Answer Set Programming for Judgment Aggregation

Ronald de Haan[1] and **Marija Slavkovik**[2]
[1]Institute for Logic, Language and Computation (ILLC), University of Amsterdam
[2]University of Bergen
me@ronalddehaan.eu, Marija.Slavkovik@uib.no

Abstract

Judgment aggregation (JA) studies how to aggregate truth valuations on logically related issues. Computing the outcome of aggregation procedures is notoriously computationally hard, which is the likely reason that no implementation of them exists as of yet. However, even hard problems sometimes need to be solved. The worst-case computational complexity of answer set programming (ASP) matches that of most problems in judgment aggregation. We take advantage of this and propose a natural and modular encoding of various judgment aggregation procedures and related problems in JA into ASP. With these encodings, we achieve two results: (1) paving the way towards constructing a wide range of new benchmark instances (from JA) for answer set solving algorithms; and (2) providing an automated tool for researchers in the area of judgment aggregation.

1 Introduction

Judgment aggregation (JA) is a general framework for modeling the aggregation of individual opinions over a set of—typically logically related—issues into one collective value judgment on these issues [Grossi and Pigozzi, 2014; Lang *et al.*, 2017]. This framework can be used to express a wide range of aggregation scenarios, including preference aggregation and voting [Endriss, 2018; Lang and Slavkovik, 2013], graph aggregation problems [Endriss and Grandi, 2017], and various collective decision making problems in multi-agent systems [Slavkovik, 2016].

A wide range of aggregation procedures has been studied in the literature, both from an axiomatic and an algorithmic point of view. These aggregation procedures essentially all have in common that computing collective judgments is computationally hard—generally Θ_2^p- or Σ_2^p-complete [Endriss *et al.*, 2012; Jamroga and Slavkovik, 2013; Lang and Slavkovik, 2014; Endriss and de Haan, 2015; De Haan and Slavkovik, 2017]. This high worst-case complexity poses a significant barrier for finding practically efficient implementations, and indeed, as of yet, no structured implementation of judgment aggregation methods is available.

However, various optimized algorithms exist for problems with high worst-case complexity that work extremely well in practice in many cases—including Boolean satisfiability solvers [see, e.g., Biere *et al.* 2009] and answer set solvers [see, e.g., Gebser *et al.* 2018]. Answer set solvers should be of particular interest for judgment aggregation, since the underlying language (answer set programming) is expressive enough to encode problems that are Θ_2^p- and Σ_2^p-complete. In this paper, we provide an encoding of JA problems into answer set programming. The approach of solving JA problems in their full generality—by means of encodings into a powerful solving framework as ASP—is complementary to the investigation of efficient JA fragments [De Haan, 2016; De Haan, 2018]. A combination of these two approaches is needed to make JA methods available for practical use in a wide range of settings.

1.1 Contributions

We present a general and modular encoding into answer set programming for (i) computing the outcome of judgment aggregation procedures and (ii) checking agenda and profile properties in JA.

Our encodings are general, because they can be directly used in combination with modern answer set solvers and because they allow for several aggregation problems to be implemented by representing them as judgment aggregation problems. Judgment aggregation generalizes voting, preference aggregation [Lang *et al.*, 2017], graph aggregation [Endriss and Grandi, 2017], etc. A small number of answer set implementations exist for specific voting rules [Konczak, 2006; Charwat and Pfandler, 2015], but they do not generalize to other aggregation problems. To the best of our knowledge, this constitutes the first implementation of JA methods.

Our encodings are modular because they allow for different parts of the judgment aggregation problem to be implemented as separate program components that can be reused and combined as needed. Our work serves as an essential starting point for efficient implementations of the framework of judgment aggregation. As such, our work establishes a baseline for experimental evaluations of future implementations of judgment aggregation methods.

We aim to achieve the following goals with our encodings. (A) By encoding a wide range of JA problems into ASP instances, we provide recipes for generating new benchmark instances for ASP solving algorithms—for problems that are Θ_2^p- and Σ_2^p-complete. These benchmark instances can be generated

from real-world data—for example, by encoding election data from the PrefLib data set[1] into JA, and subsequently using the encodings in this paper, a multitude of new ASP benchmarks can be constructed. Moreover, (B) our encodings serve as a convenient tool for researchers in judgment aggregation, and we hope and expect that they will help bring interesting judgment aggregation problems to the attention of the automated reasoning community. The efforts of this community are essential for finding optimisation solutions for judgment aggregation problems. A third objective of our work is to (C) support theoretical research in judgment aggregation: by encoding various JA problems into one unified representation and solving framework, we reveal properties of procedures and problems that are otherwise not easily visible.

