Exploiting Parallelism for Hard Problems in Abstract Argumentation

Federico Cerutti
Dept. Computing Science
University of Aberdeen, UK

Ilias Tachmazidis and **Mauro Vallati** and **Sotirios Batsakis**
School of Computing and Engineering
University of Huddersfield, UK

Massimiliano Giacomin
Dept. of Information Engineering
University of Brescia, I

Grigoris Antoniou
School of Computing and Engineering
University of Huddersfield, UK

Abstract

Abstract argumentation framework (AF) is a unifying framework able to encompass a variety of nonmonotonic reasoning approaches, logic programming and computational argumentation. Yet, efficient approaches for most of the decision and enumeration problems associated to AFs are missing, thus limiting the efficacy of argumentation-based approaches in real domains. In this paper, we present an algorithm for enumerating the preferred extensions of abstract argumentation frameworks which exploits parallel computation. To this purpose, the SCC-recursive semantics definition schema is adopted, where extensions are defined at the level of specific sub-frameworks. The algorithm shows significant performance improvements in large frameworks, in terms of number of solutions found and speedup.

Introduction

Dung's theory of abstract argumentation (Dung 1995) is a unifying framework able to encompass a large variety of specific formalisms in the areas of nonmonotonic reasoning, logic programming and computational argumentation. It is based on the notion of argumentation framework (AF), consisting of a set of arguments and an *attack* relation between them. Different *argumentation semantics* introduce in a declarative way the criteria to determine which arguments emerge as 'justified' from the conflict, by identifying a number of *extensions*, i.e. sets of arguments that can "survive the conflict together". In (Dung 1995) four "traditional" semantics were introduced, namely *complete*, *grounded*, *stable*, and *preferred* semantics. For an introduction on alternative semantics see (Baroni, Caminada, and Giacomin 2011).

The main computational problems in abstract argumentation include *decision* and *construction* problems, and turn out to be computationally intractable for most of argumentation semantics (Dunne and Wooldridge 2009). In this paper we focus on the *extension enumeration* problem, i.e. constructing *all* extensions for a given AF: its solution provides complete information about the justification status of arguments and subsumes the solutions to the other problems.

In this paper we propose the first parallel approach for enumerating preferred extensions — a problem which lies at the second level of the polynomial hierarchy — which exploits the SCC-recursive schema (Baroni, Giacomin, and Guida 2005), a semantics definition schema where extensions are defined at the level of the sub-frameworks identified by the strongly connected components. A similar approach has been recently discussed in (Cerutti et al. 2014a): however, in this work we (1) show that preferred extensions can be derived from the solution of *independent* subproblems; (2) identify *independent* clusters of SCCs to consider when solving such sub-problems; (3) develop efficient algorithms and data structures for exploiting such independence using parallel and dynamic programming techniques.

As large-scale argumentation is vastly unexplored, there is no further work directly related to our approach. The closest work is in the context of Assumption-Based Argumentation (ABA) Frameworks (Bondarenko et al. 1997), an abstract framework for default reasoning which can be instantiated with different deductive systems (e.g. logic programming, autoepistemic logic, default logic). (Craven et al. 2012) describes a parallel implementation for credulous acceptance under the acceptablity semantics for some specific instances of ABAs in the medical domain. It considers competitive parallel executions: multiple versions — equivalent w.r.t. their outcome — of a sequential process are created and then started in parallel. Once one version finds a solution to the problem, the others are killed.

Our work can be seen as part of a broader recent push towards large-scale reasoning which, among others, concerns simple semantic web reasoning (Urbani et al. 2012), fuzzy ontologies (Liu et al. 2012) and logic programming (Tachmazidis, Antoniou, and Faber 2014). Indeed, the fast-growing field of *argument mining* from content in the Web (Grosse et al. 2012; Cabrio and Villata 2013) highlights the lack of large-scale reasoning approaches in formal argumentation, and thus increases the importance of our research.

The paper is organised as follows. In the first section we recall some necessary background on Dung's AF, the SCC-recursive schema and the existing algorithmic approach exploiting it. In the subsequent section we present our approach for exploiting the SCC-recursive schema in a parallel fashion, and we discuss the theoretical remarks granting the correctness of the approach. An exhaustive experimental analysis is then presented in the forthcoming section. The last section concludes the paper and discusses future work.

Background

Dung's Argumentation Framework

An argumentation framework (Dung 1995) consists of a set of arguments[1] and a binary attack relation between them.

Definition 1. *An* argumentation framework *(AF) is a pair* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ *where* $\mathcal{A}$ *is a set of arguments and* $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$. *We say that* $\boldsymbol{b}$ attacks $\boldsymbol{a}$ *iff* $\langle \boldsymbol{b}, \boldsymbol{a} \rangle \in \mathcal{R}$, *also denoted as* $\boldsymbol{b} \rightarrow \boldsymbol{a}$. *The set of attackers of an argument* $\boldsymbol{a}$ *will be denoted as* $\boldsymbol{a}^- \triangleq \{\boldsymbol{b} : \boldsymbol{b} \rightarrow \boldsymbol{a}\}$, *the set of arguments attacked by* $\boldsymbol{a}$ *will be denoted as* $\boldsymbol{a}^+ \triangleq \{\boldsymbol{b} : \boldsymbol{a} \rightarrow \boldsymbol{b}\}$. *We also extend these notations to sets of arguments, i.e. given* $E \subseteq \mathcal{A}$, $E^- \triangleq \{\boldsymbol{b} \mid \exists \boldsymbol{a} \in E, \boldsymbol{b} \rightarrow \boldsymbol{a}\}$ *and* $E^+ \triangleq \{\boldsymbol{b} \mid \exists \boldsymbol{a} \in E, \boldsymbol{a} \rightarrow \boldsymbol{b}\}$.

An argument $\mathbf{a}$ without attackers, i.e. such that $\mathbf{a}^- = \emptyset$, is said *initial*. Moreover, each AF has an associated directed graph where the vertices are the arguments, and the edges are the attacks.

