Strategies in Flexible Dispute Derivations for Assumption-based Argumentation

Martin Diller*,†, Sarah Alice Gaggl† and Piotr Gorczyca†

[1] *Logic Programming and Argumentation Group, TU Dresden, Germany*

Abstract

We put forward and provide an empirical evaluation of strategies for fully automatic flexible dispute derivations for assumption-based argumentation (ABA). These being a novel dialectical means of judging claims in the context of ABA. Central among our findings is that our naive direct implementation of flexible disputes outperforms the current state-of-the-art system for ABA disputes (acceptance of claims, admissible semantics) especially when allowing forward moves from premisses to claims in addition to backwards from conclusions to premisses.

Keywords

Strategies, Dispute derivations, Assumption-based argumentation

1. Introduction

Dispute derivations [1, 2] are the main native (vs. reduction-based [3]) reasoning method for assumption-based argumentation or ABA [4]. Based on games for abstract argumentation [5], they are conceived of as a dispute with arguments built chaining rules from assumptions and facts being exchanged by a proponent and an opponent of some claims under scrutiny. In [6], building on previous work on ABA disputes, we presented a new form of disputes which we called "flexible dispute derivations". The main motivation behind these is to also have forward reasoning (reasoning from premisses to conclusions) and not only backward reasoning (from conclusions to premisses) as in previous versions of dispute derivations. This allows for novel, arguably quite natural, and also often shorter forms of dispute derivations for deciding acceptance of claims w.r.t. the classical admissible (and, hence, complete and preferred) semantics for argumentation. Having forward moves also allows a straightforward generalization of dispute derivations for the stable semantics as well as to determine sets of assumptions congruous with claims (w.r.t. the admissible, complete, and stable semantics).

Yet flexible dispute derivations as defined in [6] also revise existing disputes for ABA even when only making use of backward moves. The main revision is that both the proponent and

SAFA'22: Fourth International Workshop on Systems and Algorithms for Formal Argumentation 2022, September 13, 2022, Cardiff, Wales, United Kingdom

*Corresponding author.

†Authors are listed in alphabetical order.

martin.diller@tu-dresden.de (M. Diller); sarah.gaggl@tu-dresden.de (S. A. Gaggl); piotr.gorczyca@tu-dresden.de (P. Gorczyca)

0000-0001-6342-0756 (M. Diller); 0000-0003-2425-6089 (S. A. Gaggl); 0000-0002-6613-6061 (P. Gorczyca)

opponent are completely omniscient in that they remember all arguments (and sub-arguments) put forward during disputes; we have shown that this avoids redundancy. Flexible disputes, as a consequence, generalise graph-based disputes [2], not only by also having forward moves, but also in that all the arguments put forward in disputes, and not only those of the proponent, can be represented as a (single shared) graph.

In [6] we also presented a prototype system, `aba-dd-rule-based`. At that moment, in contrast to available systems for dispute derivations, the system in question was mainly interactive. We still believe the most obvious virtues of disputes for ABA to be in interactive argumentation-based reasoning. Nevertheless, (partial) automatic search for disputes can be a component of such an interactive system. Furthermore, given the above mentioned differences of flexible dispute derivations w.r.t. previous versions of ABA disputes it is of interest to compare the performance of our system w.r.t. available systems computing dispute derivations. Specifically, in [6] we hypothesized that (especially so called conservative) forward moves could lead to more efficient dispute derivations.

In this work we present, first of all, strategies and related algorithms for finding successful disputes (for the admissible, complete, and stable semantics) which we implemented in a revamped system which we appropriately redubbed to `flexABle`[1]. Secondly, we provide an in-depth experimental study of such strategies. We also compare the performance of our system to the main state-of-the-art system computing dispute derivations, `abagraph`, for determining acceptance of claims w.r.t. the admissible semantics[2]. Despite ours being a comparably naive direct implementation of flexible dispute derivations, while `abagraph` is based on SICStus Prolog, we show that our system clearly outperforms `abagraph` especially when making use of (conservative) forward moves. We finally, following the lead of [7, 8], also evaluate approximate strategies for computing disputes within `flexABle`. Here we find good accuracy, but not as promising a boost in performance. This work is based on [9] to which we refer for a detailed presentation.

2. Background

For reasons of space we will only review the absolute minimal elements of ABA and flexible dispute derivations required to follow our work. We refer to e.g. [4] and [6].

An ABA framework is a tuple $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ where $(\mathcal{L}, \mathcal{R})$ is a deductive system, with a language $\mathcal{L}$ and a set of inference rules $\mathcal{R}$. Elements of the (non-empty) set $\mathcal{A} \subseteq \mathcal{L}$ are referred to as *assumptions* and $\overline{}$ is a total mapping from $\mathcal{A}$ into $\mathcal{L}$, where $\overline{a}$ is the *contrary* of a. As in all other work on reasoning methods for ABA we are aware of, we make a few further crucial assumptions about ABA frameworks: 1) that $\mathcal{L}$ is finite[3], 2) that $\mathcal{L}$ consists of propositional atoms, 3) that contraries do not appear in the heads of rules (i.e. we deal with flat ABA), 4) that all elements in $\mathcal{R}$ are given explicitly; given the previous assumptions, this means that they are Horn rules of the form $h \leftarrow b_1, \ldots, b_n$ where h is an atom while $b_1, \ldots, b_2$ can be atoms or contraries

[1]https://github.com/gorczyca/aba-dd-rule-based

[2]The only semantics which both systems caters for - `abagraph` also supports the grounded semantics.

[3]Definitions of dispute derivations, including flexible dispute derivations as defined in [6] (in their argument-based version), do not require this, but all implementations including ours do.