All encodings presented in this paper are available as online supplementary material: https://github.com/rdehaan/ja-asp.[2] For space reasons, we present some results without proof—these results are indicated with a ⋆.

2 Preliminaries

We introduce the framework of judgment aggregation that we use in this paper. Note that several (interchangeable) variations of judgment frameworks are in use in the literature [Endriss *et al.*, 2016]—in this paper, we use the variant that is most convenient for our purposes. We also briefly introduce the syntax and semantics of answer set programming.

2.1 Judgment Aggregation

A judgment aggregation problem consists of a finite set Φ of propositions called the *agenda*, a propositional formula Γ over the variables from Φ called the *constraint*, and a *profile P (of judgments)*—defined as follows. Let $\mathcal{L}_a$ be a set of atomic propositions. A *pre-agenda* $[\Phi]$ is a finite set of issues $\varphi \in \mathcal{L}_a$ that has $\Phi = \{\varphi, \neg\varphi : \varphi \in [\Phi]\}$ as corresponding agenda. A literal is either an atom $\varphi \in \Phi$ or its negation $\neg\varphi$. We assume that Γ is a formula given in conjunctive normal form (CNF). A profile of judgments $P = (J_1, \ldots, J_n)$ is a sequence of *judgment sets* J_i each representing one voter (or opinion source). A judgment set is a subset of the agenda $J \subseteq \Phi$. It is typically required that each judgment set is *consistent* and *complete*. A judgment set J is complete if for each issue $\varphi \in [\Phi]$ either $\varphi \in J$ or $\neg\varphi \in J$. A judgment set J is consistent if $J \cup \{\Gamma\}$ is a consistent set of formulas. We let $\mathcal{J}(\Phi, \Gamma)$ denote the set of all complete and consistent judgment sets. A JA procedure selects as outcomes for a given profile a nonempty set of complete and consistent judgments sets. We use the following running example—borrowed from Lang *et al.* (2017).

Example 1. *Let* $\Phi = \{i_1, i_2, i_3, i_4, i_5, \neg i_1, \neg i_2, \neg i_3, \neg i_4, \neg i_5\}$ *and* $\Gamma = \{(i_3 \vee \neg i_4) \wedge (\neg i_1 \vee \neg i_3 \vee i_4) \wedge (\neg i_2 \vee \neg i_3 \vee i_4)\}$. *An example profile P is depicted in Table 1.*

P	i_1	$\neg i_1$	i_2	$\neg i_2$	i_3	$\neg i_3$	i_4	$\neg i_4$	i_5	$\neg i_5$
$J_1 - J_6$	+	−	+	−	+	−	+	−	+	−
$J_7 - J_{10}$	+	−	+	−	−	+	−	+	+	−
$J_{11} - J_{17}$	−	+	−	+	+	−	−	+	−	+
Majority	i_1		i_2		i_3		$\neg i_4$		i_5	
Kemeny score sum	10	7	10	7	13	4	6	11	10	7
Reversal score sum	10	14	10	14	19	8	12	15	10	7

Table 1: Profile P from Example 1.

2.2 Answer Set Programming

We firstly define the core syntax and semantics of answer set programming (ASP)—before discussing various extensions to the syntax and semantics. A *disjunctive logic program* Π is a finite set of rules of the form $h_1 \vee \cdots \vee h_k$:– $b_1, \ldots, b_l$, not $b_{l+1}, \ldots,$ not b_t, where $h_1, \ldots, h_k, b_1, \ldots, b_t$ are propositional atoms. A rule is a fact if $t = 0$, and a constraint if $k = 0$. A model M of the program Π is a truth assignment to the atoms occurring in Π that satisfies the rules of Π (when seen as propositional logic statements). For the sake of convenience, we often equate a model M with the set of atoms that it sets to true. We are interested in a particular subset of models—called answer sets. Given a model M of Π, the *reduct* Π^M of Π w.r.t. M is obtained by removing from Π every rule where $b_j \in M$ for some $1 \leq j \leq t$, and removing all literals not b_j from the remaining rules. A model M is an *answer set* of Π if is a subset-minimal model of Π^M.