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

Definition 2. *Given an* AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*:*

- *a set* $T \subseteq \mathcal{A}$ *is a* conflict–free *set of* Γ *if* $\nexists\, \boldsymbol{a}, \boldsymbol{b} \in T$ *s.t.* $\boldsymbol{a} \rightarrow \boldsymbol{b}$*;*
- *an argument* $\boldsymbol{a} \in \mathcal{A}$ *is* acceptable *with respect to a set* $T \subseteq \mathcal{A}$ *of* Γ *if* $\forall \boldsymbol{b} \in \mathcal{A}$ *s.t.* $\boldsymbol{b} \rightarrow \boldsymbol{a}$, $\exists\, \boldsymbol{c} \in T$ *s.t.* $\boldsymbol{c} \rightarrow \boldsymbol{b}$*;*
- *a set* $T \subseteq \mathcal{A}$ *is an* admissible set *of* Γ *if* T *is a conflict–free set of* Γ *and every element of* T *is acceptable with respect to* T *of* Γ.

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

Definition 3. *Given an* AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*:*

- *a set* $T \subseteq \mathcal{A}$ *is a* complete extension *of* Γ*, i.e.* $T \in \mathcal{E}_{\mathcal{CO}}(\Gamma)$*, iff* T *is admissible and* $\forall \boldsymbol{a} \in \mathcal{A}$ *s.t.* $\boldsymbol{a}$ *is acceptable w.r.t.* T*,* $\boldsymbol{a} \in T$*;*
- *a set* $T \subseteq \mathcal{A}$ *is the* grounded extension *of* Γ*, i.e.* $T \in \mathcal{E}_{\mathcal{GR}}(\Gamma)$*, iff* T *is the minimal (w.r.t.* $\subseteq$*) complete extension of* Γ*. Its existence and uniqueness have been proved in (Dung, Mancarella, and Toni 2006);*
- *a set* $T \subseteq \mathcal{A}$ *is a* preferred extension *of* Γ*, i.e.* $T \in \mathcal{E}_{\mathcal{PR}}(\Gamma)$*, iff* T *is a maximal (w.r.t.* $\subseteq$*) complete extension of* Γ.

Each extension T implicitly defines a three-valued *labelling* of arguments: an argument $\mathbf{a}$ is labelled `in` iff $\mathbf{a} \in T$; `out` iff $\exists\ \mathbf{b} \in T$ s.t. $\mathbf{b} \rightarrow \mathbf{a}$; `undec` otherwise. Argumentation semantics can be equivalently defined in terms of labellings rather than of extensions (Caminada 2006; Baroni, Caminada, and Giacomin 2011).

Definition 4. *Given a set of arguments* T*, a* labelling *of* T *is a total function* $\mathcal{L}ab : T \mapsto \{\texttt{in}, \texttt{out}, \texttt{undec}\}$*. The set of all* labellings *of* T *is denoted as* $\mathfrak{L}_T$*. Given an* AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*, a* labelling *of* Γ *is a labelling of* $\mathcal{A}$*. The set of all* labellings *of* Γ *is denoted as* $\mathfrak{L}(\Gamma)$.

[1] In this paper we consider only *finite* sets of arguments: see (Baroni et al. 2013) for a discussion on infinite sets of arguments.

Complete labellings can be defined as follows.

Definition 5. *Let* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ *be an* AF*. A labelling* $\mathcal{L}ab \in \mathfrak{L}(\Gamma)$ *is a* complete labelling *of* Γ *iff it satisfies the following conditions for any* $\boldsymbol{a} \in \mathcal{A}$*:*

- $\mathcal{L}ab(\boldsymbol{a}) = \texttt{in} \Leftrightarrow \forall \boldsymbol{b} \in \boldsymbol{a}^- \mathcal{L}ab(\boldsymbol{b}) = \texttt{out}$*;*
- $\mathcal{L}ab(\boldsymbol{a}) = \texttt{out} \Leftrightarrow \exists \boldsymbol{b} \in \boldsymbol{a}^- : \mathcal{L}ab(\boldsymbol{b}) = \texttt{in}$.

The grounded and preferred labelling can then be defined on the basis of complete labellings.

Definition 6. *Let* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ *be an* AF*. A labelling* $\mathcal{L}ab \in \mathfrak{L}(\Gamma)$ *is* the grounded labelling *of* Γ *if it is the complete labelling of* Γ *minimizing the set of arguments labelled* `in`*, and it is a* preferred labelling *of* Γ *if it is a complete labelling of* Γ *maximizing the set of arguments labelled* `in`.

The function `Ext2Lab` provides the connection between extensions and labellings.

Definition 7. *Given an* AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ *and a conflict–free set* $T \subseteq \mathcal{A}$*, the corresponding labelling* $\texttt{Ext2Lab}(T)$ *is defined as* $\texttt{Ext2Lab}(T) \equiv \mathcal{L}ab$*, where*

- $\mathcal{L}ab(\boldsymbol{a}) = \texttt{in} \Leftrightarrow \boldsymbol{a} \in T$
- $\mathcal{L}ab(\boldsymbol{a}) = \texttt{out} \Leftrightarrow \exists\, \boldsymbol{b} \in T$ *s.t.* $\boldsymbol{b} \rightarrow \boldsymbol{a}$
- $\mathcal{L}ab(\boldsymbol{a}) = \texttt{undec} \Leftrightarrow \boldsymbol{a} \notin T \wedge \nexists\, \boldsymbol{b} \in T$ *s.t.* $\boldsymbol{b} \rightarrow \boldsymbol{a}$

(Caminada 2006) shows that there is a bijective correspondence between extensions and labellings for complete, grounded, and preferred semantics.

Proposition 1. *Given an* AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*,* $\mathcal{L}ab$ *is a complete (grounded, preferred) labelling of* Γ *if and only if there is a complete (grounded, preferred) extension* T *of* Γ *such that* $\mathcal{L}ab = \texttt{Ext2Lab}(T)$.