Notation	Description
$\mathcal{D} = \mathbb{P} \cap \mathcal{A}$	Defences
$\mathcal{C} = \{u \in \mathcal{A} \mid \overline{u} \in \mathbb{P}\}$	Culprits
$\mathcal{J}_{\mathbb{B}} = \mathcal{R} \setminus \mathbb{B}$	Remaining rules for the opponent
$\mathcal{J}_{\mathbb{P}} = \mathcal{R} \setminus \mathbb{P}$	Remaining rules for the proponent
$\mathcal{J}_{\mathbb{B}}^{-} = \{h \leftarrow B \in \mathcal{J}_{\mathbb{B}} \mid B \cap \mathcal{C} \neq \emptyset\}$	Remaining rules blocked for the opponent
$\mathcal{J}_{\mathbb{P}}^{\sim} = \{h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \mid (\{h\} \cup B) \cap (\overline{B} \cup \mathcal{C} \cup \overline{\mathcal{D}}) \neq \emptyset\}$	Remaining rules blocked for the proponent
$(\mathbb{P} \cap \mathcal{L})^{\downarrow} = \{s \in \mathbb{P} \cap \mathcal{L} \mid \neg \exists h \leftarrow B \in \mathbb{P} \text{ with } h = s\}$	Played unexpanded statements of the proponent
$(\mathbb{B} \cap \mathcal{L})^{\uparrow\uparrow} = \{s \in (\mathbb{B} \cap \mathcal{L}) \mid \neg \exists h \leftarrow B \in (\mathcal{J}_{\mathbb{B}} \setminus \mathcal{J}_{\mathbb{B}}^{-} \text{ with } h = s)\}$	Played fully expanded statements
$\mathbb{B}^{-} = (\mathbb{B} \cap \mathcal{C}) \cup \{s \in (\mathbb{B} \cap \mathcal{L})^{\uparrow\uparrow} \setminus \mathcal{A} \mid \neg \exists h \leftarrow B \in (\mathbb{B} \cap \mathcal{R}) \setminus \mathbb{B}^{-} \text{ with } h = s\} \cup \{h \leftarrow B \in \mathbb{B} \cap \mathcal{R} \mid B \cap \mathbb{B}^{-} \neq \emptyset\}$	Played blocked pieces
$\mathbb{P}^{*} = (\mathbb{P} \cap \mathcal{A}) \cup \{h \leftarrow B \in (\mathbb{P} \cap \mathcal{R}) \mid B \subseteq \mathbb{P}^{*}\} \cup \{s \in (\mathbb{P} \cap (\mathcal{L} \setminus \mathcal{A})) \mid \exists h \leftarrow B \in \mathbb{P}^{*} \text{ with } h = s\}$	“Complete” played pieces of the proponent
$\mathbb{B}^{*\setminus -} = (\mathbb{B} \cap (\mathcal{A} \setminus \mathcal{C})) \cup \{h \leftarrow B \in (\mathbb{B} \setminus \mathbb{B}^{-}) \cap \mathcal{R} \mid B \subseteq \mathbb{B}^{*\setminus -}\} \cup \{s \in (\mathbb{B} \setminus \mathbb{B}^{-}) \cap (\mathcal{L} \setminus \mathcal{A}) \mid \exists h \leftarrow B \in \mathbb{B}^{*\setminus -} \text{ with } h = s\}$	Unblocked complete played pieces of the opponent
$\mathbb{B}_{S}^{!\setminus -} = ((\mathbb{B} \setminus \mathbb{B}^{-}) \cap S) \cup \{s \in (\mathbb{B} \setminus \mathbb{B}^{-}) \cap \mathcal{L} \mid \exists h \leftarrow B \in \mathbb{B}_{S}^{!\setminus -} \cap \mathcal{R} \text{ with } s \in B\} \cup \{h \leftarrow B \in (\mathbb{B} \setminus \mathbb{B}^{-}) \cap \mathcal{R} \mid h \in \mathbb{B}_{S}^{!\setminus -}\}$	Unblocked pieces supporting statements in $S \subseteq \mathcal{L}$
$\mathcal{A}^{!} = \mathcal{A} \cap \mathbb{B}_{\overline{\mathcal{D}}}^{!\setminus -}$	Culprit candidates
$\mathcal{J} = \{u \in \mathcal{A} \setminus \mathcal{C} \mid \overline{u} \notin \mathbb{B}^{*\setminus -}\}$	Currently defended assumptions
$\mathcal{H} = \{h \mid h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \setminus \mathcal{J}_{\mathbb{P}}^{\sim}, B \subseteq \mathbb{P}^{*}\}$	Statements derivable from $\mathbb{P}^{*}$

Table 1
Auxiliary notation for rule-based flexible derivations. All notions defined w.r.t. a dispute state $(\mathbb{B}, \mathbb{P})$.

thereof.

Dispute derivations give a dialectical view on acceptance of claims in ABA. In [6] we presented two definitions of flexible dispute derivations, one where the proponent and opponent exchange arguments, and a second implementation-oriented definition where the proponent and opponent exchange claims and rules. The first is, arguably, more intuitive but we have space here to present only the second definition which is at the basis of our system `flexABle`. Needless to say, there is a clear correspondence between both definitions and at each dispute state it is straightforward to convert from one representation to another (something also possible in `flexABle`).

Rule-based flexible dispute derivations are sequences of dispute states of the form $(\mathbb{B}, \mathbb{P})$, where $\mathbb{B} \subseteq (\mathcal{L} \cup \mathcal{R})$ and $\mathbb{P} \subseteq \mathbb{B}$. $\mathbb{B}$ is the opponents claim and rule-set, and $\mathbb{P}$ the proponents. The initial dispute state is of the form (γ, γ) with $\gamma \subseteq \mathcal{L}$ (for which no atom and its contrary are in γ) being the goals. To define moves in dispute derivations the auxiliary notation in Table 1 is needed.

Then, a proponent dispute state advancement from a dispute state $(\mathbb{B},\mathbb{P})$ is a dispute state $(\mathbb{B}',\mathbb{P}')$ with $\mathbb{P}' = \mathbb{P} \cup T$, $\mathbb{B}' = \mathbb{B} \cup T$, $X_1 \subseteq \overline{\mathcal{A}}$, $X_1 \subseteq \mathcal{A}$, where T is defined as in Table 2.

Move type	V.	T definition
$\textsc{pb}\text{-}\ (\overline{\mathcal{A}^!} \cup X_1)$	1	$T = \{h \leftarrow B\} \cup B$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \setminus \mathcal{J}_{\mathbb{P}}^{\sim}$ with $h \in (\mathbb{P} \cap \mathcal{L})^{\downarrow}$
	2	$T = \{h\} \cup \{h \leftarrow B\} \cup B$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \setminus \mathcal{J}_{\mathbb{P}}^{\sim}$ with $h \in (\overline{\mathcal{A}^!} \cup X_1) \setminus (\mathbb{P} \cup \overline{\mathcal{D}})$
$\textsc{pf}\text{-}((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup X_2)$	1	$T = \{h\} \cup \{h \leftarrow B\}$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{P}} \setminus \mathcal{J}_{\mathbb{P}}^{\sim}, B \subseteq \mathbb{P}^*$
	2	$T = \{u\}$ for $u \in ((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup X_2) \setminus (\mathbb{P} \cup \{\overline{u}\} \cup \mathcal{C} \cup \overline{\mathcal{D}})$

Table 2
Possible proponent moves. Column "Move type" denotes the type of move (backward or forward), with column "V." indicating two variants (1 and 2).

An opponent dispute state advancement from a dispute state $(\mathbb{B},\mathbb{P})$ is a dispute state $(\mathbb{B}',\mathbb{P})$ with $\mathbb{B}' = \mathbb{B} \cup T$, $Y_1 \subseteq \overline{\mathcal{A}}$, $Y_2 \subseteq \mathcal{A}$, where T is defined as in Table 3.