To increase the convenience and expressivity of ASP, various extensions of the basic syntax and semantics have been considered. We briefly discuss the variant of the ASP language that we use in this paper, using which extensions[3]—for more details, we refer to other resources, e.g., [Calimeri *et al.*, 2013; Gebser *et al.*, 2017]. We use ; to express disjunctions. We use a variant of the syntax that uses variables—rules with variables represent the set of ground instances of these rules. We use - as a built-in unary predicate (we use no special interpretation for this predicate in this paper). We often use shorthand to succinctly write a range of atoms—e.g., `a(1..3)` stands for `a(1)`, `a(2)` and `a(3)`; and `b(2;4)` stands for `b(2)` and `b(4)`. The language that we use contains built-in arithmetic operations—such as = and <. It also contains arithmetic aggregates, such as `#sum`, `#max`, and `#count`—these are followed by a set of atoms, and evaluate to an integer; e.g., `#count {a,b,c}` evaluates to 3. We use conditional statements—e.g., `c(X):d(X)` represents the conjunction of `c(X)` for all `X` for which `d(X)` holds. We use choice rules—e.g., `1 {a,b,c} 2` expresses that at least one and at most two atoms among `a`, `b` and `c` must be true. Finally, we use optimization statements that select a subset of answer sets that minimize or maximize a weighted sum of atoms over different (lexicographical) priority levels—e.g., `#minimize {A@L,e(A):e(A)}` expresses minimization of atoms `e(A)` weighing `A`, at priority level `L`.

3 General Encoding of the Setting of JA

We present how to encode some general aspects of JA into ASP—the encodings in this section are built upon in further sections. Individuals are declared with `voter(i)` facts, and issues with `issue(x)` facts. Moreover, the clauses of the in-

[1] http://www.preflib.org/data

[2] The online supplementary material contains more than what we present in this paper. In Section 7, we indicate some ways in which the supplementary material goes beyond the encodings in the paper.

[3] The variant of the language that we use is that used by the *Potsdam Answer Set Solving Collection* [Gebser *et al.*, 2011b; Gebser *et al.*, 2017]. This language is a superset of the ASP-Core-2.0 standard [Calimeri *et al.*, 2013].

tegrity constraint are encoded using the predicate `clause/2`.[4] For instance, the 5 issues and 17 voters in Example 1 can be declared by `voter(1..17)` and `issue(i1;i2;i3;i4;i5)`, and the constraint can be encoded by `clause(1,(i3;-i4))`, `clause(2,(-i3;i4;-i1))` and `clause(3,(-i3;i4;-i2))`.

We then use the following rules to encode that each voter must be associated with a judgment set that is consistent with the integrity constraint—represented by the predicate `js/2`.

```
1 agent(A) :- voter(A).
2 lit(X;-X) :- issue(X).
3 1 { js(A,X) ; js(A,-X) } 1 :- agent(A),
      issue(X).
4 :- agent(A), clause(C,_), js(A,-L) :
      clause(C,L).
```

Profiles can then be encoded using the predicate `js/2`. For example, the first judgment sets J_1–J_6 in Example 1 are encoded with the facts `js(1..6,(i1;i2;i3;i4;i5))`. We use the predicate `agent/1` for voters and for the collective outcome.

Lemma 1. *Let S be an answer set of a program that contains (i) an encoding of the issues using `issue/1`, (ii) an encoding of an integrity constraint Γ using `clause/2`, and (iii) lines 1–4. If `agent(X)` $\in S$ for some `X`, then the set of atoms `L` for which `js(X,L)` $\in S$ corresponds to a complete and consistent judgment set J. Moreover, without further rules and facts, this is a one-to-one correspondence.*

Proof (sketch). Line 3 (with the `issue/1` facts) ensures that J is a complete judgment set. Line 4 (with the `clause/2` facts) ensures that J is consistent with Γ. Moreover, the rules in these lines can be satisfied by a set of atoms corresponding to any complete and consistent judgment set J. □

4 Encoding Judgment Aggregation Procedures

In order to generate a collective outcome, all we need to do is to declare an agent representing the collective opinion with the fact `agent(col)`. We can then encode various JA procedures from the literature as follows. In all cases, the answer sets for the encodings represent the outcomes of the JA procedures.

That is, to compute the outcome of a judgment aggregation procedure for a given profile, one constructs the logic program consisting of (i) the basic encoding in lines 1–4, (ii) an encoding of the agenda and profile using predicates `voter/1`, `issue/1`, `clause/2`, and `js/2`, (iii) the declaration of a collective agent with the fact `agent(col)`, and (iv) the encoding of the desired JA procedure—as specified below. The (optimized) answer sets of the resulting logic program then specify the outcomes of the JA procedure for the given profile.

Throughout the remainder of this section, we will define the different JA procedures that we discuss. For more details, we refer to the literature (e.g., Lang *et al.* 2017). The encodings in this section are specific for the problem of computing a collective outcome for particular JA procedures.

4.1 Scoring Procedures

We begin with several JA procedures that are based on a notion of *score*. A score $s : \mathcal{J}(\Phi,\Gamma) \times \Phi \to \mathbb{R}^+$ is a function that assigns a nonnegative value for a judgment in Φ with respect to some J. The *scoring procedure* F_s for the score s is defined as $F_s(P) = \arg\max_{J \in \mathcal{J}(\Phi,\Gamma)} \sum_{J_i \in P} \sum_{\varphi \in J} s(J_i, \varphi)$.

