\mathtt{U}$*,* $\Sigma = \langle \mathcal{A}_\Sigma, \mathcal{R}_\Sigma \rangle$ *a state of* $\mathtt{S_U}$*,* $\mathsf{Add}' \subseteq \mathsf{Add}$ *and* $\mathsf{Rem}' \subseteq \mathsf{Rem}$*. We define the effect of applying* Add' *and* Rem' *in the state* Σ *as* $\mathsf{Effect}(\mathsf{Add}', \mathsf{Rem}', \Sigma) = \Sigma'$*, where* $\Sigma' = \langle \mathcal{A}_{\Sigma'}, \mathcal{R}_{\Sigma'} \rangle$ *is a state of* $\mathtt{S_U}$ *s.t.* $\mathcal{A}_{\Sigma'} = (\mathcal{A}_\Sigma \cup \mathsf{Add}') \setminus \mathsf{Rem}'$ *and* $\mathcal{R}_{\Sigma'} = \{(\mathbf{a}, \mathbf{b}) \in \mathcal{R}_\Sigma \mid \mathbf{a}, \mathbf{b} \in \mathcal{A}_{\Sigma'}\}$.

Note that, even though actions are specified in terms of arguments, they also affect the attacks those arguments are involved in. That is, when applying a set of actions, the system reaches a state where the available knowledge is updated both in terms of arguments (by adding/removing arguments, as determined by the applied actions) and the attacks associated with those arguments (as the resulting state only includes attacks between available arguments).

Ideally, the user of a system that is in a particular state should strategically decide which actions to apply with the aim of achieving a desired state. Next, we define the notion of strategy for a user, which identifies a set of actions the user can apply in order to change the system's current state.

Definition 9. *Let* $\mathtt{S_U} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *be the specification of a system for a user* $\mathtt{U}$ *and* $\Sigma = \langle \mathcal{A}_\Sigma, \mathcal{R}_\Sigma \rangle$ *a state of* $\mathtt{S_U}$*. A* strategy *for* $\mathtt{U}$ *in* Σ *is defined as* $\mathsf{Strat}(\mathtt{S_U}, \Sigma) = (\mathsf{Add}', \mathsf{Rem}')$*, where* $\mathsf{Add}' \subseteq \mathsf{Add}$ *and* $\mathsf{Rem}' \subseteq \mathsf{Rem}$.

Example 5. *Let us consider the system specification* $\mathtt{S}_{\mathtt{U}_{Ex}}$ *and the state* Σ_{Ex} *from Example 3. If the administrator chooses to lower the hotel rates and buy new beds to replace the old ones, she will be selecting the strategy* $\mathsf{Strat}(\mathtt{S}_{\mathtt{U}_{Ex}}, \Sigma_{Ex}) = (\{\mathbf{c}\}, \{\mathbf{d}\})$*. The effect of applying the actions corresponding to that strategy in the state* Σ_{Ex} *is* $\mathsf{Effect}(\{\mathbf{c}\}, \{\mathbf{d}\}, \Sigma_{Ex}) = \Sigma'_{Ex}$*; a new state* $\Sigma'_{Ex} = \langle \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{e}\}, \{\mathbf{b} \to \mathbf{a}, \mathbf{c} \to \mathbf{b}\} \rangle$

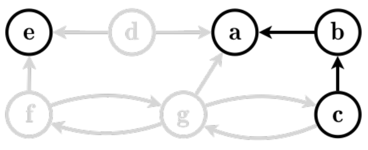

Figure 3: AF illustrating the system's state Σ'_{Ex} from Example 5

is obtained, depicted in Figure 3. Furthermore, note that state Σ'_{Ex} has only one preferred extension $\{\mathbf{a}, \mathbf{e}, \mathbf{c}\}$, under which the `in`*-goal* $\mathbf{a}$ *is achieved, but the* `out`*-goal* $\mathbf{e}$ *is not.*

As observed in the previous example, the user adopted a strategy leading to a state of the system where her goals are not fully achieved. In contrast, had she chosen a strategy where she also bought slot machines, every goal would have been achieved. Next, given a system specification for a user and a state, we will address the issue of obtaining an optimal strategy leading to a state where achievement of the user goals is maximised.

4. Strategy Generation

For the generation of user strategies, we propose a two-step approach that reduces the search for strategies to a sequence of weighted MaxSAT formulae to be solved. The algorithm is based on the idea of encoding the system specification for the user $\mathtt{S_U}$ and the current state Σ into a sequence of MaxSAT problems, in order to identify the maximum set of goals that can be simultaneously achieved while minimising the number of actions to be applied.

A weighted MaxSAT formula is a Boolean formula composed by hard constraints, that must be satisfied in every possible solution, and soft constraints. Each soft constraint has an associated weight. The MaxSAT problem consists of finding an assignment that maximises the sum of the weights of the satisfied clauses.

4.1. The MaxSAT Encoding

Given an AF $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$ representing the potential knowledge in a system specification, letting $k = |\mathcal{P}|$ we can identify each argument in $\mathcal{P}$ with an index in $\{1, \ldots k\}$ or, more precisely, we can define a bijection $\phi : \{1, \ldots, k\} \mapsto \mathcal{P}$ (the inverse map will be denoted as ϕ^{-1}). ϕ will be called an indexing of $\mathcal{P}$ and the argument $\phi(i)$ will be sometimes referred to as argument i for brevity. For each argument i we define four Boolean variables, I_i, O_i, N_i, and U_i, with the intended meaning that I_i is true when argument i is labelled `in` false otherwise, and analogously O_i, N_i, and U_i correspond to labels `out` `nondec` and `unavailable`. Formally, given $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$ we define the corresponding set of variables as $\mathcal{V}(\Omega) \triangleq \cup_{1 \le i \le k} \{I_i, O_i, N_i, U_i\}$. Remarkably, there is not an explicit labelling for describing an argument i as `available` since, in our encoding, i is considered as `available` if it is labelled as `in`, `out`, or `nondec`.

Definition 10. *Let* $\mathtt{S_U} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *be the specification of a system for a user* $\mathtt{U}$ *with* $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$ *and* $k = |\mathcal{P}|$, Σ *a state of* $\mathtt{S_U}$, *and* $\mathcal{L}ab$ *a labelling corresponding to a preferred extension of* Σ. *Let us also define, for the sake of conciseness and readability, as* `available` *a set of arguments* $\mathbf{a}$ *so that* $\{\mathbf{a} \in \mathcal{P} | \mathcal{L}ab(\mathbf{a}) = \mathtt{in} \vee \mathcal{L}ab(\mathbf{a}) =$