Move type	V.	T definition
$\textsc{ob}\text{-}(\overline{\mathcal{D}} \cup Y_1)$	1	$T = \{h \leftarrow B\} \cup B$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{B}} \setminus \mathcal{J}_{\mathbb{B}}^{\sim}$ with $h \in \mathbb{B}^{!\setminus -}_{\overline{\mathcal{D} \cup Y_1}} \cap \mathcal{L}$
	2	$\{h\} \cup \{h \leftarrow B\} \cup B$ for for $h \leftarrow B \in \mathcal{J}_{\mathbb{B}} \setminus \mathcal{J}_{\mathbb{B}}^{\sim}$ with $h \in \overline{\mathcal{D}} \cup Y_1$
$\textsc{of}\text{-}((\overline{\mathcal{D}} \cap \mathcal{A}) \cup Y_2)$	1	$\{h\} \cup \{h \leftarrow B\}$ for $h \leftarrow B \in \mathcal{J}_{\mathbb{B}} \setminus \mathcal{J}_{\mathbb{B}}^{\sim}$ with $\mathbb{B}^{* \setminus -}$ for each $b \in B$
	2	$\{u\}$ for $u \in ((\overline{\mathcal{D}} \cap \mathcal{A}) \cup Y_2) \setminus (\mathcal{A} \cap \mathbb{B})$

Table 3
Possible opponent moves.

The above definitions of moves allow several instantiations by varying the parameters X_1,X_2,Y_1,Y_2. In this work we focus on dispute advancements as given in Table 4. The termination criteria we consider in this work are in Table 5[4]. Then, we have that dispute derivations $D + \textsc{ta}$ with $D \in \{\textsc{dab}, \textsc{dabf}, \textsc{dc}, \textsc{ds}\}$ are sound and complete for acceptance of claims w.r.t. the admissible (thus also complete, preferred) semantics, while $\textsc{dc} + \textsc{tc}$ and $\textsc{ds} + \textsc{ts}$ are sound and complete for determining respectively (full) complete and stable assumption sets congruous with the goal claims [6].

Advancement	Proponent	Opponent
$\textsc{dab}$	$\textsc{pb}\text{-}(\overline{\mathcal{A}^!})$, $\textsc{pf}\text{-}(\overline{\mathcal{A}^!} \cap \mathcal{A})\text{-}2$	$\textsc{ob}\text{-}(\overline{\mathcal{D}})$, $\textsc{of}\text{-}(\overline{\mathcal{D}} \cap \mathcal{A})\text{-}2$
$\textsc{dabf}$	$\textsc{pb}\text{-}(\overline{\mathcal{A}^!})$, PF-$(\overline{\mathcal{A}^!} \cap \mathcal{A})$	$\textsc{ob}\text{-}(\overline{\mathcal{D}})$, $\textsc{of}\text{-}(\overline{\mathcal{D}} \cap \mathcal{A})\text{-}2$
$\textsc{dc}$	$\textsc{pb}\text{-}(\overline{\mathcal{A}^!})$, $\textsc{pf}\text{-}((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup \mathcal{J})$	$\textsc{ob}\text{-}(\overline{\mathcal{D}} \cup \overline{\mathcal{J}})$, $\textsc{of}\text{-}((\overline{\mathcal{D}} \cup \overline{\mathcal{J}}) \cap \mathcal{A})\text{-}2$
$\textsc{ds}$	$\textsc{pb}\text{-}(\overline{\mathcal{A}^!})$, $\textsc{pf}\text{-}(\mathcal{A})$	$\textsc{ob}\text{-}(\overline{\mathcal{D}})$, $\textsc{of}\text{-}(\overline{\mathcal{D}} \cap \mathcal{A})\text{-}2$

Table 4
Advancement types. Second and third columns give the allowed moves by the players.

[4]Note that here we introduced a slight addition to $\textsc{tc}$ correcting [6]. For details see Chapter 3 of [9].

C	Prop. winning	Opp. cannot move	Prop. cannot move
TA	$\gamma \cup \overline{\mathcal{C}} \subseteq \mathbb{P}^*, \overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus -} = \varnothing$	OB-$(\overline{\mathcal{D}}) \cup$ OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2 or OF-$(\mathcal{A})$	PB-$(\overline{\mathcal{A}^!}) \cup$ PF-$(\overline{\mathcal{A}^!} \cap \mathcal{A})$-2 or PF-$(\mathcal{A})$
TC	$\gamma \cup \overline{\mathcal{C}} \subseteq \mathbb{P}^*, \overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus -} = \varnothing, \mathcal{J} \subseteq \mathcal{D}, \mathcal{H} \subseteq \mathbb{P}^*$	OB-$(\overline{\mathcal{D}}) \cup$ OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2 or OF-$(\mathcal{A})$	PB-$(\overline{\mathcal{A}^!}) \cup$ PF-$((\overline{\mathcal{A}^!} \cap \mathcal{A}) \cup \mathcal{J})$ or PF-$(\mathcal{A})$
TS	$\gamma \cup \overline{\mathcal{C}} \subseteq \mathbb{P}^*, \overline{\mathcal{D}} \cap \mathbb{B}^{*\setminus -} = \varnothing, \mathcal{D} \cup \mathcal{C} = \mathcal{A}$	OB-$(\overline{\mathcal{D}}) \cup$ OF-$(\overline{\mathcal{D}} \cap \mathcal{A})$-2 or OF-$(\mathcal{A})$	PB-$(\overline{\mathcal{A}^!}) \cup$ PF-$(\mathcal{A})$

Table 5
Termination criteria. Proponent wins when the "Prop. winning" condition is satisfied and opponent cannot move according to any of the combination of moves given by the disjuncts under "Opp. cannot move". Opponent wins when the "Prop. winning" condition is not satisfied and the proponent cannot move according to any of the combination of moves given by the disjuncts under "Prop. cannot move".

For an example, consider the ABA framework with $\mathcal{A} = \{a,b,c,d\}$, where $\overline{a} = t$, $\overline{b} = r$, $\overline{c} = t$, $\overline{d} = c$. Also: $\mathcal{R} = \{p \leftarrow q;\ \ q \leftarrow a;\ \ r \leftarrow p;\ \ t \leftarrow b;\ \ t \leftarrow p,s;\ \ t \leftarrow q,u,d\}$. A DC + TA (as well as DS + TA, DC + TC, DS + TS) derivation for p has the initial state $(\{p\},\{p\})$. The first two moves are backward of the proponent using move type PB1[5], first with $p \leftarrow q$ and then $q \leftarrow a$ (so now $\mathcal{D} = \{a\}$). Then, the opponent makes an OB2 move using $t \leftarrow b$. Here the proponent makes a "conservative" forward PF1 move (also a PB2 move in this case) using $r \leftarrow p$ (so $\mathcal{C} = \{b\}$). Finally, the proponent makes a "non-conservative" forward move (before this step $\mathcal{I} \setminus \mathcal{D} = \mathcal{A} \setminus (\mathcal{D} \cup \mathcal{C}) = \{c,d\}$) PF2 advancing c ($\mathcal{D} = \{a,c\}$, $\mathcal{C} = \{b,d\}$). The dispute ends with the opponent not being able to advance further in OF-$(\mathcal{A})$ manner. The assumption set $\mathcal{D} = \{a,c\}$ is admissible, complete, and stable.

3. Strategies and algorithms

Strategies. We start by introducing a simple[6] "preference-on-move-types-based" framework for strategies in which these consist of tuples $(<_M, H, R, A)$.