[4] We use `pred`/x to indicate that the predicate `pred` has arity x.

4.2 Kemeny

The *Kemeny procedure* Kem is based on the Kemeny score s_K where $s_K(J,\varphi) = 1$ if $\varphi \in J$ and $s_K(J,\varphi) = 0$ otherwise. For the profile P in Example 1, for instance, Kemeny selects $\{i_1, i_2, i_3, i_4, i_5\}$.

We can encode the Kemeny procedure by maximizing agreement with the weighted majority.

```
5 wgt(X,N) :- lit(X), N = #count
      { A : voter(A), js(A,X) }.
6 #maximize { N@1,wgt(X,N) : wgt(X,N),
      js(col,X) }.
```

Theorem 1. *The optimal answer sets S for the ASP encoding of the Kemeny procedure applied to a profile P are in one-to-one correspondence with* Kem(P).

Proof (sketch). Take an arbitrary optimal answer set S. By Lemma 1, and since `agent(col)` $\in S$, we know that `{L : js(col,L)}` corresponds to a complete and consistent judgment set J^*. Lines 5–6 ensure that optimal answer sets maximize the cumulative Kemeny score s_K—thus, $J^* \in$ Kem(P). Conversely, each $J^* \in$ Kem(P) can be translated into an optimal answer set S. □

4.3 Leximax

The *Leximax procedure* Lex can be defined using a score as follows—it is more commonly defined in a procedural way (see, e.g., Lang *et al.* 2017, Definition 4). The Leximax procedure is based on the (leximax) score s_L where $s_L(P,\varphi) = (n+m)^u$, where n is the number of judgment sets in the profile, m is the number of issues in the agenda, and u is the number of judgment sets in P that contain φ, i.e., the Kemeny score. For the profile P in Example 1, for instance, the Leximax procedure selects $\{\neg i_1, \neg i_2, i_3, \neg i_4, i_5\}$.

We can encode the Leximax procedure as follows. We reuse the predicate `wgt/2` from line 5 to do a lexicographical maximization.

```
7 #maximize { 1@N,wgt(X,N) : wgt(X,N),
      js(col,X) }.
```

The above two encodings highlight an elegant symmetry between the Leximax procedure and the Kemeny procedure—as can be seen in the difference between lines 6 and 7.

Theorem 2. *The optimal answer sets S for the ASP encoding of the Leximax procedure applied to a profile P are in one-to-one correspondence with* Lex(P).

Proof (sketch). The proof is entirely analogous to the proof of Theorem 1—with the prioritized maximization statement in line 7 corresponding to maximizing the leximax score s_L. □

4.4 Reversal Scoring

The *reversal scoring procedure* Rev is the scoring procedure based on the reversal score s_R, that is defined by $s_R(J,\varphi) = \min_{J' \in \mathcal{J}(\Phi,\Gamma), \varphi \notin J'} d_H(J,J')$, where $d_H(J,J') = |J \setminus J'| = |J' \setminus J|$ denotes the Hamming distance between J and J'. For the

profile P in Example 1, for instance, reversal scoring selects $\{\neg i_1, \neg i_2, i_3, \neg i_4, i_5\}$.

We can encode the reversal scoring procedure as follows. Firstly, we introduce a virtual agent `vrt(A,X)` for each voter `A` and each issue `X`, and ensure that `vrt(A,X)` does not have `X` in its judgment set.

```
8 agent(vrt(A,X)) :- voter(A), lit(X).
9 js(vrt(A,X),-X) :- voter(A), lit(X),
      js(A,X).
```

Then, we do prioritized optimization, where firstly we minimize the difference between the virtual agents' judgment sets and the corresponding judgment sets in the profile.

```
10 disagree(A,X,Y) :- voter(A), lit(X),
       lit(Y), js(A,Y), js(vrt(A,X),-Y).
11 disagreement(A,X,D) :- voter(A), lit(X),
       D = #count { Y : disagree(A,X,Y) }.
12 #minimize { D@2,disagreemt(A,X,D) :
       disagreement(A,X,D) }.
```

Then, at a lower priority level, we maximize the score of the collective judgment set.

```
13 score(A,X,D) :- js(col,X),
       disagreement(A,X,D).
14 score(E) :- E = #sum { D,score(A,X,D) :
       score(A,X,D) }.
15 #maximize { E@1,score(E) : score(E) }.
```

This encoding is in line with the algorithm for computing outcomes of the reversal scoring procedure given by De Haan and Slavkovik (2017).