The set of complete labellings of Γ is denoted as $\mathcal{L}_{\mathcal{CO}}(\Gamma)$, the set of preferred labellings as $\mathcal{L}_{\mathcal{PR}}(\Gamma)$, while $\mathcal{L}_{\mathcal{GR}}(\Gamma)$ denotes the set including the grounded labelling.

SCC-Recursiveness

In (Baroni and Giacomin 2004) an extension-based semantics definition schema has been introduced, called *SCC (strongly connected component)-recursiveness*, based on the graph-theoretical notion of SCCs (Tarjan 1972, Lemma 9) and on the observation that most argumentation semantics can be equivalently defined at the level of SCCs.

The following definitions introduce the SCC-recursive schema (Baroni, Giacomin, and Guida 2005). First, let us recall the definition of *restriction* of an AF Γ to a set of arguments I, in symbol $\Gamma{\downarrow_I}$.

Definition 8. *Given an* AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ *and a set* $I \subseteq \mathcal{A}$*, the restriction of* Γ *to* I *is defined as* $\Gamma{\downarrow_I} \equiv (I, \mathcal{R} \cap (I \times I))$.

Then, Definition 9 introduces the function $\mathcal{GF}(\Gamma, C)$ which recursively computes the semantics extensions on the basis of the SCCs of Γ. Let us denote as SCC_Γ the set including the SCCs of an AF Γ.

Definition 9. *A given argumentation semantics* σ *is SCC-recursive if for any* AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*,* $\mathcal{E}_\sigma(\Gamma) = \mathcal{GF}(\Gamma, \mathcal{A}) \subseteq 2^{\mathcal{A}}$*. For any* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ *and for any set* $C \subseteq \mathcal{A}$*,*
$E \in \mathcal{GF}(\Gamma, C)$ *if and only if*

- $E \in \mathcal{BF}_\sigma(\Gamma, C)$ *if* $|SCC_\Gamma| = 1$

- $\forall S \in SCC_\Gamma\ (E \cap S) \in \mathcal{GF}(\Gamma{\downarrow_{S\setminus(E\setminus S)^+}}, U_\Gamma(S,E) \cap C)$ *otherwise*

where

- $\mathcal{BF}_\sigma(\Gamma, C)$ *is a function, called* base function*, that, given an* $AF\ \Gamma = \langle \mathcal{A}, \mathcal{R}\rangle$ *such that* $|SCC_\Gamma| = 1$ *and a set* $C \subseteq \mathcal{A}$*, gives a subset of* $2^{\mathcal{A}}$
- $U_\Gamma(S,E) = \{\boldsymbol{a} \in S\setminus(E\setminus S)^+ \mid \forall \boldsymbol{b} \in (\boldsymbol{a}^- \setminus S),\ \boldsymbol{b} \in E^+\}$

The schema is based on the notions of extension of an AF *in a set of arguments*.

Definition 10. *Given an* $AF\ \Gamma = \langle \mathcal{A}, \mathcal{R}\rangle$ *and a set* $C \subseteq \mathcal{A}$*, a set* $E \subseteq \mathcal{A}$ *is: an* admissible set of Γ *in* C *if and only if* E *is an admissible set of* Γ *and* $E \subseteq C$*; a* complete extension of Γ in C *if and only if* E *is an admissible set of* Γ *in* C*, and every argument* $\alpha \in C$ *which is acceptable with respect to* E *belongs to* E*; the* grounded extension of Γ in C *if and only if it is the least (w.r.t.* $\subseteq$*) complete extension of* Γ *in* C*; a* preferred extension of Γ in C *if and only if it is a maximal (w.r.t.* $\subseteq$*) complete extension of* Γ *in* C*.*

The existence and uniqueness of the grounded extension in C, as well as the existence of at least a preferred extension in C, have been proved in (Baroni, Giacomin, and Guida 2005). Moreover, (Baroni, Giacomin, and Guida 2005) proves that $\mathcal{GF}(\Gamma, C)$, as defined in Def. 9, returns the σ-extensions in C (with $\sigma \in \{\mathcal{CO}, \mathcal{GR}, \mathcal{PR}\}$), provided that $\mathcal{BF}_\sigma(\Gamma, C)$ returns the complete, grounded, and preferred extensions in C, respectively.

(Cerutti et al. 2014a) introduces the notions of complete, grounded and preferred labellings of Γ in C, i.e. the labelling-based counterparts of the corresponding notions of Definition 10, and describes a preliminary algorithm — R-PREF — exploiting the SCC-recursive schema. R-PREF implements $\mathcal{GF}$ (Def. 9), where the chosen base function $\mathcal{BF}_{\mathcal{PR}}$ is computed by a refinement of the algorithm in (Cerutti et al. 2013) which exploits a SAT solver as a NP-oracle to determine the preferred labellings. R-PREF exploits the SCC-recursive schema by constructing a sequence of strongly connected components of Γ in a topological order. Preferred labellings are incrementally constructed along the SCCs, by computing the preferred labellings of each SCC and merging them with those identified in the previous SCCs. In the following, we take advantage of two algorithms mentioned in (Cerutti et al. 2014a), namely GROUNDED (Cerutti et al. 2014a, Alg. 3) and B-PR (Cerutti et al. 2014a, Alg. 4): their usage is described in the following section.

Exploiting Parallel Computation

In this section we present our approach exploiting parallel computation in the context of the SCC-recursive schema. First of all, we need to identify when it is possible to parallelise the process aimed at verifying that given $\Gamma = \langle \mathcal{A}, \mathcal{R}\rangle$, $\forall E \subseteq \mathcal{A}, \forall C \subseteq \mathcal{A}, E \in \mathcal{GF}(\Gamma, C)$.

Theoretical Remarks

Two elements guaranteeing independence and thus the possibility to parallelise the process can be identified. First of all, each preferred extension can be computed independently from the others.