Each dispute derivation variant from Section 2 has associated a set of possible move types from $M = \{$PB1,PB2,PF1,PF2,OB1,OB2,OF2$\}$. So the first aspect of a strategy is a preference relation, a strict total order $<_M$ on move types. At each step in a dispute, from all possible moves, moves of type with lowest ranking w.r.t. $<_M$ are chosen.

There will often be several possible moves having lowest ranking w.r.t. $<_M$. If the moves involve adding assumptions to the players sets (moves of type PF2, OF2) then the strategies in this work all return a random ordering of the assumptions. H, R, and A come into play when the moves involve adding rules to the players sets. Specifically, H is first of all a "rule-head selecting function". Given a set of rules, it selects a subset by considering rule-heads. We consider two such functions, one `mostRules` which returns a (random) maximal set of rules from those rule

[5]Here and in the remainder of this work we often simplify the notation for moves by leaving the instantiations of parameters implicit.

[6]Note in particular we define strategies to be homogeneous among players. Non-homogeneous strategies can be defined in `flexABle`.

sets that result when partitioning the rules according to their heads. The other, `leastRules`, selects a minimal such set.

So H gives a new set of rules to choose from, which strategies now order by using R and A. First R associates a natural number to rules. We define three such "rule comparison functions": `bodySize`, `newStatementsSize` and `lookaheadSize`. The function `bodySize` simply returns the cardinality of the body of rules ($|B|$ for $r = h \leftarrow B$). The function `newStatementsSize` computes how many new statements the rule introduces into the dispute. Concretely, at a dispute state $(\mathbb{B},\mathbb{P})$, with the turn of $p \in \{\texttt{proponent},\texttt{opponent}\}$, $\texttt{newStatementsSize}(h \leftarrow B) = |(\{h\} \cup B) \setminus \mathbb{X}|$ with $\mathbb{X} = \mathbb{B}$ if $p = \texttt{opponent}$ and $\mathbb{X} = \mathbb{P}$ if $p = \texttt{proponent}$. The function `lookaheadSize` is a measure of how close a move will get the proponent to winning[7]. Finally, to the last element of a strategy, A is one of $\bot$ or $\top$ with rules chosen by H being ordered ascendingly w.r.t. R if $A = \top$ and descendingly if $A = \bot$ (ties are resolved randomly).

Given a set of possible moves $O = \{(t_1,p_1),\ldots,(t_n,p_n)\}$ consisting of a move type t_i and piece p_i, either an assumption or a rule, we denote the application of a strategy $Strat = (<_M ,H,R,A)$ on the possible moves as $Strat(O) = (t_x,P)$. Here $1 \leq x \leq n$ s.t. there is no t_i with $t_i <_M t_x$ and P is a random ordering of $P' = \{p \mid (t_x,p) \in O\}$ if $t_x \in \{\texttt{PF2},\texttt{OF2}\}$ or otherwise ($t_x \in \{\texttt{PB1},\texttt{PB2},\texttt{PF1},\texttt{OB1},\texttt{OB2}\}$) an ordering of $H(P')$ w.r.t. R and A.

Algorithms. We turn to describing the search algorithm for flexible ABA disputes that underlies our system `flexABle`. We first extend dispute derivation states to be tuples $S = (\mathbb{B},\mathbb{P},I_a,I_c)$ where I_a represents assumptions the proponent decides not to use (via `PF2` moves), while I_c are assumptions the proponent decides not to attack (using moves `PB2` or `PF2`). In the algorithms we denote sequences of elements where the order is important using delimiters $\langle\!\langle\,\rangle\!\rangle$ rather than $\{\}$. Function `GetNewStates` from Algorithm 1 has a current (extended) dispute state, a strategy and a set of possible moves at the dispute state as input and returns the sequence of dispute states to be examined next. These are determined, first of all, by the strategy (line 2 in Algorithm 1) and then by the move type to which priority is given by the strategy (switch statement line 4). Here move types `PB1`,`PB2`, and `PF2` introduce branching. In the case of `PB1` (lines 5-6) this is because several possible rules for a claim can be considered, for `PB2` (lines 7-11) in addition the proponent can decide not to consider some candidate for a culprit, while for `PF2` move types (lines 15-24) the proponent can choose to use an assumption or not. Move type `PF1` (lines 12-14) and all moves of the opponent (lines 25-30) do not introduce branching.

The main search procedure is then the recursive function `GetSuccessfulDDs` from Algorithm 2. This takes as input a sequence of current dispute states, a set of found successful dispute states, a strategy *Strat*, a termination criteria C and an advancement type A. In addition it has an additional parameter *St* for defining the search to be depth-first or breadth-first. The procedure is initialized with input $(\langle\!\langle(\gamma,\gamma,\emptyset,\emptyset)\rangle\!\rangle$, $\emptyset$, *Strat*, C, A, *St*$)$ (γ are the goals). If the sequence of current dispute states is empty, the procedure returns the set of successful states (lines 2-3). Otherwise, the first state in the sequence of current dispute states is taken for consideration and removed from the current states (line 4), all possible moves at that dispute state are computed (according to advancement A) (line 5), and moves which would make $I_a \cap \mathcal{D} \neq \emptyset$ or $I_c \cap \mathcal{C} \neq \emptyset$

[7]For lack of space we refer to Section 4.1 of [9] for the more complex definition.