Theorem 3. *The optimal answer sets S for the ASP encoding of the Reversal scoring procedure applied to a profile P are in one-to-one correspondence with* Rev(P).

Proof (sketch). In general lines, the proof is analogous to the proof of Theorem 1. Lines 10–12 contain a higher priority optimization to determine the values of $s_{\mathrm{R}}(J_i, \varphi)$ for each $J_i \in P$ and each $\varphi \in \Phi$. Then, lines 13-15 contain (lower priority) optimization statements for score maximization. □

4.5 Other Procedures

We continue with other procedures—including procedures based on minimal changes to the profile and procedures based on maximizing agreement with the (ranked) majority outcome.

4.6 Young

The *Young procedure* Yng selects those complete judgment sets J^* that are consistent with the majority outcome of those profiles P' obtained from P by deleting a minimum number of voters to make the majority outcome consistent. For the profile P in Example 1, for instance, the Young procedure selects as outcomes the judgment sets $\{\neg i_1, \neg i_2, i_3, \neg i_4, i_5\}$ and $\{\neg i_1, \neg i_2, i_3, \neg i_4, \neg i_5\}$.

We can encode the Young procedure as follows. Firstly, we guess a subset of voters in the profile.

```
16 in(A) ; out(A) :- voter(A).
```

We ensure that the outcome agrees with the majority outcome (for the guessed subset of voters).

```
17 inwgt(X,N) :- lit(X), N = #count
       { A : voter(A), in(A), js(A,X) }.
18 inmaj(X) :- lit(X), inwgt(X,N),
       inwgt(-X,M), N > M.
19 js(col,X) :- inmaj(X).
```

We then minimize the number of removed voters.

```
20 #minimize { 1@1,out(A) : out(A) }.
```

The explanations accompanying the above rules straightforwardly lead to the following correctness result.

Theorem⋆ 4. *The optimal answer sets S for the ASP encoding of the Young procedure applied to a profile P are in one-to-one correspondence with* Yng(P).

4.7 MSA

The *MSA procedure* msa (for maximum subset of the agenda) selects the complete and consistent judgment sets J^* that agree with a subset-maximal consistent subset of the majority outcome. For the profile P in Example 1, for instance, MSA selects $\{i_1, i_2, i_3, i_4, i_5\}$, $\{i_1, i_2, \neg i_3, \neg i_4, i_5\}$ and $\{\neg i_1, \neg i_2, i_3, \neg i_4, i_5\}$.

We can encode the MSA procedure as follows. Here we reuse the predicate `wgt/2` from line 5. The language of answer set programming does not offer native commands to express optimization w.r.t. set-inclusion. Therefore, we use the meta-programming technique of Gebser *et al.* (2011a), that provides a natural encoding of inclusion-based minimization (we refer to their work for further details).

```
21 maj(X) :- lit(X), wgt(X,N), wgt(-X,M),
       N > M.
22 majdisagree(X) :- lit(X), maj(X),
       js(col,-X).
23 _criteria(0,1,X) :- majdisagree(X).
24 _optimize(0,1,incl).
25 #show _criteria/3. #show _optimize/3.
```

Theorem⋆ 5. *The optimal answer sets S for the ASP encoding of the MSA procedure applied to a profile P are in one-to-one correspondence with* msa(P).

The meta-programming encodings of Gebser *et al.* (2011a) use disjunction in the head of rules—making this a Σ_2^{p}-level ASP encoding, which matches the complexity of the MSA procedure [Lang and Slavkovik, 2014].

4.8 Ranked Agenda

The *ranked agenda procedure* ra is defined as follows. Take a profile P, and let $\varphi_1, \ldots, \varphi_{2m}$ be an enumeration of the formulas in the profile such that for each $1 \leq i < j \leq 2m$ it holds that φ_i is contained in at least as many sets in P as φ_j. Then the sets $S_0, \ldots, S_{2m}$ are defined as follows: $S_0 = \emptyset$, and for each $1 \leq i \leq 2m$ the set $S_i = S_{i-1} \cup \{\varphi_i\}$ if this is consistent with Γ, and $S_i = S_{i-1}$ otherwise. The ranked agenda procedure selects as outcomes all judgment sets J^* for which there is an enumeration of the formulas in Φ such that $J^* = S_{2m}$. For the profile P in Example 1, for instance, the ranked agenda procedure selects $\{\neg i_1, \neg i_2, i_3, \neg i_4, i_5\}$.

