Exploiting Parallelism for Hard Problems in Abstract Argumentation

Federico Cerutti and Ilias Tachnazidis and Mauro Vallati and Sotirios Batsakis
Dept. of Computing Science
University of Aberdeen, 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 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 sub-problems; (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 acceptability 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 (Tachnazidis, Antoniou, and Faber 2014). Indeed, the fast-growing field of argument mining from content in the Web (Grose 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 and a binary attack relation between them.

Definition 1. An argumentation framework (AF) is a pair $\Gamma = \langle A, R \rangle$ where $A$ is a set of arguments and $R \subseteq A \times A$.

We say that $b$ attacks $a$ iff $(b, a) \in R$, also denoted as $b \rightarrow a$.

The set of attackers of an argument $a$ will be denoted as $a^{-} \triangleq \{ b : b \rightarrow a \}$, the set of arguments attacked by $a$ will be denoted as $a^{+} \triangleq \{ b : a \rightarrow b \}$. We also extend these notions to sets of arguments, i.e., given $E \subseteq A$, $E^{-} \triangleq \{ b \mid \exists a \in E, b \rightarrow a \}$ and $E^{+} \triangleq \{ b \mid \exists a \in E, a \rightarrow b \}$.

An argument $a$ without attackers, i.e., such that $a^{-} = \emptyset$, 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 all sets of arguments are fundamental for the definition of argumentation semantics.

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

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

An argumentation semantics $\sigma$ preserves for any AF $\Gamma$ a set of extensions, denoted as $\mathcal{E}_\sigma(\Gamma)$, namely a set of sets of arguments satisfying the conditions dictated by $\sigma$. 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 $\Gamma = \langle A, R \rangle$:

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

Each extension $T$ implicitly defines a three-valued labelling of arguments: an argument $a$ is labelled in iff $a \in T$; out iff $\exists b \in T \text{ s.t. } b \rightarrow a$; 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 $\text{Lab} : T \rightarrow \{\text{in}, \text{out}, \text{undec}\}$. The set of all labellings of $T$ is denoted as $\mathcal{L}_T$. Given an AF $\Gamma = \langle A, R \rangle$, a labelling of $\Gamma$ is a labelling of $A$. The set of all labellings of $\Gamma$ is denoted as $\mathcal{L}(\Gamma)$.

Complete labellings can be defined as follows.

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

- $\text{Lab}(a) = \text{in} \iff \forall b \in a^{-}\ \text{Lab}(b) = \text{out}$;
- $\text{Lab}(a) = \text{out} \iff \exists b \in a^{-}\ : \text{Lab}(b) = \text{in}$.

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

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

The function $\text{Ext2Lab}$ provides the connection between extensions and labellings.

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

- $\text{Lab}(a) = \text{in} \iff a \in T$
- $\text{Lab}(a) = \text{out} \iff \exists b \in T \text{ s.t. } b \rightarrow a$
- $\text{Lab}(a) = \text{undec} \iff a \notin T \land \exists b \in T \text{ s.t. } b \rightarrow a$

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

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

The set of complete labellings of $\Gamma$ 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 $\Gamma$ to a set of arguments $I$, in symbol $\Gamma_{\downarrow I}$.

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

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

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

- $E \in \mathcal{BF}_{\sigma}(\Gamma, C)$ if and only if $|\text{SCC}_{\Gamma}| = 1$
• ∀ ∈ SCCΓ (E ∩ S) ∈ GF(Γ \ S, (E \ S)⁺, UΓ(S, E) ∩ C) otherwise

where
• BFΓ (Γ, C) is a function, called base function, that, given an AF Γ = ⟨A, R⟩ such that |GCCΓ| = 1 and a set C ⊆ A, gives a subset of 2A
• UΓ(S, E) = {a ∈ S \ (E \ S)⁺ | ∀b ∈ (a⁻ \ S), b ∈ E⁺}