Remark 1. *Given an* $\Gamma = \langle \mathcal{A}, \mathcal{R}\rangle$*,* $\forall E \in \mathcal{E}_\sigma(\Gamma), \forall C \subseteq \mathcal{A}$*, proving that* $E \in \mathcal{GF}(\Gamma, C)$ *does not require any knowledge about* $\overline{E} \in \mathcal{E}_\sigma(\Gamma) \setminus \{E\}$*.*

A second, rather more articulated, condition of independence requires to identify two sets of SCCs, $\overline{S} = \{S_1, \ldots, S_n\} \subseteq \text{SCC}_\Gamma$ and $P_{\overline{S}} = \{P_1, \ldots, P_m\} \subseteq \text{SCC}_\Gamma$ such that (1) each SCC in $\overline{S}$ does not attack the others in $\overline{S}$; and (2) each SCC in $\overline{S}$ is attacked only by SCCs in $P_{\overline{S}}$.

Remark 2. *Given an* $\Gamma = \langle \mathcal{A}, \mathcal{R}\rangle$*,* $\forall E \subseteq \mathcal{A}, \forall C \subseteq \mathcal{A}$*, if there exist* $\overline{S} = \{S_1, \ldots, S_n\} \subseteq SCC_\Gamma$ *such that* $\forall S_i, S_j, S_i^+ \cap S_j = \emptyset$*, and there exists* $P_{\overline{S}} \subseteq SCC_\Gamma$ *such that* $\forall S_i \in \overline{S}, (S_i{}^- \setminus S_i) \subseteq \bigcup_{P \in P_{\overline{S}}} P$*, then* $\forall S \in \overline{S}$ *proving that* $(E \cap S) \in \mathcal{GF}(\Gamma{\downarrow_{S\setminus(E\setminus S)^+}}, U_\Gamma(S,E) \cap C)$ *can be determined in function of* $P_{\overline{S}}$ *and does not require any knowledge about* $S' \in \overline{S} \setminus \{S\}$*.*

The P-SCC-REC Algorithm

In this section we introduce a meta-algorithm — P-SCC-REC (Alg. 2) — which exploits the SCC-recursive schema using parallel computation and a pro-active *greedy* approach which *memoizes* some notable cases.

First of all, the function P-PREF (Algorithm 1) receives as input an $AF\ \Gamma = \langle \mathcal{A}, \mathcal{R}\rangle$ and returns the set of preferred labellings of Γ. This is simply achieved by invoking (at line 3) P-SCC-REC$(\Gamma, \mathcal{A})$, where the function P-SCC-REC ($\mathcal{GF}$ in Def. 9) receives as input an $AF\ \Gamma = \langle \mathcal{A}, \mathcal{R}\rangle$ and a set $C \subseteq \mathcal{A}$, and computes the set $\mathcal{L}_{\mathcal{PR}}(\Gamma, C)$, i.e. the set of preferred labellings of Γ in C.

P-SCC-REC first pre-processes (at line 3) — via the function GROUNDED (Cerutti et al. 2014a, Alg. 3) — Γ by computing the grounded labelling in C: $\mathcal{L}ab$ contains the restriction of the grounded labelling to those arguments which are either `in` or `out`; U is the set of arguments that are labelled `undec` in the grounded labelling.

At line 4 P-SCC-REC initialises to $\{\mathcal{L}ab\}$ the variable E_p, which stores the set of labellings that are incrementally constructed. At line 5 P-SCC-REC restricts Γ to $\Gamma{\downarrow_U}$. Then, at line 6, P-SCC-REC exploits Remark 2 by building a list $L := (L^1, \ldots, L^n)$ of sets of SCCs — (Cormen et al. 2009, p. 617) with some modifications — such that $\forall L^i \in L, L^i = \{S_j^i \in \text{SCC}_\Gamma \mid (S_j^i)^- \setminus S_j^i \in \bigcup_{z\in\{1,\ldots,i-1\}} \bigcup_{S\in L^z} S$ and $(S_j^i)^+ \setminus S_j^i \in \bigcup_{z\in\{i+1,\ldots,n\}} \bigcup_{S\in L^z} S\}$.

At line 7, the GREEDY function (Alg. 3) is called; it receives as input the list of SCCs L and the set of arguments C, and returns a set M of pairs (S_i, B_i) where $S_i \in \text{SCC}_\Gamma$, and $B_i = \mathcal{L}_{\mathcal{PR}}(\Gamma{\downarrow_{S_i}}, S_i \cap C)$. B_i — computed by the function B-PR (Cerutti et al. 2014a, Alg. 4) — is the set of preferred labellings for S_i when no argument in S_i is attacked by `in` or `undec` arguments in previous, w.r.t. the L list, SCCs.

Then, at lines $8-36$, P-SCC-REC performs a first loop among the elements of the list L. At line 9 a rather articulated data structure, E_l, is initialised. For each $S \in L_l$, E_l^S is a list of pairs $(\mathcal{L}ab, E_l^S[\mathcal{L}ab] \subseteq \mathfrak{L}_S)$: to ease of notation, hereafter we omit the pair-structure thus referring directly to $E_l^S[\mathcal{L}ab]$ which contains the set of preferred labellings of S

Algorithm 1 Computing preferred labellings of an AF

P-PREF(Γ)

1: **Input:** $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$
2: **Output:** $E_p \in 2^{\mathfrak{L}(\Gamma)}$
3: **return P-SCC-REC**$(\Gamma, \mathcal{A})$

Algorithm 2 Computing preferred labellings of an AF in C

P-SCC-REC(Γ, C)