$\texttt{out} \vee \mathcal{L}ab(\mathbf{a}) = \texttt{nondec}\}$, *and* $\texttt{unavailable} = \mathcal{P} \backslash \texttt{available}$. *Given a propositional language and the definitions provided above, the weighted MaxSAT encoding for identifying a strategy* $\mathsf{Strat}(\mathtt{S_U}, \Sigma)$ *minimising the number of actions needed to achieve the user goals is given by the conjunction of clauses (1)–(31) listed below for each argument* $\phi(i)$*:*

$$\bigwedge_{i \in \{1,\ldots,k\}} \Big((I_i \vee O_i \vee U_i \vee N_i) \wedge (\neg I_i \vee \neg O_i) \wedge (\neg I_i \vee \neg N_i) \wedge (\neg I_i \vee \neg U_i) \wedge (\neg O_i \vee \neg N_i) \wedge (\neg O_i \vee \neg U_i) \wedge (\neg N_i \vee \neg U_i) \Big) \tag{1}$$

$$\bigwedge_{\{i \mid \phi(i) \in \texttt{available} \,\wedge\, \phi(i)^- = \emptyset\}} (I_i) \tag{2}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \emptyset\}} \left(I_i \vee U_i \vee \left(\bigvee_{\{j \mid \phi(j) \to \phi(i)\}} I_j \vee N_j \right) \right) \tag{3}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \emptyset\}} \left(\bigwedge_{\{j \mid \phi(j) \to \phi(i)\}} \neg I_i \vee O_j \vee U_j \right) \tag{4}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \emptyset\}} \left(\bigwedge_{\{j \mid \phi(i) \to \phi(j)\}} \neg I_i \vee O_j \vee U_j \right) \tag{5}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \emptyset\}} \left(\neg O_i \vee \left(\bigvee_{\{j \mid \phi(j) \to \phi(i)\}} I_j \right) \right) \tag{6}$$

$$\bigwedge_{\{i \mid \phi(i) \in \mathcal{I}\}} \Big(I_i \Big) \tag{7}$$

$$\bigwedge_{\{i \mid \phi(i) \in \mathcal{O}\}} \Big(O_i \Big) \tag{8}$$

$$\bigwedge_{\{i \mid \phi(i) \in (\texttt{available} \backslash \mathsf{Rem})\}} \Big(\neg U_i \Big) \tag{9}$$

$$\bigwedge_{\{i \mid \phi(i) \in (\texttt{unavailable} \backslash \mathsf{Add})\}} \Big(U_i \Big) \tag{10}$$

$$\bigwedge_{\{i \mid \phi(i) \in \mathsf{Add}\}} \Big(U_i \Big) \tag{11}$$

$$\bigwedge_{\{i \mid \phi(i) \in \mathsf{Rem}\}} \Big(\neg U_i \Big) \tag{12}$$

In order to encode the current state Σ *of the system, different sets of soft clauses are added, according to the current label of an argument* $\phi(i)$*. If* $\mathcal{L}ab(\phi(i)) = \texttt{in}$*, clauses (13)–(16) are added:*

$$\bigwedge_{i \in \{1,\ldots,k\}} \Big(I_i \Big) \tag{13}$$

$$\bigwedge_{i \in \{1,\ldots,k\}} \Big(\neg O_i \Big) \tag{14}$$

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg N_i\Big) \quad (15) \qquad \bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg U_i\Big) \quad (16)$$

If $\mathcal{L}ab(\phi(i)) = \texttt{out}$*, clauses (17)–(21) are added:*

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg I_i\Big) \quad (17) \qquad \bigwedge_{i\in\{1,\ldots,k\}} \Big(O_i\Big) \quad (18)$$

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg N_i\Big) \quad (19) \qquad \bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg U_i\Big) \quad (20)$$

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(I_i\Big) \quad (21)$$

If $\mathcal{L}ab(\phi(i)) = \texttt{nondec}$*, clauses (22)–(26) are added:*

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg I_i\Big) \quad (22) \qquad \bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg O_i\Big) \quad (23)$$

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(N_i\Big) \quad (24) \qquad \bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg U_i\Big) \quad (25)$$

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(I_i\Big) \quad (26)$$

If $\phi(i) \in \texttt{unavailable}$*, clauses (27)–(31) are added:*

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg I_i\Big) \quad (27) \qquad \bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg O_i\Big) \quad (28)$$

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(\neg N_i\Big) \quad (29) \qquad \bigwedge_{i\in\{1,\ldots,k\}} \Big(U_i\Big) \quad (30)$$

$$\bigwedge_{i\in\{1,\ldots,k\}} \Big(I_i\Big) \quad (31)$$

Finally, weights of clauses (11)–(31) are assigned maintaining the following proportion: sets (13)–(31) are assigned a weight of w*, and sets (11)–(12) are assigned a weight of* $5 \times w \times k$*.*

The set of clauses (1) ensures that exactly one label per argument is enforced. Set of clauses (2) settles the case of unattacked `available` arguments –that cannot be removed by any action– that must be labelled `in`. Note that there is no need to add similar clauses for the arguments that are currently labelled as `unavailable`, but that can be “availabilised” via actions. This comes from the MaxSAT optimisation: if an argument is made `available`, this is because it is needed to a achieve some goal and, in the case of unattacked arguments, the only way for affecting the extension is by being set `in`.

Set of clauses (3) states that if none of the attackers of argument i is labelled as `in` or `nondec`, then i must be `in` or `unavailable`. Set (4) encodes that if an argument i is `in`, then all of its attackers must be either `out` or `unavailable`. Similarly, set (5) encodes that if an argument i is `in`, the arguments attacked by i must be either `out` or `unavailable`. Set (6) encodes that if an argument is `out`, then one of its attackers must be `in`. It is worth noting that sets of clauses (1)–(6) are encoding the constraints of Definition 3 in terms of a complete extension, incorporating the notion of `unavailable` arguments, and with the additional condition that exactly one label must be assigned to each argument i.

The sets of clauses (7) and (8) are used to specify the goals the user has. Then, when searching for a strategy, every argument associated with an `in`-goal will have to be labelled as `in`. Analogously, every argument corresponding to an `out`-goal will have to be labelled as `out`.

Further hard constraints, specified by sets of clauses (9) and (10), are used to encode the fact that some arguments which are `available` (respectively, `unavailable`) cannot be removed (respectively, added) as the effect of actions performed by the user. Thus, an `available` argument i for which there is no Rem-action in the system specification for the user, can never be labelled as `unavailable`. On the other hand, if i is `unavailable` and there is no Add-action for it, then it must remain labelled as `unavailable`.

Sets of soft clauses (11) and (12) encode the actions available to the user, in terms of the arguments that can be added or removed from the system's knowledge (expressed as an AF). In particular, these sets of clauses encode the pre-state of arguments that may be subject to the actions performed by the user: in order for an argument to be added, it has to be labelled as `unavailable`; analogously, in order to be removed, an argument has not to be labelled as `unavailable`. In that way, only relevant actions are considered. It is important to note that the application of an action leads to unsatisfying the corresponding soft clause, thus reducing the quality of the solution. This mechanism prevents the solver from changing the availability of arguments that are not relevant for the sake of achieving the specified goals. Also, the weights assigned to sets of clauses (11) and (12) are equal, so that all actions have the same cost.

Finally, in order to encode the current state Σ of the system, different sets of soft clauses are added, according to the current label of each argument i: clauses (13)–(16) if i is initially labelled as `in`, (17)–(21) if i is initially labelled as `out`, (22)–(26) if i is initially labelled as `nondec` or (27)–(31) if i is initially labelled as `unavailable`.