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

Definition 10. Given an AF Γ = ⟨A, R⟩ and a set C ⊆ A, a set E ⊆ A is: an admissible set of Γ in C if and only if E is an admissible set of Γ and E ⊆ C; a complete extension of Γ in C if and only if E is an admissible set of Γ in C, and every argument α ⊆ 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. ⊆) complete extension of Γ in C; a preferred extension of Γ in C if and only if it is a maximal (w.r.t. ⊆) 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 GF(Γ, C), as defined in Def. 9, returns the σ-extensions in C (with σ ∈ {CO, GF, PR}), provided that BFΓ (Γ, 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 GF (Def. 9), where the chosen base function BFPR 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 Γ = ⟨A, R⟩, ∀E ⊆ A, ∀C ⊆ A, E ∈ GF(Γ, 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 AF Γ = ⟨A, R⟩, ∀E ∈ Eα(Γ), ∀C ⊆ A, proving that E ∈ GF(Γ, C) does not require any knowledge about Γ ∈ Eα(Γ) \ {E}.

A second, rather more articulated, condition of independence requires to identify two sets of SCCs, S := {S₁, ..., Sₙ} ⊆ SCCΓ and PΓ := {P₁, ..., Pₘ} ⊆ SCCΓ such that (1) each SCC in S does not attack the others in S; and (2) each SCC in S is attacked only by SCCs in PΓ.

Remark 2. Given an AF Γ = ⟨A, R⟩, ∀E ⊆ A, ∀C ⊆ A, if there exist S := {S₁, ..., Sₙ} ⊆ SCCΓ such that ∀Sᵢ, Sᵢ⁺ ∩ Sⱼ = ∅, and there exists PΓ := SCCΓ such that ∀Sᵢ ∈ S, (Sᵢ⁺ \ Sᵢ) ⊆ ∪L ∈ PΓ L, then ∀S ∈ S proving that (E \ S) ∈ GF(Γ \ S, (E \ S)⁺, UΓ(S, E) ∩ C) can be determined in function of PΓ and does not require any knowledge about S′ ∈ S \ {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 Γ = ⟨A, R⟩ and returns the set of preferred labellings of Γ. This is simply achieved by invoking (at line 3) P-SCC-REC(Γ, A), where the function P-SCC-REC (GF in Def. 9) receives as input an AF Γ = ⟨A, R⟩ and a set C ⊆ A, and computes the set LPR(Γ, 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: Lab 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 initialised to {Lab} the variable E₀, which stores the set of labellings that are incrementally constructed. At line 5 P-SCC-REC restricts Γ to Γ↓U. Then, at line 6, P-SCC-REC exploits Remark 2 by building a list L := (L₁, ..., Lₙ) of sets of SCCs — (Cormen et al. 2009, p. 617) with some modifications — such that ∀Lᵢ ∈ L, Lᵢ⁺ = {Sᵢ⁺ \ SCCΓ | {Sᵢ⁺ \ SCCΓ | {Sᵢ⁺ \ SCCΓ \ Sᵢ} ⊆ ∪S ∈ Lᵢ S \ S⁻} ∩ Lᵢ \ Sᵢ}.

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 (Sᵢ, Bᵢ) where Sᵢ ∈ SCCΓ, and Bᵢ = LPR(Γ \ Sᵢ, Sᵢ ∩ C). Bᵢ — computed by the function B-PR (Cerutti et al. 2014a, Alg. 4) — is the set of preferred labellings for Sᵢ when no argument in Sᵢ 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, Eᵢ, is initialised. For each S ∈ Lᵢ, Eᵢ(S) is a list of pairs (Lab, Eᵢ(S)Lab) ⊆ LS; to ease of notation, hereafter we omit the pair-structure thus referring directly to Eᵢ(S)Lab which contains the set of preferred labellings of S.
Algorithm 1 Computing preferred labellings of an AF

P-PREF($\Gamma$)
1: Input: $\Gamma = (A, R)$
2: Output: $E_p \in 2^E(\Gamma)$
3: return P-SCC-REC($\Gamma, A$)