Algorithm 1: Function `GetNextStates`.

```
input  : $S = (\mathbb{B},\mathbb{P},I_a,I_c)$, $Strat = (<_M,H,R,A)$, $O = \{(t_1,p_1),\dots,(t_n,p_n)\}$ // current dispute state, strategy, and set
         of possible moves at the current state $S$ w.r.t. the chosen advancement type
output : $newStates = \langle\!\langle(\mathbb{B}_1,\mathbb{P}_1,I_{a_1},I_{c_1}),\dots,(\mathbb{B}_l,\mathbb{P}_l,I_{a_l},I_{c_l})\rangle\!\rangle$ // chosen sequence of next states to be examined next;
         here $1 \le l \le n$.

function GetNextStates($S$, $Strat$, $O$)
    $(chosenMoveType, chosenMovesSequence) \longleftarrow Strat(O)$ // get chosen move type and moves sequence with
      $Strat$
    $newStates \longleftarrow \langle\!\langle\rangle\!\rangle$ // initialize $newStates$
    switch $chosenMoveType$ do
        case PB1 do // $chosenMovesSequence$ is a sequence of rules
            $newStates \longleftarrow \langle\!\langle(\mathbb{B}\cup(\{h\}\cup B_1),\mathbb{P}\cup(\{h\}\cup B_1),I_a,I_c),\dots,(\mathbb{B}\cup(\{h\}\cup B_l),\mathbb{P}\cup(\{h\}\cup B_l),I_a,I_c)\rangle\!\rangle$
              for $h \leftarrow B_i \in chosenMovesSequence$ // create a sequence of new dispute states using rules in
              $chosenMovesSequence$, preserve the order
        case PB2 do // $chosenMovesSequence$ is a sequence of rules
            $attackingStates \longleftarrow$
              $\langle\!\langle(\mathbb{B}\cup(\{h\}\cup B_1),\mathbb{P}\cup(\{h\}\cup B_1),I_a,I_c),\dots,(\mathbb{B}\cup(\{h\}\cup B_{l-1}),\mathbb{P}\cup(\{h\}\cup B_{l-1}),I_a,I_c)\rangle\!\rangle$ for
              $h \leftarrow B_i \in chosenMovesSequence$ // create a sequence of new dispute states in which the proponent
              attacks some culprit candidate using rules in $chosenMovesSequence$, preserve the order
            $attackedAssumptions \longleftarrow \{a \in \mathcal{A} \mid \overline{a} \leftarrow B \in chosenMovesSequence\}\}$ // get the set of assumptions
              attacked by head of rules from $chosenMovesSequence$
            $ignoringState \longleftarrow (\mathbb{B},\mathbb{P},I_a,I_c \cup attackedAssumptions)$ // get new state in which the attacked
              assumptions get ignored
            $newStates \longleftarrow attackingStates.append(ignoringState)$ // append $ignoringState$ to $attackingStates$
        case PF1 do // $chosenMovesSequence$ is a sequence of rules
            $h \leftarrow B \longleftarrow chosenMovesSequence[0]$ // take first rule, does not matter which
            $newStates \longleftarrow \langle\!\langle(\mathbb{B}\cup(\{h\}\cup B),\mathbb{P}\cup(\{h\}\cup B),I_a,I_c)\rangle\!\rangle$ // get new state by adding rule statements to
              $\mathbb{B}$ and $\mathbb{P}$
        case PF2 do // $chosenMovesSequence$ is a sequence of assumptions
            $b \longleftarrow chosenMovesSequence[0]$ // take first assumption $b$
            $takingState \longleftarrow (\mathbb{B}\cup\{a\},\mathbb{P}\cup\{a\},I_a,I_c)$ // get new state by adding $b$ to $\mathbb{B}$ and $\mathbb{P}$
            $attackedAssumptions \longleftarrow \{a \in \mathcal{A} \mid \overline{a} = b\}$ // get set of assumptions attacked by $b$
            $ignoringState \longleftarrow Nil$ // initialize $ignoringState$
            if $attackedAssumptions \neq \varnothing$ then // if $b$ attacks some assumptions
                $ignoringState \longleftarrow (\mathbb{B},\mathbb{P},I_a,I_c \cup attackedAssumptions)$ // get new $ignoringState$ by adding
                  attacked assumptions to ignored culprit candidates $I_c$
            else // otherwise, get new $ignoringState$ by adding $b$ to ignored assumptions $I_a$
                $ignoringState \longleftarrow (\mathbb{B},\mathbb{P},I_a \cup \{b\},I_c)$
            $newStates \longleftarrow \langle\!\langle takingState, ignoringState\rangle\!\rangle$ // return both states
        case OB1 or OB2 or OF1 do // $chosenMovesSequence$ is a sequence of rules
            $h \leftarrow B \longleftarrow chosenMovesSequence[0]$ // take first rule
            $newStates \longleftarrow \langle\!\langle(\mathbb{B}\cup(\{h\}\cup B),\mathbb{P},I_a,I_c)\rangle\!\rangle$ // get new state by adding rule to $\mathbb{B}$
        case OF2 do // $chosenMovesSequence$ is a sequence of assumptions
            $a \longleftarrow chosenMovesSequence[0]$ // take first assumption $a$
            $newStates \longleftarrow \langle\!\langle(\mathbb{B}\cup\{a\},\mathbb{P},I_a,I_c)\rangle\!\rangle$ // get new state by adding $a$ to $\mathbb{B}$
    return $newStates$
```

hold are filtered-out (line 6). Then, the termination criteria C is applied at the taken state (line 9), if the proponent wins the state is appended to the successful states (lines 10-11), if the opponent wins the state is ignored (line 12), while if there is no winner yet (lines 13-18) function `GetNextStates` defined in Algorithm 1 is used to compute next possible states. The obtained sequence of the next possible states is then either appended at the beginning or the end of the sequence of current dispute states, depending on the search type (depth or breadth first). Then, the function is called again with the new sequence of current dispute states and set of successful dispute states (line 19).

Algorithm 2: Function `GetSuccessfulDDs`.

```
input  : $States = \langle\langle(\mathbb{B}_1,\mathbb{P}_1,I_{a_1},I_{c_1}),\ldots,(\mathbb{B}_n,\mathbb{P}_n,I_{a_n},I_{c_n})\rangle\rangle$ // collection of dispute states
         $SuccStates = \{(\mathbb{B}_1,\mathbb{P}_1,I_{a_1},I_{c_1}),\ldots,(\mathbb{B}_l,\mathbb{P}_l,I_{a_l},I_{c_l})\}$ // currently found succ. states
         $Strat = (<_M,H,R,A)$, $C \in \{\mathrm{TA},\mathrm{TC},\mathrm{TS}\}$, $A \in \{\mathrm{DAB},\mathrm{DABF},\mathrm{DC},\mathrm{DS}\}$, $St \in \{\mathrm{DFS},\mathrm{BFS}\}$ // strategy, termination
         criteria, advancement type, search type (depth-first-search or breadth-first search)
output : $succStates = \{(\mathbb{B}_1,\mathbb{P}_1,I_{a_1},I_{c_1}),\ldots,(\mathbb{B}_k,\mathbb{P}_k,I_{a_k},I_{c_k})\}$ // successful states found

1  function GetSuccessfulDDs($States$, $SuccStates$, $Strat$, $C$, $A$, $St$)
2      if $States = \langle\langle\rangle\rangle$ then // if there are no more states to examine
3          return $SuccStates$ // return found successful states
4      $currState \longleftarrow States.pop()$ // take first state from $States$
5      $possibleMoves \longleftarrow getPossibleMoves(currState, A)$ // get possible moves at the current state w.r.t.
         advancement type $A$
6      $possibleMoves \longleftarrow filter(possibleMoves, currState)$ // filter possible moves
7      $states' \longleftarrow States$ // initialize $states'$ with remaining states
8      $succStates' \longleftarrow SuccStates$ // initialize $succStates'$ with currently found successful states
9      switch $isOver(currState, possibleMoves, C)$ do // check if dispute over w.r.t. termination criteria $C$
10         case ProponentWon do // $C$ indicates that proponent has won
11             $succStates'.append(currState)$
12         case OpponentWon do // $C$ indicates that opponent has won
               // do nothing...
13         case NotOver do // $C$ indicates that dispute has not terminated
14             $newStates \longleftarrow$ GetNextStates ($currState$, $Strat$, $possibleMoves$)
15             if $searchType$ = DFS then // if DFS prepend $newStates$ to the beginning of the list
16                 $states' \longleftarrow newStates.join(states')$
17             else // else, if BFS append $newStates$ at the end
18                 $states' \longleftarrow states'.join(newStates)$
19     GetSuccessfulDDs($states'$, $succStates'$, $Strat$, $C$, $A$, $St$) // continue searching
```

Approximate Reasoning. We follow the lead of [7] but mainly [8] in defining approximate reasoning for flexible ABA disputes. We consider static and dynamic sampling of rules. In the first case, rules are removed from the entire ABA framework, thus not being available in entire disputes. In the second case, rules are removed at each step; such rules may become available for use at a later dispute state. As in [8] sampling is done independently for players: each gets assigned a value denoting the probability of a rule not to be removed, p_{prop} and p_{opp} for the proponent and the opponent respectively.