It is important to notice the differences between the soft clauses used to encode arguments labelled as `in` in the current state Σ and the clauses used for representing arguments currently labelled as `out`, `nondec` or `unavailable`. In fact, four clauses are used for `in` arguments, and five clauses are used for the remaining cases. This is due to the fact that the MaxSAT encoding is aiming at maximising the number of arguments labelled as `in`, in cases where the current label of an argument has to be changed in order to achieve some predefined goals. For arguments currently labelled as `in`, any change in the labelling of such arguments will result in a reduction of the weight of satisfied soft clauses. In particular, given a weight per clause of 1, changing the label of an `in` argument will reduce the score achieved by clauses (13)–(16) from 4 to 2. In that, the MaxSAT solver will try to maintain the current labelling of such `in` arguments in order to maximise the overall quality of the solution. For arguments that are not labelled as `in` in the current state, an additional soft clause is needed. As an example, let us focus on arguments labelled as `out`. Soft clauses (17)–(20), which all have a weight of 1, encode the preference for maintaining the current label. For cases in which this is not possible, then the additional soft clauses (21) encode the preference for labelling arguments `in`, if possible. Considering only clauses (17)–(21): if the argument maintains the same label, the overall score achieved is 4. If the argument is labelled as `in`, the score is then 3; any other label would result in a lower score of 2. This is also the case for arguments labelled as `nondec` and `unavailable` in the current state Σ.

Sets (11)–(12) and sets (13)–(31) are, to some extent, complementary. The former sets explicitly encode the actions that the user can employ with the aim of achieving her goals. The latter sets are used to evaluate the impact that an action has on the current state of the system. It should be noted that the weight assigned to sets (11)–(12) should be significantly larger than weight of sets (13)–(31): in this way, the user will minimise the number of actions and, in cases where different actions allow to satisfy the goals, the action with the smallest impact on the framework is selected. Continuing with our example, clauses (11)–(12) would have a weight of $5 \times k$; in this way, the cost of applying an action is higher than the cost of changing the labellings of all the considered arguments. This is because the application of an action would imply that the corresponding clause (11) or (12) ceases to be satisfied, losing its weight. Therefore, actions are applied only when strictly necessary. If, instead, weights of sets (11)–(12) and (13)–(31) are similar, the user will not minimise the number of actions, but will minimise the overall impact on the current state of the system, ignoring the actual number of actions exploited.

4.2. The Proposed Algorithm

We are now in the position to illustrate the proposed procedure, listed in Algorithm 1, to generate optimal strategies for the user.

Algorithm 1 requires as input the elements in a system specification for a user: the argumentation framework Ω —which encodes the potential knowledge stored within the system— the set of available actions for the user $A_\Omega = \mathsf{Add} \cup \mathsf{Rem}$, the sets of user `in`-goals $\mathcal{I}$ and `out`-goals $\mathcal{O}$. Also, it takes as input an initial labelling L_i —which distinguishes what arguments are initially `in`, `out`, `nondec` or `unavailable` given an initial state of the system and a preferred extension of the AF encoding that state. The provided output is a labelling L_m which maximises the achievement of the user goals, and at the same time minimises the number of actions to be undertaken.

Initially, in lines 1–3 of Algorithm 1, it is checked whether it is possible for the agent to achieve all her goals. This can be checked by generating an appropriate MaxSAT formula, where goals are encoded using hard constraints. If the formula is satisfiable, then the MaxSAT encoding proposed in the previous section will return the labelling L_m allowing to generate the appropriate strategy $\mathsf{Strat}(\mathtt{S}_\mathtt{U}, \Sigma)$ minimising the number of actions needed to achieve every goal.

In cases where total achievement is not possible, the proposed procedure identifies the maximum set (w.r.t. size) of goals that can be achieved. This is done in lines 5–7, and corresponds to generating a MaxSAT formula where sets of clauses (13)–(31) are omitted, sets of clauses (7) and (8) are encoded as *soft* clauses with a high weight, and sets of clauses (11) and (12) have a very small weight. In this way, the focus is given to the maximisation of goals achievement. Once the largest achievement set has been identified, it is established as the new set of goals for the user, and a new MaxSAT formula is generated to obtain the labelling L_m (lines 11–12) and then create the strategy $\mathsf{Strat}(\mathtt{S}_\mathtt{U}, \Sigma)$. In this formula, only the goals identified in lines 5–7 are encoded by setting their corresponding *hard* clauses (7) and (8).

Algorithm 1 The proposed approach for maximising achievement of $\mathcal{I}, \mathcal{O}$.

```
Input: $\Omega$, $A_\Omega$, $L_i$, $\mathcal{I}$, $\mathcal{O}$
Output: $L_m$
 1: $F$=formulaAchieveAll($\Omega, A_\Omega, L_i, \mathcal{I}, \mathcal{O}$)
 2: if satisfiable($F$) then
 3:     $L_m$ = MaxSAT($F$)
 4: else
 5:     $F_c$=formulaMaxAchievement($\Omega, A_\Omega, L_i, \mathcal{I}, \mathcal{O}$)
 6:     $L_c$ = MaxSAT($F_c$)
 7:     $\mathcal{I}_s, \mathcal{O}_s$= maximisedAchievement($L_c$)
 8:     if $\mathcal{I}_s = \emptyset$ and $\mathcal{O}_s = \emptyset$ then
 9:         return unsatisfiable
10:     end if
11:     $F$=formulaAchieveAll($\Omega, A_\Omega, L_i, \mathcal{I}_s, \mathcal{O}_s$)
12:     $L_m$ = MaxSAT($F$)
13: end if
14: return $L_m$
```

It may occur that none of the user goals can be achieved. This case is handled in lines 8–9, and an *unsatisfiable* value is returned by the algorithm.

Finally, given a system specification for a user $\mathtt{S_U} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ with $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$, and a labelling L_i corresponding to a preferred extension of a state Σ of $\mathtt{S_U}$, the strategy $\mathsf{Strat}(\mathtt{S_U}, \Sigma) = (\mathsf{Add}', \mathsf{Rem}')$ is obtained from Algorithm 1 as follows: for each argument $\mathbf{a} \in \mathcal{P}$ labelled as `unavailable` in L_i and labelled as `in`, `out` or `nondec` in L_m, it holds that $\mathbf{a} \in \mathsf{Add}'$; for each argument $\mathbf{b}$ labelled as `in`, `out` or `nondec` in L_i and labelled as `unavailable` in L_m, it holds that $\mathbf{b} \in \mathsf{Rem}'$.

Next, we will show that Algorithm 1 indeed returns a strategy that minimises the number of actions and leads to a state of the system where there exists a preferred extension maximising the goals achievement. For that purpose, given a system specification for a user $\mathtt{S_U}$ and a labelling L_i corresponding to a preferred extension $\mathcal{E}$ of a state Σ the system is in, the number of goals being achieved in Σ under $\mathcal{E}$ will be identified as $\mathsf{Ach}(\mathtt{S_U}, \Sigma, L_i)$. We start by showing that the strategy obtained form Algorithm 1 is optimal in the sense that it maximises the number of user goals that are achieved.