1: **Input:** $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle, C \subseteq \mathcal{A}$
2: **Output:** $E_p \in 2^{\mathfrak{L}(\Gamma)}$
3: $(\mathcal{L}ab, U) =$ GROUNDED(Γ, C)
4: $E_p := \{\mathcal{L}ab\}$
5: $\Gamma = \Gamma{\downarrow_U}$
6: $L := (L^1 := \{S_1^1, \ldots, S_k^1\}, \ldots, L^n := \{S_1^n, \ldots, S_h^n\})$ $= $ SCCS-LIST(Γ)
7: $M := \{\ldots, (S_i, B_i), \ldots\} =$ GREEDY(L, C)
8: **for** $l \in \{1, \ldots, n\}$ **do**
9: $\quad E_l := \{E_l^{S_1} := (), \ldots, E_l^{S_k} := ()\}$
10: $\quad$ **for** $S \in L^l$ **do in parallel**
11: $\quad\quad$ **for** $\mathcal{L}ab \in E_p$ **do in parallel**
12: $\quad\quad\quad (O, I) :=$ L-COND$(\Gamma, S, L^l, \mathcal{L}ab)$
13: $\quad\quad\quad$ **if** $I = \emptyset$ **then**
14: $\quad\quad\quad\quad E_l^S[\mathcal{L}ab] = \{\{(\mathbf{a}, \texttt{out}) \mid \mathbf{a} \in O\} \cup \{(\mathbf{a}, \texttt{undec}) \mid \mathbf{a} \in S \setminus O\}\}$
15: $\quad\quad\quad$ **else**
16: $\quad\quad\quad\quad$ **if** $I = S$ **then**
17: $\quad\quad\quad\quad\quad E_l^S[\mathcal{L}ab] = B$ where $(S, B) \in M$
18: $\quad\quad\quad\quad$ **else**
19: $\quad\quad\quad\quad\quad$ **if** $O = \emptyset$ **then**
20: $\quad\quad\quad\quad\quad\quad E_l^S[\mathcal{L}ab] =$ B-PR$(\Gamma{\downarrow_S}, I \cap C)$
21: $\quad\quad\quad\quad\quad$ **else**
22: $\quad\quad\quad\quad\quad\quad E_l^S[\mathcal{L}ab] = \{\{(\mathbf{a}, \texttt{out}) \mid \mathbf{a} \in O\}\}$
23: $\quad\quad\quad\quad\quad\quad E_l^S[\mathcal{L}ab] = E_l^S[\mathcal{L}ab] \otimes$ P-SCC-REC$(\Gamma{\downarrow_{S \setminus O}}, I \cap C)$
24: $\quad\quad\quad\quad\quad$ **end if**
25: $\quad\quad\quad\quad$ **end if**
26: $\quad\quad\quad$ **end if**
27: $\quad\quad$ **end for**
28: $\quad$ **end for**
29: $\quad$ **for** $S \in L^l$ **do**
30: $\quad\quad E'_p := \emptyset$
31: $\quad\quad$ **for** $\mathcal{L}ab \in E_p$ **do in parallel**
32: $\quad\quad\quad E'_p = E'_p \cup (\{\mathcal{L}ab\} \otimes E_l^S[\mathcal{L}ab])$
33: $\quad\quad$ **end for**
34: $\quad\quad E_p := E'_p$
35: $\quad$ **end for**
36: **end for**
37: **return** E_p

Algorithm 3 Greedy computation of base cases

GREEDY(L, C)

1: **Input:** $L = (L^1, \ldots, L^n := \{S_1^n, \ldots, S_h^n\}), C \subseteq \mathcal{A}$
2: **Output:** $M = \{\ldots, (S_i, B_i), \ldots\}$
3: $M := \emptyset$
4: **for** $S \in \bigcup_{i=1}^n L^i$ **do in parallel**
5: $\quad B :=$ B-PR$(\Gamma{\downarrow_S}, S \cap C)$
6: $\quad M = M \cup \{(S, B)\}$
7: **end for**
8: **return** M

constructed on the basis of a specific labelling $\mathcal{L}ab$ identified in the previous (w.r.t. the list L) SCCs.

Two more loops are thus considered and their execution can be safely parallelised: the loop at lines $10-28$ exploits Remark 2 by considering each SCC in a given element of the list L; while the loop at lines $11-27$ considers a single preferred labelling, each of which is independent from the others — cf. Remark 1.

L-COND$(\Gamma, S, L^l, \mathcal{L}ab)$ at line 12 computes the effect of previous SCCs, and returns (O, I), where:

- $O = \{\mathbf{a} \in S \mid \exists \mathbf{b} \in T \cap \mathbf{a}^- : \mathcal{L}ab(\mathbf{b}) = \texttt{in}\}$ and
- $I = \{\mathbf{a} \in S \mid \forall\, \mathbf{b} \in T \cap \mathbf{a}^-, \mathcal{L}ab(\mathbf{b}) = \texttt{out}\}$,

with $T \equiv \bigcup_{i=1}^{l-1} \bigcup_{S \in L_i} S$. Variable O is set to include arguments of S that are attacked by "outside" `in`-labelled arguments according to $\mathcal{L}ab$, and variable I is set to include arguments of S that are only attacked by "outside" `out`-labelled arguments. This gives rise to three cases:

1. each argument of S is attacked by `in` or `undec` arguments in previous SCCs — hence each argument of S is labelled `out` or `undec` (line 14);
2. no argument of S is attacked by `in` arguments in previous SCCs: this is the base case of the recursion and thus either we exploit the *memoization* technique implemented with the GREEDY algorithm (line 17) or we exploit the function B-PR (line 20);
3. in the remaining case, arguments attacked by `in`-arguments are labelled as `out` and P-SCC-REC is recursively called on the restriction of S to the unlabelled arguments (lines $22-23$).

Finally, at lines $29-35$ the computed preferred labellings $E_l^S[\mathcal{L}ab]$ are merged together ($E_1 \otimes E_2 = \{\mathcal{L}ab_1 \cup \mathcal{L}ab_2 | \mathcal{L}ab_1 \in E_1, \mathcal{L}ab_2 \in E_2\}$) with the $\mathcal{L}ab$ labelling of previous SCCs. Once again, due to Remark 1, this process can be parallelised (lines $31-33$).