We can encode the ranked agenda procedure as follows. Here we reuse the predicate `wgt/2` from line 5, and we use auxiliary facts `litnum(1..2m)`. Firstly, we guess a suitable enumeration of the formulas in Φ.

```
26 1 { order(X,Y) : litnum(Y) } 1 :- lit(X).
27 1 { order(X,Y) : lit(X) } 1 :- litnum(Y).
```

```
28 :- order(X1,Y1), order(X2,Y2), wgt(X1,N1),
      wgt(X2,N2), N1 > N2, Y1 > Y2.
```

We then use the technique of saturation [Eiter and Gottlob, 1995] to verify that the collective outcome is obtained as S_{2m}. We introduce an atom `w(W)` for each $1 \leq \mathtt{W} \leq 2m$ that needs to be derived in each answer set. For each such `W`, we consider all truth assignments—represented using the predicate `vrt/2`—and ensure that `w(W)` is derived for each truth assignment.

```
29 w(W) :- not w(W), litnum(W).
30 vrt(W,X) :- lit(X), w(W).
31 vrt(W,X) ; vrt(W,-X) :- var(X), litnum(W).
32 w(W) :- var(X), vrt(W,X), vrt(W,-X).
```

Intuitively, we selectively eliminate truth assignments (by deriving `w(W)`), and ultimately for every answer set all truth assignments must be eliminated. We firstly eliminate all assignments that falsify Γ.

```
33 w(W) :- clause(C), litnum(W), vrt(W,-L) :
      clause(C,L).
```

We eliminate all assignments for `W` whose corresponding literal `X` is ordered after its negation `-X`.

```
34 w(W) :- order(X,W), order(-X,Y), Y <= W.
```

We eliminate all assignments for `W` for which the corresponding literal `X` can be consistently added (to the collective outcome), and all assignments for `Y < W` that disagree with this literal `X`.

```
35 w(W) :- litnum(W), order(X,W), js(col,X).
36 w(W) :- litnum(W), order(X,Y), Y < W,
      js(col,X), vrt(W,-X).
```

Finally, we eliminate all assignments for `W` that disagree with the corresponding literal `X`.

```
37 w(W) :- litnum(W), order(X,W), vrt(W,-X).
```

If there is a `W` for which some assignment is not eliminated, then, the literal `X` corresponding to `W` can be consistently added (which is witnessed by the non-eliminated assignment). Therefore, every literal `X` that does not agree with the collective outcome must be inconsistent with the partial outcome constructed so far (w.r.t. the guessed order of the agenda) for the model to be an answer set. As a result, the answer sets correspond to the outcomes of the ranked agenda procedure.

Theorem 6. *The optimal answer sets S for the ASP encoding of the Ranked agenda procedure applied to a profile P are in one-to-one correspondence with* $\textsc{ra}(P)$.

Proof (sketch). The encoding in lines 26–37 generally mirrors the definition of the ranked agenda procedure. One aspect of the encoding for which the connection to the definition is not immediately obvious is the use of atoms `w(W)`. These are used to employ the technique of saturation: line 29 ensures that these atoms must be derived in any answer set. Lines 32, 33, 34, 35, 36, and 37 allow such atoms `w(W)` corresponding to a literal `X` to be derived—respectively—(i) if the guessed truth assignment α is inconsistent, (ii) if α falsifies Γ, (iii) if the complementary literal `-X` appears earlier in the guessed ordering, (iv) if α agrees on `X` with the outcome, (v) if α disagrees with a literal in the outcome that appears earlier in the guessed ordering, and (vi) if α falsifies `X`. Then S can only be an answer set if for every `W` and every truth assignment guessed for `W` using `vrt/2`, one of these conditions holds. Using conditions (i)–(vi), one can show that this is the case if and only if S corresponds to some set in $\textsc{ra}(P)$. □

4.9 Further Procedures

The encodings of many other JA procedures are similar. Encoding the majority, quota, Slater, max-Hamming and MPC procedures (see, e.g., Lang *et al.* 2017)—for instance—can be done entirely analogously to the encodings given above. Encodings for many further procedures can be found in the online supplementary material.

5 Encoding Agenda Properties

In this section, we will describe how to encode various agenda properties into answer set programs. The answer sets for these programs will represent witnesses or counterexamples for these properties.

To use the encodings in this section, one constructs the logic program consisting of (i) the basic encoding in lines 1–4, (ii) an encoding of the agenda and integrity constraint using predicates `issue/1` and `clause/2`, (iii) where relevant, an encoding of the profile using `js/2`, and (iv) the encoding of the desired property—as specified below. The encodings in this section work without declaring a 'collective' agent `agent(col)`. The encodings are specific for the problem of deciding the agenda properties that we discuss.