Theorem 1. *Let* $\mathtt{S_U} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *be the specification of a system for a user* $\mathtt{U}$ *with* $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$*,* Σ *be a state of* $\mathtt{S_U}$*, and* L_i *a labelling corresponding to a preferred extension of* Σ*. Given the labelling* L_m *returned by Algorithm 1, the strategy* $\mathsf{Strat}(\mathtt{S_U}, \Sigma) = (\mathsf{Add}', \mathsf{Rem}')$ *obtained from Algorithm 1, and* $\mathsf{Effect}(\mathsf{Add}', \mathsf{Rem}', \Sigma) = \Sigma'$*, it holds that: there is no strategy* $\mathsf{Strat}(\mathtt{S_U}, \Sigma) = (\mathsf{Add}'', \mathsf{Rem}'')$*, with* $\mathsf{Effect}(\mathsf{Add}'', \mathsf{Rem}'', \Sigma) = \Sigma''$*, s.t. there is a labelling* L''_m *corresponding to a preferred extension of* Σ'' *and* $\mathsf{Ach}(\mathtt{S_U}, \Sigma'', L''_m) > \mathsf{Ach}(\mathtt{S_U}, \Sigma', L_m)$*.*

Proof. If every goal in the agent's specification can be achieved, then the labelling L_m will be returned by Algorithm 1 in line 3. Therefore, there will be no strategy leading to a higher level of achievement than the one found by Algorithm 1. Otherwise, if total achievement is not possible, Algorithm 1 maximises the number of goals that can be simultaneously achieved by setting them as soft constraints instead of hard constraints (lines 5–7). Then, Algorithm 1 returns labelling L_m by setting the sets of goals identified in line 7 as hard constraints (lines 11-12). As a result, since labelling L_m allows to achieve every goal obtained in line 7, there is no other strategy leading to a higher level of achievement. $\square$

Theorem 2 then shows that Algorithm 1 leads to obtaining an optimal strategy with respect to the number of actions to be applied.

Theorem 2. *Let* $\mathtt{S_U} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *be the specification of a system for a user* $\mathtt{U}$ *with* $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$*,* Σ *be a state of* $\mathtt{S_U}$*, and* L_i *a labelling corresponding to a preferred extension of* Σ*. Given the labelling* L_m *returned by Algorithm 1, the strategy* $\mathsf{Strat}(\mathtt{S_U}, \Sigma) = (\mathsf{Add}', \mathsf{Rem}')$ *obtained from Algorithm 1, and* $\mathsf{Effect}(\mathsf{Add}', \mathsf{Rem}', \Sigma) = \Sigma'$*, it holds that: if there is a strategy* $\mathsf{Strat}(\mathtt{S_U}, \Sigma) = (\mathsf{Add}'', \mathsf{Rem}'')$*, with* $\mathsf{Effect}(\mathsf{Add}'', \mathsf{Rem}'', \Sigma) = \Sigma''$*, and a labelling* L''_m *corresponding to a preferred extension of* Σ'' *s.t.* $\mathsf{Ach}(\mathtt{S_U}, \Sigma'', L''_m) = \mathsf{Ach}(\mathtt{S_U}, \Sigma', L_m)$*, then* $|\mathsf{Add}'' \cup \mathsf{Rem}''| >= |\mathsf{Add}' \cup \mathsf{Rem}'|$.

Proof. By Theorem 1, the labelling L_m maximises the number of user goals that can be achieved. Then, when the calls to MaxSAT are made in line 3 or 12, Algorithm 1 ensures that the selected goals (respectively, the sets $\mathcal{I}$ and $\mathcal{O}$, or the sets $\mathcal{I}_s$ and $\mathcal{O}_s$) can be fully achieved. Therefore, sets of clauses (7)–(8) will be set as hard constraints and the solver will focus on minimising the number of actions to be undertaken by considering the weights assigned to the sets of clauses (11)–(12) and (13)–(31). By Definition 10, weights of clauses (11)–(12) are $5 \times w \times k$, with w being the weight of clauses (13)–(31) and $k = |\mathcal{P}|$. Then, when the current status of an argument cannot be maintained to achieve every goal, the MaxSAT solver will choose (if possible) to change its labelling without moving it from `available` (`in`, `out`, `nondec`) to `unavailable` or vice-versa, as the cost reduction of doing so will be smaller than that of applying an Add/Rem action on that argument. As a result, the solver will find the assignment of values (the labelling L_m) requiring the minimum number of actions (hence, the lower reduction in the quality of the solution) to be applied in order to achieve every goal. $\square$

Finally, Theorem 3 shows that by applying the strategy obtained from Algorithm 1 the system reaches a state where there exists a preferred extension corresponding to the labelling returned by Algorithm 1.

Theorem 3. *Let* $\mathtt{S_U} = (\Omega, \mathsf{Add}, \mathsf{Rem}, \mathcal{I}, \mathcal{O})$ *be the specification of a system for a user* $\mathtt{U}$ *with* $\Omega = \langle \mathcal{P}, \mathcal{R} \rangle$*,* Σ *be a state of* $\mathtt{S_U}$*, and* L_i *a labelling corresponding to a preferred extension of* Σ*. Given the labelling* L_m *returned by Algorithm 1, the strategy* $\mathsf{Strat}(\mathtt{S_U}, \Sigma) = (\mathsf{Add}', \mathsf{Rem}')$ *obtained from Algorithm 1, and* $\mathsf{Effect}(\mathsf{Add}', \mathsf{Rem}', \Sigma) = \Sigma'$*, it holds that* L_m *is the labelling of a preferred extension of* Σ'.

Proof. Sets of clauses (1)–(6) in Definition 10 characterise complete labellings (Cerutti et al., 2013). Then, since Algorithm 1 sets them as hard constraints for the MaxSAT solver, the returned labelling L_m is a complete labelling. The encoding of sets of clauses (13)–(31) with equal weights results in that, when the current labelling of an argument is `out` or `nondec` and cannot be maintained (respectively, clauses (18) and (24)), then the solver will try to label that argument as `in`. This is because shifting from the current labelling to `in` would result in satisfying three clauses ((19)–(21) in the case of `out` arguments, and (23), (25)–(26) in the case of `nondec` arguments), whereas shifting to `nondec`, `out` (depending on the initial label) or `unavailable` would lead to satisfy only two clauses; furthermore, in the case of shifting to `unavailable`, the quality of the solution will be reduced even further as it would require to apply a Rem

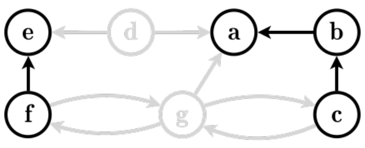

Figure 4: AF illustrating the system's state Σ''_{Ex} from Example 6

action, losing the weight of the previously satisfied clause (12). On the other hand, in the case of an argument labelled as `unavailable` whose label cannot be maintained (thus, the application of an Add action is strictly necessary) the solver will also try to choose the `in` labelling for the same reason as with the `out` and `nondec` labellings. Thus, since by Theorem 2 the strategy obtained from Algorithm 1 keeps the number of actions to a minimum, the MaxSAT solver will maximise the `in` labellings and L_m will be a labelling corresponding to a preferred extension of Σ'; this is because, by Definition 3, maximising the number of arguments labelled as `in` leads to a maximal (w.r.t $\subseteq$) complete extension. $\square$

Next, we illustrate the outcome of Algorithm 1, when applied to the running example.