Str.	$<_M$	H	R	A
S_1	⟪ PF1, OB1, OB2, OF2, PB1, PB2, PF2 ⟫	`leastRules`	`newStatementsSize`	$\top$
S_{1a}		`mostRules`	`newStatementsSize`	$\bot$
S_{1b}	⟪ OB1, OB2 , OF2, PB1, PB2, PF2, PF1 ⟫	`leastRules`	`newStatementsSize`	$\top$
S_2	⟪ PF1, PB2, PF2, PB1, OB2, OF2, OB1 ⟫	`leastRules`	`newStatementsSize`	$\top$
S_{2a}		`mostRules`	`newStatementsSize`	$\bot$
S_{2b}	⟪ PB2, PF2, PB1, OB2, OF2, OB1, PF1 ⟫	`leastRules`	`newStatementsSize`	$\top$
S_{AG}	⟪ PF1, PB1, OB2, OF2, OB1, PB2, PF2 ⟫	*random*	`lookaheadSize`	$\top$

Table 6
Strategies of `flexABle` selected for experimental evaluation.

Str.	*Turn*	*OppSet*	*Sentence*	*Rule*
Default	*Proponent*	*Smallest*	*Patient*	*Look-ahead* 1*-step*
1	*Proponent*	*Smallest*	*Patient*	*Smallest body*
2	*Opponent*	*Smallest*	*Patient*	*Smallest body*
3	*Proponent*	*Smallest*	*Eager*	*Smallest body*
4	*Opponent*	*Smallest*	*Eager*	*Smallest body*

Table 7
Strategies for `abagraph`. Values for parameters *Sentence* and *Rule* are the same for both players.

4. Evaluation

Experimental Setup. In our experiments we aimed to evaluate flexible dispute variants and strategies in their exact and approximate versions, comparing also our system `flexABle` to the current best system computing dispute derivations `abagraph` [2][8]. We concentrate on acceptance w.r.t. the admissible (hence also complete and preferred) semantics for which we can compare to `abagraph`, but also report on results on exact disputes finding complete and stable assumptions congruous with goals.

The strategies (following Section 3) we fixed are detailed in Table 6. We tried to keep strategies as diverse as possible, while varying in few parameters to ease comparison. The main distinction is between "patient strategies" S_1 prioritizing opponents moves, and "eager strategies" S_2 prioritizing the proponent. Within these, variants S_{xa} and S_x ($x \in \{1,2\}$) differ in their use of $H =$ `leastRules` vs. $H =$ `mostRules` and $A = \top$ vs. $A = \bot$. Variants S_{xb} serves to test the value of conservative forward moves, giving `PF1` moves least priority. Strategy S_{AG} simulates the default strategy of `abagraph`.

Using the same criteria as when selecting strategies of `flexABle`, we also settled on the strategies for `abagraph` depicted in Table 7[9]. Here parameter *Turn* indicates player priority. Setting *OppSet* to smallest roughly means arguments of the opponent which have the fewest yet unexpanded statements are expanded first. *Sentence* denotes which sentences are chosen to expand by, with *Patient* meaning non-assumptions and *Eager* assumptions. *Rule* offers choices for backward expansion from proponent sentences: *Look-ahead* 1*-step* and selecting a rule with

[8]http://robertcraven.org/proarg/abagraph.html; last accessed on 1.7.22.

[9]The parameters used to define strategies in `abagraph` are not directly comparable to the parameters from Section 3. To the best of our knowledge there has also been no in-depth study of strategies of `abagraph`, hence the need to define our own.

the smallest body. We refer to [2] and the `abagraph` webpage for further description.

As to benchmark sets, we follow previous experiments for ABA [3] in using the 680 randomly generated ABA frameworks used first in [2][10]. For each framework, we consider 10 different (single) goal claims, thus having 6800 instances in total[11]. All experiments have been performed in parallel on a cluster operating on Red Hat Enterprise Linux Server version 7.9, kernel version 3.10.0-1127.19.1.el7.x86_64. Each task has been allocated a 2.5Ghz Intel Xeon E5-2680 v3 CPU with 16GB RAM and given a 600-second timeout for admissible semantics tasks and a 1200-second timeout for the other semantics. For `flexABle` we used version 1.0 and we downloaded `abagraph` on 3.3.22 (there is no version information) and ran it on SICStus Prolog version 4.7.1

Results for Admissible Semantics. Table 8 presents results for the admissible semantics with our system `flexABle`. When setups are compared on advancement type, DABF is the clear winner. For each setup, the DABF variants have significantly less time-outs and least total running times. The exception are setups with S_{1b} and S_{2b} strategies, where `PF1` moves are essentially switched-off. The overall best performing dispute variants are those making use of DABF + S_2, having no time-outs, least total solving time (3.06h with DFS), as well as least mean (1.62s with DFS) solving time. For advancement types for which strategies $S_{xb} \neq S_x$ for $x \in \{1,2\}$ (DABF, DC, and DS) setups with S_{xb} clearly perform worse than with S_x, this being a further indication of the benefit of `PF1` moves.

As to the role of strategies, for setups with DAB, those making use of S_2 outperform the others. For setups with DC and DS on the other hand, those making use of S_2 perform worse. The latter further highlights that our experiments do not show random branching on `PF2` moves to be an advantage (over the use of `PF1`) in terms of efficiency.

About the effects of strategies, furtheron, for all setups, S_{xa} (with $H = \texttt{mostRules}$ and $A = \bot$) performs worse than S_x for $x \in \{1,2\}$, but especially with DAB. Also for DAB setups, those using S_{AG} are among the worst performing, while for setups with other advancement types those making use of S_{AG} have performance comparable to the best.

DAB setups with BFS have slightly better performance than DFS setups, particularly with strategies S_{1a}, S_{2a}, and S_{AG}. Our experiments suggest BFS to be avoided when non-conservative moves are often used (setups with DC and DS + strategies S_2, S_{2a} and S_{2b}).

Comparison with abagraph. Table 9 shows our results for the admissible semantics with `abagraph` strategies. Figure 1 (x-axis indicates solving-time in seconds and y-axis number of instances solved) shows best and worst variants w.r.t. timeouts for: `flexABle` with DAB, `flexABle` with DABF, and `abagraph`. We also included the directly comparable strategies: S_{AG} for `flexABle` and the default strategy for `abagraph`.

Clearly, `flexABle` with DABF significantly outperforms `abagraph` on any count. When comparing `flexABle` with DAB and `abagraph` on the other hand several of the strategies of `abagraph` have less time-outs, yet DAB setups with strategy S_2 still clearly outperforms all

[10]http://robertcraven.org/proarg/experiments.html Last accessed on 1.7.22.