Then the algorithm considers the next element in the list L. Once the outer loop is exited, all strongly connected components have been processed, thus E_p is returned as the set of preferred labellings in C (line 37). Proof of the correctness of Alg. 2 is provided in (Cerutti et al. 2014b).

Empirical Analysis

The solvers have been run on a cluster with computing nodes equipped with 2.4 Ghz Dual Core AMD Opteron™, 8 GB of RAM and Linux operating system. As in the International Planning Competition (IPC) (Jiménez et al. 2012), a cutoff of 900 seconds was imposed to compute the preferred extensions for each AF. No limit was imposed on

	P1	P2	P2G	P4	P4G
IPC score	50.0	58.6	44.7	**74.9**	56.9
% success	58.8	68.3	57.3	**74.9**	65.8
% best	1.0	0.0	0.0	**74.4**	0.0
Avg runtime	439.4	464.3	385.7	**269.6**	384.8
Speedup	–	1.9	1.3	2.8	1.7

Table 1: Performance achieved by using 1, 2 and 4 processors with/out exploiting the *greedy* approach (+G). Results are shown in terms of normalised IPC score, percentages of success, percentages of AFs in which the system has been the fastest, average runtime (considering AFs in which at least one approach succeeded) and max speedup against P1. Values in bold indicate the best results.

the RAM usage, but a run fails at saturation of the available memory. Moreover, we adopted the IPC speed score, also borrowed from the planning community, which is defined as follows. For each AF, each system gets a score of $1/(1+\log_{10}(T/T^*))$, where T is its execution time and T^* the best execution time among the compared systems, or a score of 0 if it fails in that case. Runtimes below 0.01 sec get by default the maximal score of 1. In our experimental analysis, IPC score is normalised to 100. For each solver we recorded the overall result: success (if it finds each preferred extension), crashed, timed-out or ran out of memory.

As shown in (Cerutti, Giacomin, and Vallati 2014a), most of the state-of-the-art approaches for enumerating preferred extensions hardly solve large (w.r.t. the number of arguments) frameworks. In this work, we focus on extremely large AFs; the largest – as far as we know – that have ever been used for testing solvers.

We randomly generated a set of 200 AFs, varying the number of SCCs between 90 — 80 SCCs is the upper-bound for experiments in (Cerutti et al. 2014a) — and 210, the number of arguments between 2,700 and 8,400, and considering different uniformly distributed probabilities of attacks, either between arguments or between different SCCs, leading to AFs with a number of attacks between approximately 100 thousands and 2 millions. AFs were generated using `AFBenchGen` (Cerutti, Giacomin, and Vallati 2014b).

Table 1 shows the results of the overall comparison between R-PREF (henceforth P1) and P-SCC-REC (henceforth P2 or P4, according to the number of processors). The latter exploits either two or four processors, and has been run with and without the *greedy* approach. From Table 1, two main conclusions can be derived. First, the exploitation of *greedy* approach introduces a significant overhead, due to the required pre-calculation. In our testing instances, pre-calculated knowledge is not used by algorithms and therefore, exploiting a *greedy* approach has a detrimental effect on P2 and P4 performance. This behaviour is confirmed also by a comparison (not shown) between P1 with/out *greedy* approach. Given this result, the *greedy* approach will not be considered in the rest of this section.

The second conclusion we derive from Table 1 is that parallelisation improves significantly the performance of both runtime and the number of successfully analysed AFs.

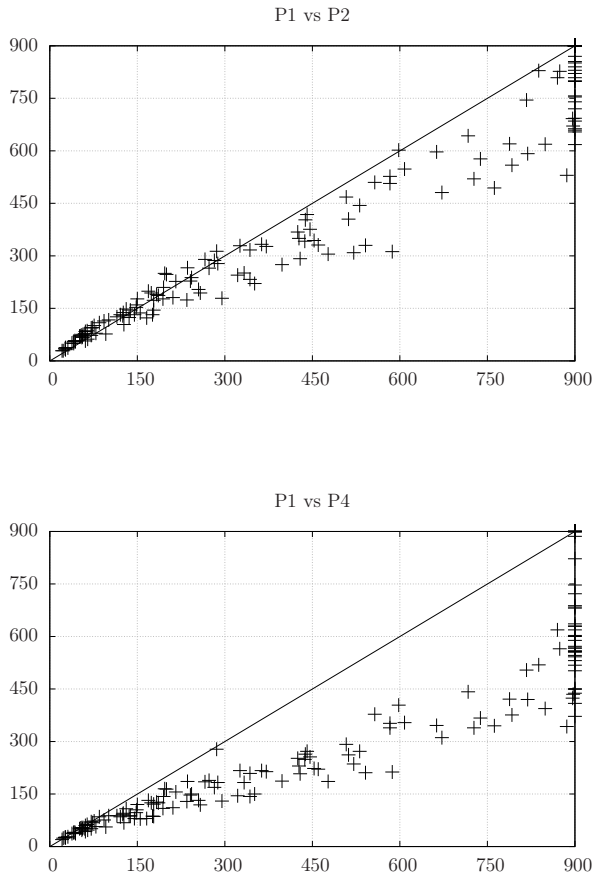

Figure 1: CPU-time of P2 (upper) or P4 (lower) w.r.t. P1 for all the considered AFs. The x-axis refers to CPU seconds of P1; the y-axis refers to CPU seconds of P2 (upper) or P4 (lower). CPU-time of 900 seconds indicates timeout.

Since the normalised IPC score of P4 is equal to its percentage of successes, P4 is always the fastest approach on the whole testing set. Both P4 and P2 — according to the Wilcoxon Signed-Rank Test (WSRT) (Wilcoxon 1945) — perform significantly better than P1 ($p < 0.05$). Using 2 (resp. 4) processors provides a maximum speed-up of 1.9 (resp. 2.8) times w.r.t. serial execution. Such results justify the use of parallel approaches in abstract argumentation.