Example 6. *Let us consider the system specification* $\mathtt{S}_{\mathtt{U}_{Ex}}$ *and the state* Σ_{Ex} *from Example 3. Suppose we apply Algorithm 1 with the input parameters* Ω*,* A_Ω*,* $\mathcal{I}$ *and* $\mathcal{O}$ *being taken from the specification* $\mathtt{S}_{\mathtt{U}_{Ex}}$*, and the labelling* L_i *corresponding to the preferred extension of the state* Σ_{Ex}*:* $\mathcal{L}ab(\mathbf{a}) = \mathtt{out}$*,* $\mathcal{L}ab(\mathbf{b}) = \mathtt{in}$*,* $\mathcal{L}ab(\mathbf{d}) = \mathtt{in}$*,* $\mathcal{L}ab(\mathbf{e}) = \mathtt{out}$*. Then, Algorithm 1 returns the labelling* L_m*:* $\mathcal{L}ab(\mathbf{a}) = \mathtt{in}$*,* $\mathcal{L}ab(\mathbf{b}) = \mathtt{out}$*,* $\mathcal{L}ab(\mathbf{c}) = \mathtt{in}$*,* $\mathcal{L}ab(\mathbf{e}) = \mathtt{out}$*,* $\mathcal{L}ab(\mathbf{f}) = \mathtt{in}$*. Note that the labelling* L_m *corresponds to the preferred extension* $\{\mathbf{a}, \mathbf{c}, \mathbf{f}\}$ *of state* $\Sigma''_{Ex} = \langle\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{e}, \mathbf{f}\}, \{\mathbf{b} \to \mathbf{a}, \mathbf{c} \to \mathbf{b}, \mathbf{f} \to \mathbf{e}\}\rangle$*, illustrated in Figure 4. Specifically, the optimal strategy obtained from the labelling* L_m *(output of Algorithm 1) is* $\mathsf{Strat}(\mathtt{S}_{\mathtt{U}_{Ex}}, \Sigma_{Ex}) = (\mathsf{Add}'', \mathsf{Rem}'')$*, with* $\mathsf{Add}'' = \{\mathbf{c}, \mathbf{f}\}$ *and* $\mathsf{Rem}'' = \{\mathbf{d}\}$*; in addition, note that* $\mathsf{Effect}(\mathsf{Add}'', \mathsf{Rem}'', \Sigma_{Ex}) = \Sigma''_{Ex}$*. Finally, as expressed by the labelling* L_m*, the user* `in`*-goal* $\mathbf{a}$ *and the* `out`*-goal* $\mathbf{e}$ *are achieved in the state* Σ''_{Ex}*, reached by applying the optimal strategy* $(\mathsf{Add}'', \mathsf{Rem}'')$*.*

5. Experimental Analysis

We implemented the proposed approach using a mix of C++, for generating the encodings, and Python. Our current implementation exploits the open-wbo MaxSAT framework (Martins et al., 2014), using glucose3.0 (Audemard and Simon, 2014).

For the purposes of testing, we selected 50 AFs from the benchmarks available for the ICCMA 2015 and 2017. We focused on AFs with different underlying structures, by covering all categories considered by the competition benchmarks. We selected AFs among the smallest and medium-size instances of each category, leading to consider AFs with a proportion of $\sim 10\times$ number of attacks with respect to the number of arguments for the 2015 benchmarks, and $\sim 3\times$ for the 2017 ones; the number of arguments ranges between 189 and $1,880$, whereas the number of attacks goes from 221 to $35,559$.

***D*=50, *A*=30, *G*=20 (20.0)**				
Approach	Runtime	Cov.	Ach. Imp.	Actions
MaxSAT	1.6	98.0	14.5 (4.6)	24.7 (20.7)
Enumeration	21.3	86.0	6.5 (5.9)	79.2 (62.9)
Brute-Force	–	0.0	–	—
Randomised	0.6	98.0	-0.5 (7.2)	24.7 (20.7)
***D*=50, *A*=45, *G*=20 (18.1)**				
Approach	Runtime	Cov.	Ach. Imp.	Actions
MaxSAT	4.3	90.0	21.6 (4.5)	34.6 (21.6)
Enumeration	56.2	74.0	9.4 (7.0)	110.6 (77.5)
Brute-Force	–	0.0	–	—
Randomised	0.7	90.0	0.7 (2.7)	34.6 (21.6)
***D*=50, *A*=30, *G*=40 (19.7)**				
Approach	Runtime	Cov.	Ach. Imp.	Actions
MaxSAT	7.5	88.0	13.2 (5.9)	35.2 (24.8)
Enumeration	53.9	80.0	4.0 (9.9)	72.4 (52.3)
Brute-Force	–	0.0	–	—
Randomised	0.4	88.0	0.8 (5.4)	35.2 (24.8)

Table 1: Performance of the proposed approach (MaxSAT) and the three baselines on the considered benchmarks. Results are presented in terms of average CPU-time (seconds), coverage, average achievement improvements over the initial achievement level (standard deviation), and average number of applied actions (standard deviation). Averages are computed by considering only instances solved by at least three approaches.

We generated different sets of experiments by considering three parameters: the deactivation quota D —which refers to the percentage of unavailable arguments in the initial state— the percentage of arguments to be considered as goals G, and the percentage of arguments that can be made available or unavailable via actions A. Given an AF and the value of the D, G, and A parameters, the system specification for a user $\mathtt{S_U}$ is randomly generated using a specifically-developed Prolog approach.

A strategy $\mathtt{Strat}(\mathtt{S_U}, \Sigma)$ is then obtained using the proposed approach (hereinafter, MaxSAT) and three baseline approaches: *Enumeration, Brute-Force, and Randomised*. Enumeration is based on the idea of enumerating all the preferred extensions of the largest possible state the system can reach, and then selecting the preferred extension that maximises the achievement of goals. In a nutshell, it simulates cases where no reasoning can be done in terms of actions, and existing solvers —in this case ArgTools (Nofal et al., 2014), that is also used to provide the initial labelling L_i— are used to determine the level of achievement of a set of goals. Brute-Force exploits a typical brute force approach, testing all the possible combinations of actions in order to identify the combination that maximises goals achievement. Finally, Randomised exploits the knowledge of the number of actions to be applied with MaxSAT: it applies the same number of actions, but randomly selected, enumerates the preferred extensions of the resulting state and selects the one maximising goals achievement. Note that, for the three baseline approaches, the chosen preferred extension (labelling) is then used to obtain the corresponding strategy following the methodology described in Section 4.2.

Experiments were performed on Intel i5 2.80GHz with 8GB RAM. Each approach run was limited to a single core. The cut-off time was set to 600 CPU-time seconds.

Table 1 compares the performance of the proposed MaxSAT approach and the three considered baseline approaches on the 50 AFs benchmarks when using D=50, A=30, and G=20 parameters' value. Achievement improvement is calculated by considering the initial achievement level, which corresponds to the achievement level under the initial preferred extension of Σ, where no actions have been applied. Unsurprisingly, Brute-Force is never able to increase the initial achievement within the given cut-off time. It is also worth noting that an unwise actions selection can lead to dramatical reduction in the achievement level, as testified by the performance of Randomised. MaxSAT is generally fast in providing the optimal strategies, that significantly improve the initial achievement levels.