[11]We do not filter-out trivial instances, either derivable from empty assumptions or not derivable at all, as in other work [3]. Results filtering-out such instances are comparable and are found in the appendix of [9].

Str.		DAB		DABF		DC		DS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	193	179	0	2	13	13	36	39
	total time [h]	38.39	35.70	3.73	4.05	7.37	7.84	11.84	12.56
	95% time [h]	2.79	2.83	2.67	2.77	2.96	3.27	2.73	2.84
	mean [s]	3.39	3.18	1.97	1.97	2.76	3.01	3.11	3.22
S_{1a}	#timeout.	868	369	9	11	17	22	82	76
	total time [h]	153.97	68.50	6.37	6.32	9.29	9.25	21.24	20.52
	95% time [h]	97.29	11.80	3.07	3.07	2.96	3.06	2.77	2.94
	mean [s]	5.63	3.90	2.58	2.38	3.42	2.97	4.06	4.20
S_{1b}	#timeout.	193	179	195	180	272	236	300	257
	total time [h]	38.24	35.79	38.55	35.69	52.23	45.33	56.28	48.84
	95% time [h]	2.67	2.92	2.79	2.72	3.50	3.08	3.52	3.32
	mean [s]	3.31	3.23	3.30	3.08	3.80	3.28	3.47	3.29
S_2	#timeout.	133	115	0	0	180	260	167	240
	total time [h]	28.64	26.08	3.06	3.12	40.82	56.41	39.12	53.04
	95% time [h]	2.98	3.00	2.68	2.73	3.92	6.48	3.77	5.46
	mean [s]	3.49	3.72	1.62	1.65	5.88	7.19	6.12	7.15
S_{2a}	#timeout.	760	337	1	4	181	262	173	252
	total time [h]	136.43	61.98	4.46	4.88	41.10	56.48	39.80	54.46
	95% time [h]	79.75	5.36	2.73	2.88	3.93	6.46	3.68	5.84
	mean [s]	5.80	3.22	2.27	2.23	5.95	7.04	5.96	6.84
S_{2b}	#timeout.	133	115	133	115	416	838	416	800
	total time [h]	28.55	26.16	28.61	25.84	81.07	166.78	80.48	159.82
	95% time [h]	2.95	3.05	2.93	2.65	24.39	110.09	23.80	103.13
	mean [s]	3.44	3.76	3.47	3.59	6.61	16.34	6.28	15.86
S_{AG}	#timeout.	566	331	1	2	11	15	54	59
	total time [h]	102.05	62.26	4.51	4.65	7.54	7.49	15.16	15.33
	95% time [h]	45.37	5.91	2.95	2.95	3.20	2.67	2.89	2.80
	mean [s]	4.45	3.93	2.30	2.29	3.03	2.65	3.29	2.93

Table 8
Results for the admissible semantics using `flexABle`. Column "Str." is for the strategies as in Table 6 with advancement types (DAB,DABF,DC,DS) and search type (depth vs. breadth-first); all use TA for termination. We give the total time and the time to solve 95% of the instances. Timeouts are not considered in the mean running time. For each parameter, best results among all setups are highlighted using green, worst with red.

	abagraph strategy				
	Def.	1	2	3	4
#timeout.	142	139	258	139	764
total time [h]	62.53	76.73	120.86	80.52	256.42
95% time [h]	34.41	48.72	74.27	52.70	199.57
mean [s]	21.00	28.94	42.82	30.99	76.84

Table 9
Results for the admissible semantics using `abagraph`. Strategies as in Table 7.

`abagraph` strategies in terms of both timeouts and total solving time (115 and 26.08h in BFS variant). Also, as is most clearly seen in Figure 1, `flexABle` setups also with DAB solve instances faster on average than `abagraph`.

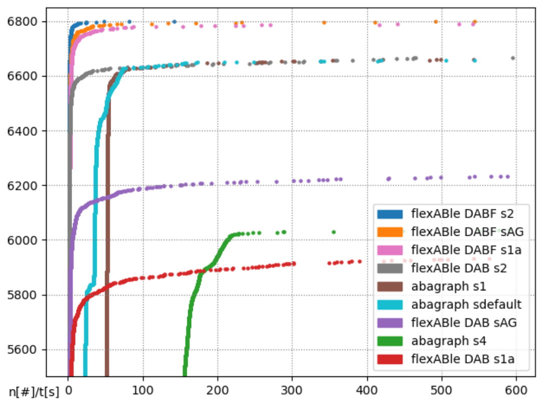

Figure 1: Inverted "cactus plot" comparing the performance of `flexABle` and `abagraph`.

			p_{prop}					
			0.5	0.6	0.7	0.8	0.9	1
p_{opp}	0.5	acc.	0.925					0.977
		spec.	0.975	–	–	–	–	0.97
		sens.	0.733					1.0
	0.6	acc.		0.943				0.982
		spec.	–	0.981	–	–	–	0.978
		sens.		0.796				1.0
	0.7	acc.			0.959			0.988
		spec.	–	–	0.986	–	–	0.984
		sens.			0.854			1.0
	0.8	acc.				0.973		0.992
		spec.	–	–	–	0.991	–	0.99
		sens.				0.907		1.0
	0.9	acc.					0.987	0.996
		spec.	–	–	–	–	0.996	0.995
		sens.					0.955	1.0
	1	acc.	0.939	0.954	0.967	0.98	0.99	
		spec.	1.0	1.0	1.0	1.0	1.0	–
		sens.	0.702	0.773	0.84	0.901	0.953	

			p_{prop}					
			0.5	0.6	0.7	0.8	0.9	1
p_{opp}	0.5	acc.	0.926					0.976
		spec.	0.974	–	–	–	–	0.97
		sens.	0.741					1.0
	0.6	acc.		0.944				0.982
		spec.	—	0.98	–	–	–	0.978
		sens.		0.804				1.0
	0.7	acc.			0.961			0.987
		spec.	–	–	0.986	–	–	0.984
		sens.			0.865			1.0
	0.8	acc.				0.975		0.992
		spec.	–	–	–	0.991	–	0.99
		sens.				0.918		1.0
	0.9	acc.					0.988	0.996
		spec.	–	–	–	–	0.995	0.995
		sens.					0.962	1.0
	1	acc.	0.945	0.959	0.971	0.983	0.992	
		spec.	1.0	1.0	1.0	1.0	1.0	–
		sens.	0.732	0.802	0.862	0.916	0.962	

Table 10
Accuracy, specificity and sensitivity for both static (left) and dynamic (right) sampling. Greyed-out fields indicate combinations which we did not perform experiments for.

Approximate Reasoning. Table 10 shows surprisingly good results on accuracy, specificity, and sensitivity for approximate disputes for the combinations of p_{prop} and p_{opp} values we considered. Figure 2 compares the rather representative run-time performance of static and dynamic bi-directional sampling for all considered values of p, as well as exact reasoning for the worst- and best performing strategies for advancement types DAB and DABF and search type DFS. In general, we found little-to-no improvement in performance for DABF setups, yet some for DAB setups, especially using dynamic sampling.