5.1 The k-Median Property

An agenda Φ and constraint Γ satisfy the *k-median property (k-MP)* if every Γ-inconsistent subset of Φ has itself a Γ-inconsistent subset of size at most k. This property is useful for deciding whether particular JA procedures are free from the risk of selecting inconsistent outcomes [Endriss *et al.*, 2012]. For example, if an agenda satisfies the 2-MP, then the majority outcome is always consistent.

We can encode the k-MP as follows. Firstly, we encode the agenda Φ and the constraint Γ using the predicates `issue/1` and `clause/2`, and use the encoding of the basic JA setting from lines 1–4 (in fact, only line 2 is needed in this case). Secondly, we declare the constant `k`, e.g., `#const k=2`. Then we guess a subset of Φ of size $> k$ as follows.

```
38 0 { assign(X); assign(-X) } 1 :- lit(X).
39 k+1 { assign(X) : lit(X) }.
```

We use the technique of saturation [Eiter and Gottlob, 1995] to verify that this subset is inconsistent. We introduce an atom `w` that must be derived for the model to be an answer set, and we ensure that this atom `w` is derived only if there is no truth assignment agreeing with the guessed subset. We do this by considering all truth assignments (to the formulas $\varphi \in \Phi$).

```
40 w :- not w.
41 vrt(X) :- lit(X), w.
42 vrt(X) ; vrt(-X) :- var(X).
43 w :- var(X), virtual(X), virtual(-X).
```

For every assignment that falsifies Γ and for every assignment that does not agree with the guessed subset, we derive `w`.

```
44 w :- clause(C), virtual(-L) : clause(C,L).
45 w :- lit(X), assign(X), virtual(-X).
```

We verify that the guessed subset is minimally inconsistent.

```
46 agent(vrt(X)) :- assign(X), lit(X).
47 js(vrt(Y),X) :- agent(vrt(Y)), lit(X),
       assign(X), Y != X.
```

The agenda Φ then satisfies the k-median property if and only if the constructed logic program has no answer sets. Conversely, any answer set corresponds to an inconsistent subset that witnesses that Φ does not satisfy the k-MP.

Theorem 7. *The answer sets S for the ASP encoding of the k-MP applied to an agenda Φ are in one-to-one correspondence with minimally Γ-inconsistent subset of Φ of size $> k$.*

Proof (idea). In the encoding, we guess a subset $\Phi' \subseteq \Phi$ of size $> k$, we use the technique of saturation to ensure that Φ' is Γ-inconsistent, and we guess satisfying truth assignments for all subsets $\Phi'' \subsetneq \Phi'$ of size $|\Phi'| - 1$ (that satisfy Γ too). □

5.2 Agenda Separability

Let Φ be an agenda, and Γ a constraint. A partition (Φ_1, Φ_2) of Φ consists of two sets $\Phi_1, \Phi_2 \subseteq \Phi$ such that (i) $\Phi_1 \cup \Phi_2 = \Phi$, (ii) $\Phi_1 \cap \Phi_2 = \emptyset$, and (iii) for each $\varphi \in [\Phi]$, if $\varphi \in \Phi_i$ then $\neg\varphi \in \Phi_i$ for both $i \in \{1, 2\}$. A partition (Φ_1, Φ_2) of Φ is an *independent partition* if for all judgment sets $J_1 \in \mathcal{J}(\Phi_1, \Gamma)$ and $J_2 \in \mathcal{J}(\Phi_2, \Gamma)$, $J_1 \cup J_2$ is Γ-consistent. This concept can be used to divide the JA process into two processes—for procedures that respect independent partitions [Lang *et al.*, 2016].

We can encode the problem of finding independent partitions of an agenda Φ as follows. We encode the agenda Φ and the constraint Γ using the predicates `issue/1` and `clause/2`, and use the basic encoding from lines 1–4 (also here only line 2 is needed). Then, we declare the auxiliary facts `side(1;2)` and guess a partition of Φ.

```
48 part(1,X); part(2,X) :- issue(X).
49 :- part(1,X), part(2,X), issue(X).
50 part(S,-X) :- issue(X), part(S,X), side(S).
51 :- side(S), part(S,X) : issue(X).
```

We then use the technique of saturation to verify that this partition is independent. We consider all combinations of two truth assignments (to the formulas $\varphi \in \Phi$), for both parts of the partition.

```
52 w :- not w.
53 vrt(S,X) :- lit(X), w, side(S).
54 vrt(S,X) ; vrt(S,-X) :- var(X), side(S).
55 w :- var(X), vrt(S,X), vrt(S,-X), side(S).
```

We eliminate all assignments that do not satisfy Γ.

```
56 w :- clause(C), side(S), vrt(S,-L) :
       clause(C,L).
```

Moreover, we eliminate all combinations of assignments that—when combined according to the guessed partition—do satisfy the constraint Γ.

```
57 w(C) :- clause(C,L), part(S,L), vrt(S,L),
       side(S).
58 w :- w(C) : clause(C).
59 w(C) :- w, clause(C).
```

Any combination of assignments that is not eliminated corresponds to a choice of J_1, J_2 that witnesses that the guessed partition is not independent—leading to the following result.