Table 1 also reports the results of experiments where, in turn, the value of A and G has been increased, in order to test cases where a larger number of actions is available, and where a higher number of goals needs to be satisfied. A higher number of actions leads to a general increment in the average runtime of the considered approaches, but allows MaxSAT to furtherly increase the achievement improvement. Conversely, the higher the number of goals to be satisfied is, the harder it is to increase achievement levels. This was also confirmed by a set of additional experiments in which the A and G values were decreased: that led to an easier strategy generation process. Overall, the proposed approach demonstrated to be able to quickly generate optimal strategies for the considered benchmarks. Only in few cases, 7 across all the experiments run, it runs out of CPU-time before returning a strategy, as shown by the coverage in Table 1.

In order to evaluate the impact of the deactivation quota value on the runtime performance of the proposed approach, we run additional experiment by fixing the values of A and G to, respectively, 30 and 20, while changing the value of D between 30, 50, and 70. Figure 5 shows how the PAR10 of the considered approaches is affected by the value of D. We omitted Brute-Force results as it did not solve any of the testing instances.

Results shown in Figure 5 indicate that a higher percentage of de-activated arguments leads to smaller states of the system, that can be analysed faster by the approaches. On the contrary, a lower D value results in larger and more complex states. Nevertheless, the proposed MaxSAT approach showed to be able to quickly provide optimal strategies.

6. Related Work

Recently, the community of argumentation has gained interest in studying the computation of extensions in AFs that may change dynamically. On the one hand, there exist works aimed at efficiently computing extensions after changes in an AF have been produced, such as (Baroni et al., 2014) and (Alfano et al., 2017). When an AF is updated, the division-based method of Baroni et al. (2014) separates the updated AF into two parts, *affected* and *unaffected*, where the set of affected arguments consists of those reachable from the added/removed arguments. Then, existing outcomes of extension computation in the unaffected sub-AF can be reused and only the acceptability status of affected arguments needs to be recomputed after updates.

Based on the work by Baroni et al. (2014), Alfano et al. (2017) propose an incremental approach for efficiently re-computing a given extension of such an AF, given a set of updates to be applied to the AF known a-priori. For that purpose, they first identify a sub-AF, called *reduced AF*, on the basis of the influenced set (the portion of the original AF affected by the updates) and additional information provided by the initial extension. Then, they use any non-incremental algorithm to compute an extension of

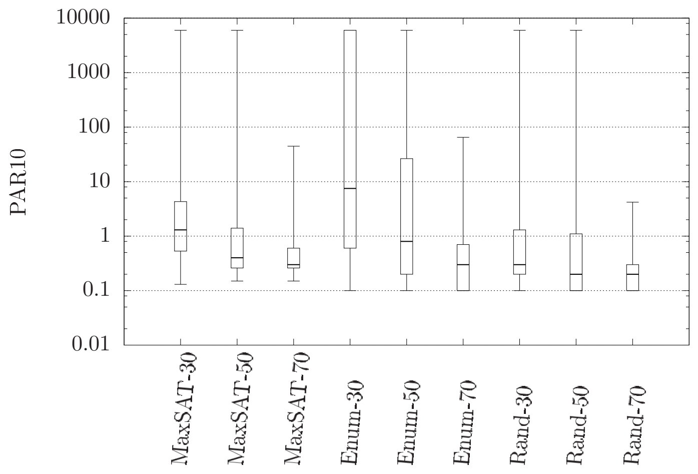

Figure 5: PAR10 performance variability of MaxSAT, Enumeration (Enum) and Randomised (Rand) approaches with regards to 30, 50, and 70% deactivation quota values. Whiskers refer to the best and worst performance achieved by the system, while the box indicates the first, second, and third quartiles.

the reduced AF. Finally, they obtain the resulting extension by merging a portion of the initial extension with that computed for the reduced AF.

Although, similarly to Baroni et al. (2014) and Alfano et al. (2017), our approach deals with the dynamics behind the re-computation of extensions in updated AFs, there are differences between them. In particular, our approach does not rely on a set of updates (*i.e.*, the set of add/remove actions that can be performed over the AF) provided beforehand; in contrast, our proposal has the aim of identifying the optimal set of actions the user should perform in order to induce a change of state in the system (represented by an updated AF) where achievement of her goals is maximised. Furthermore, the selected course of action (following the terminology in Alfano et al. (2017), the update) will be minimal in terms of the number of actions to be applied.

Another line of work addressing dynamics in argumentation frameworks concerns the laying of theoretical foundations for adding/removing arguments and/or attacks from AFs. As an example, Doutre et al. (2014, 2017) provide the grounds for updating an AF by adding/removing attacks or arguments, as well as changing the acceptance status of arguments. They represent an AF using propositional logical formulas, together with constraints encoding a semantics, and describe how to update the AF in terms of operations over these formulas by means of propositional dynamic logic. Similarly to us, their approach aims at minimising the changes needed to guarantee the acceptance of one or more arguments. However, the main difference between (Doutre et al., 2014, 2017) and our work is that, given a set of possible changes (available actions), a user in our approach will select the course of action (strategy) leading to maximise achievement of her goals, even in the case where total achievement is not possible.

Coste-Marquis et al. (2015) propose an approach where it is possible to add arguments, and also add or remove attacks between arguments of an AF in order to enforce the existence of a given extension under a particular semantics (*strict enforcement*); alternatively, when strict enforcement cannot be achieved, they propose a *non-strict* version of *enforcement* that aims at finding an extension that contains a given set of arguments. It should be noted that enforcement in (Coste-Marquis et al., 2015) can only be performed on the set of original arguments of the AF (*e.g.*, new arguments cannot be enforced to belong to an extension of the modified framework). In contrast, in our approach, the sets of `in`-goals and `out`-goals are defined in terms of the potential knowledge; hence, for instance, arguments that are originally unavailable can be considered as goals as well. Regarding the notion of enforcement, our approach can be considered to be closer to *non-strict enforcement*, when considering the `in`-goals. Furthermore, when the set of goals cannot be satisfied simultaneously, our approach allows to maximise (in terms of quantity) the set of goals that can be simultaneously achieved. On the other hand, it should be noted that our approach not only accounts for "positive" enforcement (*i.e.*, the presence of arguments in a given extension) but also for "negative" enforcement; in other words, for the `out`-goals we seek a specific labelling to be assigned on arguments: we require the corresponding arguments to be part of the available knowledge within a state, but to be labelled as `out`.

It is important to note that, when characterising the addition of arguments and the addition or removal attacks, Coste-Marquis et al. (2015) do not specify why those changes can be made. On the one hand, the origin of the new arguments is not specified (it is not known where do they come from, or whether they could exist at all). On the other hand, the addition or removal of attacks between arguments in their approach can be made arbitrarily. For instance, one could just remove all attacks between arguments in order to enforce the desired extension. Notwithstanding this, it is important to note that similarly to us, Coste-Marquis et al. (2015) follow a minimal-change approach, where modifications to an AF are tried to be kept to a minimum, and exploited only when strictly necessary; however, this does not change the fact that the choice for the addition or removal of elements within the AF is arbitrary. In our approach, the modifications that are allowed to be made on an AF are specified in terms of the set of actions that can be performed; that is, they specify what arguments from the potential knowledge (and their associated attacks, when corresponding) can be added or removed from the AF that encodes a given state. As a result, arguments and attacks that do not belong to the AF encoding the potential knowledge will never be allowed to be added or removed. In this regard also note that, differently from Coste-Marquis et al. (2015), our actions can have the effect of removing arguments.

