Efficient Computation of Extensions for Dynamic Abstract Argumentation Frameworks: An Incremental Approach

GIANVINCENZO ALFANO, SERGIO GRECO, FRANCESCO PARISI

Department of Informatics, Modeling, Electronics and System Engineering, University of Calabria, ITALY

{g.alfano, greco, fparisi}@dimes.unical.it

ABSTRACT ARGUMENTATION

An *(abstract) argumentation framework* (*AF*) is a pair $\langle A, \Sigma \rangle$, where A is a set of *arguments* and $\Sigma \subseteq A \times A$ is a set of *attacks*.

- It allows representing dialogues, making decisions, and handling inconsistency;
- An AF can be viewed as a direct graph, whose nodes are arguments and whose edges are attacks.

SEMANTICS FOR AFS

An argumentation semantics specifies the criteria for identifying "reasonable" sets of arguments, called *extensions*.

A *complete extension* (co) is an admissible set that contains all the arguments that it defends.

A complete extension S is said to be:

- *preferred (pr)* iff it is maximal (w.r.t. $\subseteq$);
- *stable (st)* iff it attacks all the arguments in $A \setminus S$;
- *grounded (gr)* iff it is minimal (w.r.t. $\subseteq$).

UPDATES

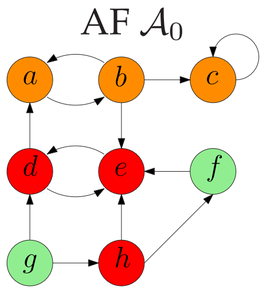
An *update* u for an AF $\mathcal{A}_0$ consists in modifying $\mathcal{A}_0$ into an AF $\mathcal{A}$ by adding or removing arguments or attacks.

- $+(a, b)$ (resp. $-(a, b)$) denotes the addition (resp. deletion) of an attack (a, b);
- $u(\mathcal{A}_0)$ means applying $u = \pm(a, b)$ to $\mathcal{A}_0$;
- **multiple (attacks) updates** can be simulated by a single attack update.

DYNAMIC ARGUMENTATION FRAMEWORKS

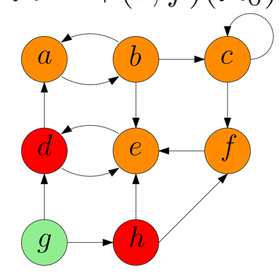
- An argumentation framework models a temporary situation as **new arguments and attacks can be added/removed** to take into account new available knowledge.
- For each semantics $\mathcal{S}$, the **sets of extensions change** if we update an initial AF $\mathcal{A}_0$ by adding/removing arguments/attacks. For instance, $\mathcal{E}_{\texttt{gr}}(\mathcal{A}_0) = \{\{f, g\}\}$ becomes $\mathcal{E}_{\texttt{gr}}(\mathcal{A})) = \{\{g\}\}$ for the updated AF $\mathcal{A} = +(c, f)(\mathcal{A}_0)$ obtained from $\mathcal{A}_0$ by adding attack (c, f).

AF $\mathcal{A}_0$



$\mathcal{S}$	$\mathcal{E}_{\mathcal{S}}(\mathcal{A}_0)$	$\mathcal{E}_{\mathcal{S}}(\mathcal{A}))$
co	$\{\{f, g\}, \{a, f, g\}, \{b, f, g\}\}$	$\{\{g\}, \{a, g\}, \{b, f, g\}\}$
pr	$\{\{a, f, g\}, \{b, f, g\}\}$	$\{\{a, g\}, \{b, f, g\}\}$
st	$\{\{b, f, g\}\}$	$\{\{b, f, g\}\}$
gr	**{f,g}**	**{g}**

$\mathcal{A} = +(c, f)(\mathcal{A}_0)$

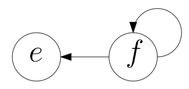


- **Should we recompute the semantics of updated AFs from scratch?**

CONTRIBUTIONS

- We show that an extension of the updated AF can be efficiently computed by looking only at a **small part of the AF**, called the *Reduced* AF, which is "influenced by" the update operation.

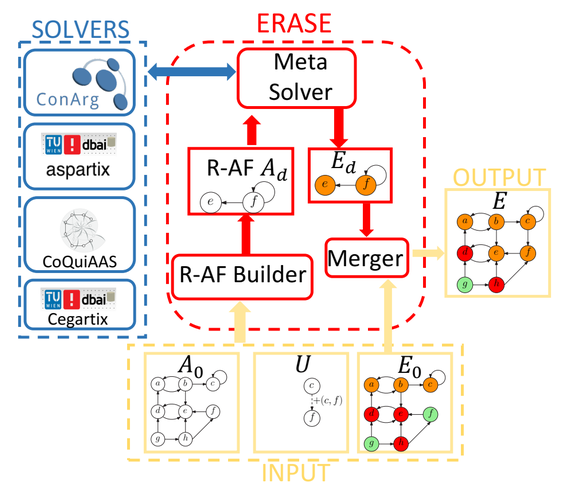
For the example above, the reduced AF is:



- We present an **incremental technique** for recomputing an extension of an updated AF for the *grounded*, *complete*, *preferred*, and *stable* semantics.

  It consists of the following three main steps:

  1) Identify a sub-AF $\mathcal{A}_d = \langle A_d, \Sigma_d \rangle$, called *reduced* AF (R-AF) on the basis of the updates in U and additional information provided by the initial extension E_0.
  2) Give R-AF $\mathcal{A}_d$ as input to an external (non-incremental) solver to compute an $\mathcal{S}$-extension E_d of the reduced AF.
  3) Merge E_d with the portion $(E_0 \setminus A_d)$ of the initial extension that does not change.



Architecture of ERASE, our system for Efficiently Recomputing Argumentation SEmantics.

- A thorough **experimental analysis** showing the effectiveness of our approach.

EXPERIMENTS

For each semantics $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{gr}\}$, we compared the performance of our technique with that of the solver that won the last ICCMA competition for the computational task $\mathcal{S}$-SE: Given an AF, determine some $\mathcal{S}$-extension.

Datasets: ICCMA'15 benchmarks.

Results: The figure reports the average run times (ms) of ICCMA solvers and our algorithm *(Incr-Alg)* for different semantics $\mathcal{S}$ over different datasets versus the number of attacks.

- Our algorithm significantly outperforms the competitors that compute the extensions from scratch for single updates. In fact, on average, our technique is **two orders of magnitude faster** than them. Moreover, the harder the computation from scratch is, the larger the improvements are: the improvements obtained for $\mathcal{S} \in \{\texttt{st}, \texttt{pr}\}$ go beyond those for $\mathcal{S} \in \{\texttt{gr}, \texttt{co}\}$.
- Our algorithm remains faster than the competitors even when recomputing an extension after performing a quite **large number of updates simultaneously**. In particular, in the graphs we show the threshold percentages of updated attacks (green lines) up to which the incremental approach for multiple updates is faster than the computation from scratch.
- For sets of updates regarding a relevant portion of the input AF (on average at least 1% of the attacks for $\mathcal{S} \in \{\texttt{st}, \texttt{pr}\}$ and 0,1% of the attacks for $\mathcal{S} \in \{\texttt{gr}, \texttt{co}\}$) recomputing extensions after applying them **simultaneously is faster than** recomputing extensions after applying them **sequentially**. Indeed, the green lines in the graphs are mostly below the (dashed) orange lines representing the run times of recomputing extensions after applying the updates sequentially.

The experiments also showed that, on average, the **size of the reduced AF** w.r.t. that of the input AF is about 9% for single updates and 52% for multiple updates with about 1% of the attacks updated.