Results for Complete and Stable Semantics. For DC + TC and DS + TS disputes we compared with an approach in which admissible assumptions congruous with the goal claim are searched for first and then extended to complete and stable sets. Results are in Table 11 and show

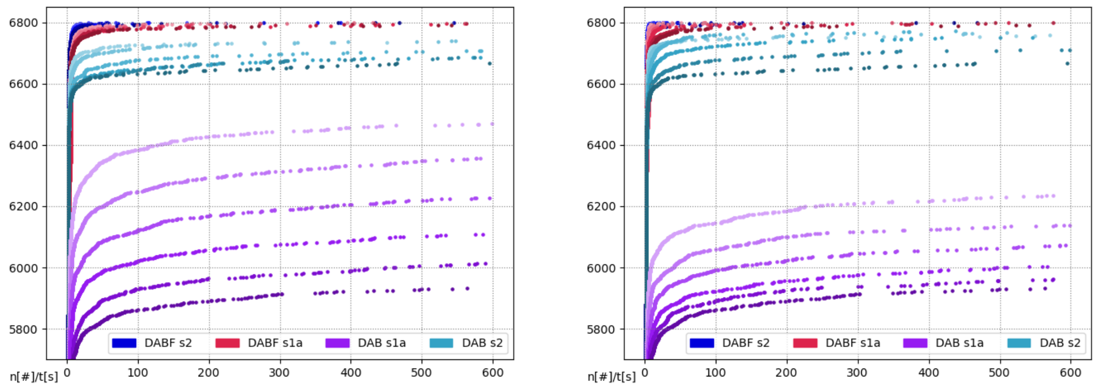

Figure 2: Inverted cactus plots showing time vs. instances solved of static (left) and dynamic (right) sampling for approximate reasoning using $p_{prop} = p_{opp} = p$, $p \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ and exact reasoning ($p = 1.0$) for the best (S_2) and worst (S_{1a}) performing strategies for DAB, DABF and DFS. Lines with same colors indicate same advancement type and strategy, with the darkest shades indicating exact reasoning ($p = 1.0$). The lighter the shade, the smaller the value of the p parameter, with the lightest shade indicating $p = 0.5$.

Str.		**complete** semantics				**stable** semantics			
		Start w/ DABF + TA		Start w/ DC + TC		Start w/ DABF + TA		Start w/ DS + TS	
		DFS	BFS	DFS	BFS	DFS	BFS	DFS	BFS
S_1	#timeout.	20	34	73	101	168	163	526	549
	total time [h]	18.70	20.67	51.84	61.85	71.28	72.12	230.24	237.71
	95% time [h]	3.63	3.38	8.50	8.87	5.32	5.34	116.89	124.34
S_{1a}	#timeout.	26	45	87	120	171	174	533	557
	total time [h]	21.85	25.62	58.06	69.32	74.76	75.38	231.30	241.24
	95% time [h]	3.30	3.63	8.87	10.16	5.76	5.65	117.95	127.87
S_{1b}	#timeout.	275	682	663	1155	892	869		
	total time [h]	110.71	246.23	263.77	420.20	330.54	318.26	–	–
	95% time [h]	8.49	132.85	150.43	306.81	217.19	204.88		
S_2	#timeout.	212	230	887	1004	106	137	377	442
	total time [h]	93.49	99.24	503.27	551.97	54.02	60.29	178.07	195.34
	95% time [h]	7.03	7.50	389.93	438.60	4.68	4.76	64.73	81.97
S_{2a}	#timeout.	211	233	895	1022	104	133	382	443
	total time [h]	96.35	102.98	516.70	559.69	55.00	61.52	178.67	195.20
	95% time [h]	8.12	9.65	403.36	446.33	4.67	4.96	65.32	81.84
S_{2b}	#timeout.	853	931			891	896		
	total time [h]	328.05	350.38	–	–	337.54	332.56	–	–
	95% time [h]	214.70	236.99			224.19	219.18		
S_{AG}	#timeout.	21	34	93	125	175	176	557	580
	total time [h]	19.97	21.69	62.92	71.38	76.14	76.54	250.66	259.80
	95% time [h]	3.71	3.39	10.69	10.35	6.01	5.92	137.31	146.43

Table 11
Results for the complete and stable semantics using `flexABle`. Dark red cells containing "–" indicate setups which exceeded the total available running time.

that starting with DABF + TA provides better results. The patient strategy S_1 performs better than S_2 for the complete semantics, the opposite for stable.

5. Conclusions

Strategies have been defined for ABA disputes [1, 2], but there has only been limited empirical investigation. We had several interesting findings about strategies in the context of flexible disputes, central among which is that (conservative) forward moves lead to a significant performance boost also when comparing our direct implementation `flexABle` of flexible disputes to the Prolog-based `abagraph`, implementing graph-based disputes. Note that we did not compare to the still more efficient reduction-to-answer-set-programming approach of [3], since this approach does not compute dialectical justifications and cannot readily be incorporated into interactive argument-based reasoning. In future work, among several things, we want to extend the experiments to include more and larger benchmarks as well as consider other semantics.

Acknowledgments

This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 389792660 – TRR 248, and by the Bundesministerium für Bildung und Forschung (BMBF) Förderkennzeichen 01IS20056_NAVAS.

References

[1] F. Toni, A generalised framework for dispute derivations in assumption-based argumentation, Artif. Intell. 195 (2013) 1–43.

[2] R. Craven, F. Toni, Argument graphs and assumption-based argumentation, Artif. Intell. 233 (2016) 1–59.

[3] T. Lehtonen, J. P. Wallner, M. Järvisalo, Declarative algorithms and complexity results for assumption-based argumentation, J. Artif. Intell. Res. 71 (2021) 265–318.

[4] F. Toni, A tutorial on assumption-based argumentation, Argument Comput. 5 (2014) 89–117.

[5] M. Caminada, Argumentation semantics as formal discussion, in: P. Baroni, D. Gabbay, M. Giacomin (Eds.), Handbook of Formal Argumentation, College Publications, 2018, pp. 487–518.

[6] M. Diller, S. A. Gaggl, P. Gorczyca, Flexible dispute derivations with forward and backward arguments for assumption-based argumentation, in: CLAR, volume 13040 of *LNCS*, Springer, 2021, pp. 147–168.

[7] M. Thimm, T. Rienstra, Approximate reasoning with ASPIC+ by argument sampling, in: SAFA@COMMA, volume 2672 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 22–33.

[8] C. Sun, Approximate Inference for Assumption-based Argumentation in AI, Master's thesis, Universität Koblenz - Landau, Koblenz, 2021. URL: https://hbz.opus.hbz-nrw.de/opus45-kola/frontdoor/deliver/index/docId/2183/file/ChuyiSun_thesis.pdf.

[9] P. Gorczyca, Automatic and Interactive Search in Flexible Dispute Derivations for Assumption-Based Argumentation: Analysis, Implementation, Evaluation., Master's thesis, TU Dresden, Dresden, 2022. URL: https://iccl.inf.tu-dresden.de/w/images/e/e6/Gorczyca_MScThesis_Signed.pdf.