Figure 1 provides results in the form of scatterplots, showing the performance of P1 and, respectively, P2 and P4. Using 2 processors has a remarkable impact on runtimes, in particular on complex AFs, which require approximately more than 300 seconds. A larger number of AFs can be successfully analysed by P2: this can be derived by observing the elements on the right axis of the graph. On *simple* AFs, the impact of using 2 processors is not so clear. On the other hand, parallelising on 4 processors guarantee to obtain lower runtimes on the whole testing set. This behaviour is probably due to the fact that the overhead introduced by parallelisation (generating threads, communication overhead, etc.)

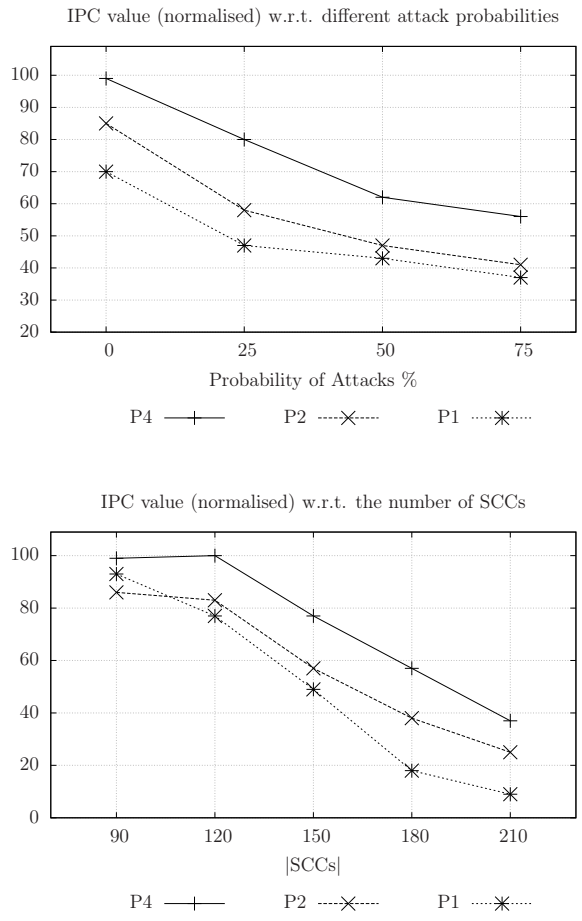

Figure 2: IPC scores of P2, P4 and P1 w.r.t. the probability of attacks between different SCCs (upper) and the number of SCCs (lower).

is not completely compensated by using 2 processors only, specially when a short amount of CPU time is needed for enumerating the extensions of a given AF.

The number of SCCs in the same set, cf. Remark 2, critically affects the performance of the proposed parallel algorithm. The larger the size of each *level* — i.e. each element of list L, Alg. 2, line 6 — the higher the degree of parallelisation that can be reached, since parallelisation is primarily based on processing simultaneously SCCs that are located on the same level. Figure 2 (upper) show the IPC score of parallelised and serial algorithms, with regards to the probability of attacks between SCCs. As expected, the performance gap between parallelised (P2, P4) and serial (P1) algorithms is maximum when the probability is 0 — i.e., all the SCCs are on the same level — and slowly decreases as the percentage increases. With a probability of 75%, most of the levels have a single SCC, therefore parallelisation does not provide a great speedup. It is worthy to notice that at higher attacks probability percentages, enumerating all the preferred extensions is very complex, and requires a significant amount of CPU-time. The differences of performance between P1 and P4 are always statistically significant (WSRT $p < 0.05$). It is not the case of P1 and P2, their performance are statistically indistinguishable when the probability of attacks is 50% ($p = 0.39$) and 75% ($p = 0.66$).

Finally, Figure 2 shows how IPC score of considered algorithms changes with regard to the number of SCCs of the AFs. As a general trend, increasing the number of SCCs increases the runtime (and decreases the number of successes) for all implementations. This is expected, as larger inputs are harder to solve. On the other hand, P1 is very quick on smallest considered AFs; on average it is faster than P2. P1 performance rapidly decreases as the number of SCCs increases. This is also confirmed by the WSRT: while P4 is always statistically better than P1, P2 performs statistically worse than P1 on AFs with $|\text{SCCs}| = 90$, but it performs statistically better when $|\text{SCCs}| \geq 120$. Generally, parallelisation provides best speedup on very large AFs, with lower probability of attacks among SCCs.

Conclusions

In this paper we propose an approach for exploiting the SCC-recursive schema for computing semantics extensions in Dung's AFs taking advantage of parallel executions and dynamic programming. It is worth mentioning that Alg. 1, in conjunction with Algs. 2 and 3, are meta-algorithms that implement the SCC-recursive schema independently from the chosen semantics. Although we chose to consider the preferred semantics in order to provide a direct comparison with recent works (Cerutti et al. 2014a) — in essence equivalent to P1 — , the same algorithms can work for all the SCC-recursive semantics (Baroni, Giacomin, and Guida 2005).

Moreover, the empirical analysis shows that there is a substantial statistically significant increment of performance due to the partial parallel execution of the proposed algorithms. This results in:

1. an increment (approx. 50%) of the number of AFs for which we can solve the preferred semantics enumeration problem before the chosen cutoff time;
2. a significant speedup of the computation of preferred extension up to 280% just considering 4 processors.

Future work is already envisaged in the area of additional experimentation analyses by considering different benchmarks, including (Correia, Cruz, and Leite 2014), in particular for assessing the relationship between number of cores and performance, and for investigating the worst-cases scenarios and the effects of graph shapes.

We also plan to apply more dynamic programming techniques (e.g. *memoization*) by improving the current proposal of the *greedy* computation of some preferred labelling — Alg. 3. In addition, we will compare our approach with (Dvořák et al. 2011; Liao, Lei, and Dai 2013; Ellmauthaler and Strass 2014; Nofal, Atkinson, and Dunne 2014), which adopt different techniques for enumerating preferred extensions, like reducing such a problem to an ASP program. Finally, recent works on Input/Output behaviour characterisation of AFs (Baroni et al. 2012; 2014) can be exploited for determining conditions of independent computation and thus exploiting parallel executions on structures different from SCCs.