In (Niskanen et al., 2016) the authors focus on the status enforcement of arguments. Specifically, they account for *accepted* and *rejected* statuses of arguments, similarly to our specification of arguments within the sets of `in`-goals and `out`-goals. Like in our approach, they allow changes in an AF to be made, and enforcement is then considered over the modified AF. However, differently from us, the only changes allowed into an AF are the addition and removal of attacks, and these can be made arbitrarily like in (Coste-Marquis et al., 2015). As another remarkable difference, their approach also characterises sceptical and credulous acceptance and rejection of sets of arguments. In particular, for instance, to enforce the credulous acceptance of a set of arguments, they look for an AF whose union of extensions contains the given set of arguments. That is, they do not enforce that the set of arguments as a whole is contained in the same extension of the AF. Hence, it could be the case that the arguments within the given set are separately contained in different extensions of the AF. In contrast, our approach guarantees to reach a state (encoded as an AF) where, when possible, the entire set of `in`-goals is satisfied. In other words, we ensure that the set of arguments corresponding to the `in`-goals will be contained in the same extension of the AF.

The work presented in (Baumeister et al., 2018) tackles the problem of expressing incomplete knowledge in AFs. Similarly to us, it considers a notion of uncertain knowledge (following our terminology, potential knowledge) expressed in terms of sets of uncertain arguments and attacks, leading to the existence of argument-incomplete AFs, attack-incomplete AFs, or the combination of both. In that paper, the authors seek to investigate how the modifications on AFs and the different kinds of uncertainty (on the set of arguments, on the set of attacks, or both) affects the complexity of verification problems; for instance, given an AF and a set of arguments S, they consider the following question: is there a completion AF* (where AF* is the modified version of AF after adding some uncertain arguments and/or attacks) such that S is an extension of AF*? Note that their approach does not focus on what characteristics such a completion should have (*e.g.*, if it adds the least possible amount of arguments and/or attacks); rather, their interest relies on the computational cost of obtaining completions, with the aim of making the process as efficient as possible.

Considering the dynamics of AFs, our approach focuses on obtaining a new AF (which encodes a new state) by applying a minimal number of actions expressed in terms of additions/removals of arguments and attacks on the original AF (encoding the previous state) so that the new AF maximises the achievement of the established sets of goals. Moreover, like in the previous cases, we can highlight the following differences between our approach and the one proposed by Baumeister et al. (2018): whereas their approach only allows to add arguments and attacks, our approach also allows to remove arguments and their associated attacks. Also, the problem of finding a completion such that a given set of arguments is one of its extensions, addressed by Baumeister et al. (2018), can be considered to be similar to the (positive) strict enforcement problem from (Coste-Marquis et al., 2015). On the other hand, as discussed before, our approach handles both “positive” and “negative” enforcement.

7. Discussion and Concluding Remarks

In this work we proposed a combined argumentation-MaxSAT approach for maximising the achievement of the user goals, given the specification of a system. Furthermore, our approach allows the user to select a strategy for reaching the desired state of the system, requiring the minimum number of actions to be undertaken; thus, our approach is optimal with regards to two different dimensions: goals achievement and number of actions. We provided a formal analysis of the correctness of our approach, and we empirically showed that it outperforms the three baseline approaches it was tested against.

It should be noted that our approach is general but, as also illustrated in the paper, could be applied in contexts where argumentation is used to perform reasoning within expert or recommender systems. Furthermore, our approach can complement such systems, providing a decision-support mechanism for the user. This is because our approach allows the user to identify what actions she should perform in order to induce a change of state in the system, so that she obtains the recommendations she expects.

Following the discussion in Section 6, we can conclude that our approach is complementary to existing works in the literature of argumentation that deal with efficient re-computation of extensions in dynamic AFs, as well as those addressing the issue of extension enforcement. Future work will include an experimental evaluation of the proposed approach on complex real-world scenarios, and the possibility for the user to challenge her goals. That is, it would be interesting to argumentatively reason about the validity of the user goals. Then, for instance, a user may be able to find arguments to challenge the fact that $\mathtt{g}$ is a goal and, in that case, the acceptability status of $\mathtt{g}$ will not be relevant for determining the level of achievement.

Furthermore, we plan to extend our approach to account for priorities between the user goals in cases where total achievement is not possible. To that end, when turning sets (7) and (8) into soft clauses, their weights would have to be adjusted to reflect the priorities so that the higher the weight of a clause is, the higher the importance of the goal it encodes. As a result, the strategy obtained from Algorithm 1 would lead the system to achieve the set of user goals that provides the best balance between quantity and priority; this is because the MaxSAT solver will search for the solution that maximises the sum of the weights associated with the soft clauses encoding the agent's goals.

Related to the above, the literature of argumentation offers some approaches where the elements in an AF can be associated with different weights or strengths (Bench-Capon, 2003; Amgoud et al., 2017), which can be considered to extend our approach. In particular, Value-based Argumentation Frameworks (Bench-Capon, 2003) extend Dung's AFs by mapping each argument in the framework to a value. Hence, the strength of an argument depends on the social values it advances, and determining whether the attack of one argument on another succeeds depends on the comparative strength of the values advanced by the arguments concerned. Similarly, (Amgoud et al., 2017) incorporates a weighting function w which associates each argument in the framework with a real number in the interval $[0, 1]$, where $w(\mathbf{a})$ represents the intrinsic strength of argument $\mathbf{a}$. Then, the authors focus on the characterisation of different acceptability semantics which satisfy a set of principles, with the aim to establish the overall strength of each argument in the framework. Specifically, as a perspective for future work, we could consider one of these extended frameworks and make use of the weights associated with arguments to, as mentioned before, establish priorities between the user goals. In that way, the weights of the goal arguments will have a direct impact in the solution obtained through our combined argumentation-MaxSAT approach.

Finally, it should be noted that our approach currently has the limitation that each action only allows for the addition or removal of one argument, respectively. As a perspective for future work, we also plan to generalise the characterisation of actions in two complementary directions. On the one hand, we plan to redefine the actions so that they allow for the addition and/or removal of multiple arguments. In that sense, they would be more closely related to the characterisation of actions in domains like planning. On the other hand, we intend to include new kinds of actions enabling the addition and removal of attacks between arguments in the framework. In this way, we would bring our approach even more close to others like (Alfano et al., 2017), where updates to an AF concerning the addition and/or removal of attacks are considered.

Acknowledgments

This work has been partially supported by EU H2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 690974 for the project "MIREL: MIning and REasoning with Legal texts", and by CONICET and Universidad Nacional del Sur, Argentina, under PGI-UNS grants 24/N046 and 24/ZN32.

References

Alfano, G., Greco, S., and Parisi, F. (2017). Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pages 49–55.

Amgoud, L., Ben-Naim, J., Doder, D., and Vesic, S. (2017). Acceptability semantics for weighted argumentation frameworks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 56–62.