Algorithm 2 Computing preferred labellings of an AF in C

P-SCC-REC($\Gamma, C$)
1: Input: $\Gamma = (A, R), C \subseteq A$
2: Output: $E_p \in 2^E(\Gamma)$
3: $(\mathcal{Lab}, U) = \text{GROUNDED}(\Gamma, C)$
4: $E_p := \{\mathcal{Lab}\}$
5: $\Gamma = \Gamma_U$
6: $L := (L_1 := (S_1^1, \ldots, S_k^1), \ldots, L_n := (S_1^n, \ldots, S_k^n))$
7: = SCCLS-LIST($\Gamma$)
8: for $i \in \{1, \ldots, n\}$ do
9: $E_i := \{E_i^{S_1}, \ldots, E_i^{S_k} := ()\}$
10: for $S \in L_i$ do in parallel
11: for $\mathcal{Lab} \in E_p$ do in parallel
12: $(O, I) := L-COND(\Gamma, S, L_i, \mathcal{Lab})$
13: if $I = \emptyset$ then
14: $E_i^S[\mathcal{Lab}] = \{\{a, \text{out}\} \mid a \in O\} \cup \{\{a, \text{undec}\} \mid a \in S \setminus O\}$
15: else if $I = S$ then
16: $E_i^S[\mathcal{Lab}] = B$ where $(S, B) \in M$
17: else if $O = \emptyset$ then
18: $E_i^S[\mathcal{Lab}] = B-PR(\Gamma \downarrow_S, I \cap C)$
19: else
20: $E_i^S[\mathcal{Lab}] = \{\{a, \text{out}\} \mid a \in O\}$
21: $E_i^S[\mathcal{Lab}] = E_i^S[\mathcal{Lab}] \otimes P-\text{SCC-REC}(\Gamma \downarrow_S \setminus O, I \cap C)$
22: end if
23: end if
24: end if
25: end if
26: end for
27: end for
28: end for
29: for $S \in L_i$ do
30: $E_p := \emptyset$
31: for $\mathcal{Lab} \in E_p$ do in parallel
32: $E_p = E_p \cup \{\mathcal{Lab}\}$
33: end for
34: $E_p := E_p$
35: end for
36: end for
37: return $E_p$

constructed on the basis of a specific labelling $\mathcal{Lab}$ identified

Algorithm 3 Greedy computation of base cases

GREEDY($L, C$)
1: Input: $L = (L_1, \ldots, L^u := (S_1^1, \ldots, S_k^1))$, $C \subseteq A$
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: $B := B-PR(\Gamma \downarrow_S, S \cap C)$
6: $M = M \cup \{(S, B)\}$
7: end for
8: return $M$

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($\Gamma, S, L_i, \mathcal{Lab}$) at line 12 computes the effect of previous SCCs, and returns $(O, I)$, where:

• $O = \{a \in S \mid \exists b \in T \cap a^- : \mathcal{Lab}(b) = \text{in}\}$ and
• $I = \{a \in S \mid \forall b \in T \cap a^+ : \mathcal{Lab}(b) = \text{out}\}$,

with $T = \bigcup_{S \in L_i} S$. Variable $O$ is set to include arguments of $S$ that are attacked by “outside” in-labelled arguments according to $\mathcal{Lab}$, 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 $E_i^S[\mathcal{Lab}]$ are merged together ($E_1 \otimes E_2 = \{\mathcal{Lab}_1 \cup \mathcal{Lab}_2\} | \mathcal{Lab}_1 \in E_1, \mathcal{Lab}_2 \in E_2\}$) with the $\mathcal{Lab}$ 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
**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.

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P2G</th>
<th>P4</th>
<th>P4G</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC score</td>
<td>50.0</td>
<td>58.6</td>
<td>44.7</td>
<td>74.9</td>
<td>56.9</td>
</tr>
<tr>
<td>% success</td>
<td>58.8</td>
<td>68.3</td>
<td>57.3</td>
<td>74.9</td>
<td>65.8</td>
</tr>
<tr>
<td>% best</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>74.4</td>
<td>0.0</td>
</tr>
<tr>
<td>Avg runtime</td>
<td>439.4</td>
<td>464.3</td>
<td>385.7</td>
<td>269.6</td>
<td>384.8</td>
</tr>
<tr>
<td>Speedup</td>
<td>–</td>
<td>1.9</td>
<td>1.3</td>
<td>2.8</td>
<td>1.7</td>
</tr>
</tbody>
</table>

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).

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.)
is not completely compensated by using 2 processors only, especially 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 (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 et al. 2012; 2014) can be exploited for determining conditions of independent computation and thus exploiting parallel executions on structures different from SCCs.

**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).
References