References

Baroni, P., and Giacomin, M. 2004. A General Recursive Schema for Argumentation Semantics. In *Proc. of the 14th European Conf. on Artif. Intell. (ECAI 2004)*, 783–787.

Baroni, P.; Boella, G.; Cerutti, F.; Giacomin, M.; van der Torre, L.; and Villata, S. 2012. On Input/Output Argumentation Frameworks. In *Proc. of the 4th Int. Conf. on Computational Models of Arguments (COMMA 2012)*, 358–365.

Baroni, P.; Cerutti, F.; Dunne, P. E.; and Giacomin, M. 2013. Automata for Infinite Argumentation Structures. *Artif. Intell.* 203(0):104–150.

Baroni, P.; Boella, G.; Cerutti, F.; Giacomin, M.; van der Torre, L.; and Villata, S. 2014. On the input/output behavior of argumentation frameworks. *Artif. Intell.* 217(0):144–197.

Baroni, P.; Caminada, M.; and Giacomin, M. 2011. An introduction to argumentation semantics. *Knowl. Eng. Rev.* 26(4):365–410.

Baroni, P.; Giacomin, M.; and Guida, G. 2005. SCC-recursiveness: a general schema for argumentation semantics. *Artif. Intell.* 168(1-2):165–210.

Bondarenko, A.; Dung, P.; Kowalski, R.; and Toni, F. 1997. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.* 93(1–2):63–101.

Cabrio, E., and Villata, S. 2013. A natural language bipolar argumentation approach to support users in online debate interactions. *Argument & Computation* 4(3):209–230.

Caminada, M. 2006. On the Issue of Reinstatement in Argumentation. In *Proc. of the 10th European Conf. on Logics in Artif. Intell. (JELIA 2006)*, 111–123.

Cerutti, F.; Dunne, P. E.; Giacomin, M.; and Vallati, M. 2013. Computing Preferred Extensions in Abstract Argumentation: A SAT-Based Approach. In *Proc. of Theory and Applications of Formal Argumentation (TAFA 2013)*, 176–193.

Cerutti, F.; Giacomin, M.; Vallati, M.; and Zanella, M. 2014a. A SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation. In *Proc. of the 14th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2014)*, 42–51.

Cerutti, F.; Tachmazidis, I.; Vallati, M.; Batsakis, S.; Giacomin, M.; and Antoniou, G. 2014b. Exploiting Parallelism for Hard Problems in Abstract Argumentation: Technical Report. `http://arxiv.org/abs/1411.2800`.

Cerutti, F.; Giacomin, M.; and Vallati, M. 2014a. Algorithm selection for preferred extensions enumeration. In *Proc. of the 5th Int. Conf. on Computational Models of Argument (COMMA 2014)*, 221–232.

Cerutti, F.; Giacomin, M.; and Vallati, M. 2014b. Generating challenging benchmark AFs. In *Proc. of the 5th Int. Conf. on Computational Models of Argument (COMMA 2014)*, 457–458.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms*. MIT Press.

Correia, M.; Cruz, J.; and Leite, J. a. 2014. On the Efficient Implementation of Social Abstract Argumentation. In *Proc. of the 21st European Conf. on Artif. Intell. (ECAI 2014)*, 225–230.

Craven, R.; Toni, F.; Cadar, C.; Hadad, A.; and Williams, M. 2012. Efficient argumentation for medical decision-making. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2012)*, 598–602.

Dung, P. M.; Mancarella, P.; and Toni, F. 2006. A dialectic procedure for sceptical, assumption-based argumentation. In *Prooceedings of the 1st Int. Conf. on Computational Models of Arguments (COMMA 2006)*, 145–156.

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.

Dunne, P. E., and Wooldridge, M. 2009. Complexity of abstract argumentation. In Rahwan, I., and Simari, G., eds., *Argumentation in AI*. Springer-Verlag. chapter 5, 85–104.

Dvořák, W.; Gaggl, S. A.; Wallner, J.; and Woltran, S. 2011. Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems. In *Proc. of the 19th Int. Conf. on Applications of Declarative Programming and Knowledge Management (INAP 2011)*.

Ellmauthaler, S., and Strass, H. 2014. The DIAMOND System for Computing with Abstract Dialectical Frameworks. In *Proc. of the 5th Int. Conf. on Computational Models of Argument (COMMA 2014)*, 233–240.

Grosse, K.; Chesñevar, C. I.; Maguitman, A. G.; and Estevez, E. 2012. Empowering an e-government platform through twitter-based arguments. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* 15(50):46–56.

Jiménez, S.; de la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *Knowl. Eng. Rev.* 27(4):433–467.

Liao, B.; Lei, L.; and Dai, J. 2013. Computing Preferred Labellings by Exploiting SCCs and Most Sceptically Rejected Arguments. In *Second Int. Workshop on Theory and Applications of Formal Argumentation (TAFA-13)*.

Liu, C.; Qi, G.; Wang, H.; and Yu, Y. 2012. Reasoning with Large Scale Ontologies in Fuzzy pD* Using MapReduce. *IEEE CIM* 7(2):54–66.

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

Tachmazidis, I.; Antoniou, G.; and Faber, W. 2014. Efficient Computation of the Well-Founded Semantics over Big Data. *TPLP* 14(4-5):445–459.

Tarjan, R. E. 1972. Depth-first search and linear graph algorithms. *SIAM J. Comput.* 1(2):146–160.

Urbani, J.; Kotoulas, S.; Maassen, J.; Van Harmelen, F.; and Bal, H. 2012. WebPIE: A Web-scale Parallel Inference Engine using MapReduce. *J. Web Sem.* 10:59–75.

Wilcoxon, F. 1945. Individual comparisons by ranking methods. *Biometrics Bulletin* 1(6):80–83.