Amgoud, L. and Vesic, S. (2009). On revising argumentation-based decision systems. In *ECSQARU*, volume 5590 of *Lecture Notes in Computer Science*, pages 71–82. Springer.

Atkinson, K., Cerutti, F., McBurney, P., Parsons, S., and Rahwan, I. (2016). Special issue on argumentation in multi-agent systems. *Argument & Computation*, 7(2-3):109–112.

Audemard, G. and Simon, L. (2014). Lazy clause exchange policy for parallel sat solvers. In *Theory and Applications of Satisfiability Testing–SAT 2014*, pages 197–205.

Baroni, P., Cerutti, F., Dunne, P. E., and Giacomin, M. (2013). Automata for infinite argumentation structures. *Artif. Intell.*, 203:104–150.

Baroni, P., Cerutti, F., Giacomin, M., and Guida, G. (2011). AFRA: argumentation framework with recursive attacks. *Int. J. Approx. Reasoning*, 52(1):19–37.

Baroni, P. and Giacomin, M. (2009). Semantics of abstract argument systems. In *Argumentation in Artificial Intelligence*, pages 25–44.

Baroni, P., Giacomin, M., and Liao, B. (2014). On topology-related properties of abstract argumentation semantics. A correction and extension to dynamics of argumentation systems: A division-based method. *Artif. Intell.*, 212:104–115.

Baumeister, D., Neugebauer, D., Rothe, J., and Schadrack, H. (2018). Verification in incomplete argumentation frameworks. *Artif. Intell.*, 264:1–26.

Bedi, P. and Vashisth, P. B. (2014). Empowering recommender systems using trust and argumentation. *Inf. Sci.*, 279:569–586.

Bench-Capon, T. J. M. (2003). Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13(3):429–448.

Bench-Capon, T. J. M. and Dunne, P. E. (2007). Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641.

Besnard, P., García, A. J., Hunter, A., Modgil, S., Prakken, H., Simari, G. R., and Toni, F. (2014). Introduction to structured argumentation. *Argument & Computation*, 5(1):1–4.

Besnard, P. and Hunter, A. (2008). *Elements of Argumentation*. MIT Press.

Black, E. and Hunter, A. (2009). An inquiry dialogue system. *Autonomous Agents and Multi-Agent Systems*, 19(2):173–209.

Briguez, C. E., Budán, M. C., Deagustini, C. A. D., Maguitman, A. G., Capobianco, M., and Simari, G. R. (2014). Argument-based mixed recommenders and their application to movie suggestion. *Expert Syst. Appl.*, 41(14):6467–6482.

Caminada, M. W. A. and Gabbay, D. M. (2009). A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145.

Cayrol, C., de Saint-Cyr, F. D., and Lagasquie-Schiex, M. (2008). Revision of an argumentation system. In *KR*, pages 124–134. AAAI Press.

Cayrol, C. and Lagasquie-Schiex, M. (2013). Bipolarity in argumentation graphs: Towards a better understanding. *Int. J. Approx. Reasoning*, 54(7):876–899.

Cerutti, F., Dunne, P. E., Giacomin, M., and Vallati, M. (2013). Computing preferred extensions in abstract argumentation: A sat-based approach. In *Theory and Applications of Formal Argumentation - Second International Workshop, TAFA 2013, Beijing, China, August 3-5, 2013, Revised Selected papers*, pages 176–193.

Charwat, G., Dvorák, W., Gaggl, S. A., Wallner, J. P., and Woltran, S. (2015). Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63.

Chesñevar, C. I., Maguitman, A. G., and González, M. P. (2009). Empowering recommendation technologies through argumentation. In *Argumentation in Artificial Intelligence*, pages 403–422.

Cohen, A., Gottifredi, S., García, A. J., and Simari, G. R. (2014). A survey of different approaches to support in argumentation systems. *Knowledge Eng. Review*, 29(5):513–550.

Coste-Marquis, S., Konieczny, S., Mailly, J., and Marquis, P. (2015). Extension enforcement in abstract argumentation as an optimization problem. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2876–2882.

Doutre, S., Herzig, A., and Perrussel, L. (2014). A dynamic logic framework for abstract argumentation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*.

Doutre, S., Maffre, F., and McBurney, P. (2017). A dynamic logic framework for abstract argumentation: Adding and removing arguments. In *Advances in Artificial Intelligence: From Theory to Practice - 30th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2017, Arras, France, June 27-30, 2017, Proceedings, Part II*, pages 295–305.

Doutre, S. and Mailly, J. (2018). Constraints and changes: A survey of abstract argumentation dynamics. *Argument & Computation*, 9(3):223–248.

Dung, P. M. (1995). On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming, and n-Person Games. *Artif Intell*, 77(2):321–357.

Ferretti, E., Tamargo, L. H., García, A. J., Errecalde, M. L., and Simari, G. R. (2017). An approach to decision making based on dynamic argumentation systems. *Artif. Intell.*, 242:107–131.

Gómez, S. A., Goron, A., Groza, A., and Letia, I. A. (2016). Assuring safety in air traffic control systems with argumentation and model checking. *Expert Syst. Appl.*, 44:367–385.

Gottifredi, S., Cohen, A., García, A. J., and Simari, G. R. (2018). Characterizing acceptability semantics of argumentation frameworks with recursive attack and support relations. *Artif. Intell.*, 262:336–368.

He, C., Parra, D., and Verbert, K. (2016). Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities. *Expert Syst. Appl.*, 56:9–27.

Li, C. M. and Manya, F. (2009). Maxsat, hard and soft constraints. *Handbook of Satisfiability*, 185:613–631.

Liao, B. S., Jin, L., and Koons, R. C. (2011). Dynamics of argumentation systems: A division-based method. *Artif. Intell.*, 175(11):1790–1814.

Martins, R., Manquinho, V., and Lynce, I. (2014). Open-wbo: a modular maxsat solver. In *Theory and Applications of Satisfiability Testing–SAT 2014*, pages 438–445.

Niskanen, A., Wallner, J. P., and Järvisalo, M. (2016). Optimal status enforcement in abstract argumentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1216–1222.

Nofal, S., Atkinson, K., and Dunne, P. E. (2014). Algorithms for decision problems in argument systems under preferred semantics. *Artif. Intell.*, 207:23–51.

Rodríguez, P., Heras, S., Palanca, J., Poveda, J. M., Duque, N. D., and Julián, V. (2017). An educational recommender system based on argumentation theory. *AI Commun.*, 30(1):19–36.

Simari, G. R. and Rahwan, I., editors (2009). *Argumentation in Artificial Intelligence*. Springer.

Teze, J. C., Gottifredi, S., García, A. J., and Simari, G. R. (2015a). Improving argumentation-based recommender systems through context-adaptable selection criteria. *Expert Syst. Appl.*, 42(21):8243–8258.

Teze, J. C., Gottifredi, S., García, A. J., and Simari, G. R. (2015b). Improving argumentation-based recommender systems through context-adaptable selection criteria. *Expert Syst. Appl.*, 42(21):8243–8258.

Thimm, M. (2014). Strategic argumentation in multi-agent systems. *KI*, 28(3):159–168.