Theorem⋆ 8. *The answer sets S for the ASP encoding of the independent partition property applied to an agenda Φ are in one-to-one correspondence with independent partitions of Φ.*

6 Encoding Voting and Other Settings

Due to the general nature of judgment aggregation, our encodings can also be used to encode other aggregation settings into ASP—by first encoding these other settings into JA, and then applying the encodings in this paper. For example, we can encode the settings of preference aggregation and voting as follows. We declare candidates with the predicate `cand/1`, and then use the following rule to specify issues `p(X,Y)` representing a preference of candidate `X` over candidate `Y`.

```
60 issue(p(X,Y)) :- cand(X), cand(Y), X != Y.
```

We then specify constraints that require preferences to be irreflexive, total, and transitive, respectively.

```
61 clause(c1(X,Y),(p(X,Y);p(Y,X))) :- cand(X), cand(Y),
       X != Y.
62 clause(c2(X,Y),(-p(X,Y);-p(Y,X))) :- cand(X), cand(Y),
       X != Y.
63 clause(c3(X,Y,Z),(-p(X,Y);-p(Y,Z);p(X,Z))) :- cand(X),
       cand(Y), cand(Z), X != Y, Y != Z, X != Z.
```

In combination with the Kemeny JA procedure, this directly gives us an encoding of the Kemeny voting rule, for instance. Moreover, our encodings can be straightforwardly adapted to work with other encodings of voting rules into JA—e.g., using the encodings of Endriss (2018) we can obtain ASP encodings of the Borda, Slater, Copeland$^\alpha$ and k-approval voting rules. Similarly, we can slightly modify our encodings to implement the Dodgson rule and positional scoring rules. As such, we cover all voting rules that have been encoded in ASP—scoring rules [Konczak, 2006] and Kemeny, Dodgson, Copeland$^\alpha$, k-approval [Charwat and Pfandler, 2015]—and more.

Moreover, in a similar way, our encodings can be used for other settings such as graph aggregation [Endriss and Grandi, 2017] and voting in combinatorial domains [Lang and Xia, 2016]—after encoding these settings into JA.

7 Conclusion and Future Research

We provided a general and modular encoding of various JA problems into ASP—thereby providing the first structured implementation of JA methods. Our encodings are freely available in the online supplementary material.[5] The problems that we encoded include computing outcomes for a wide range of JA procedures and checking various agenda properties. These encodings open up answer set programming for an even wider range of applications than for which it has already been used—by providing recipes for creating benchmark instances based on JA problems, that can be used with real-world data sets.

We hope that this work sparks developments on improved and optimized ASP encodings for JA methods. Future work also includes ASP encodings of strategic behavior problems in JA—such as manipulation, bribery and control [Endriss *et al.*, 2012; Botan *et al.*, 2016; Baumeister *et al.*, 2013; Baumeister *et al.*, 2015; De Haan, 2017]—and further agenda properties, such as total blockedness and even negatability [Dokow and Holzman, 2010].

[5] The online material—available at https://github.com/rdehaan/ja-asp—contains encodings that go beyond those presented in this paper. They allow auxiliary variables to express constraints in the JA scenario—see, e.g., [Endriss *et al.*, 2016]. Moreover, they include encodings for more JA procedures, and for more agenda properties (e.g., finding overlapping independent partitions [Lang *et al.*, 2016]).

References

[Baumeister *et al.*, 2013] D. Baumeister, G. Erdélyi, O.J. Erdélyi, and J. Rothe. Computational aspects of manipulation and control in judgment aggregation. In *Proceedings of ADT'13*, pages 71–85, 2013.

[Baumeister *et al.*, 2015] D. Baumeister, G. Erdélyi, O.J. Erdélyi, and J. Rothe. Complexity of manipulation and bribery in judgment aggregation for uniform premise-based quota rules. *Mathematical Social Sciences*, 76:19–30, 2015.

[Biere *et al.*, 2009] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185. IOS Press, 2009.

[Botan *et al.*, 2016] S. Botan, A. Novaro, and U. Endriss. Group manipulation in judgment aggregation. In *Proceedings of AAMAS'16*, pages 411–419, 2016.

[Calimeri *et al.*, 2013] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-Core-2: 4th ASP competition official input language format, 2013. https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf.

[Charwat and Pfandler, 2015] G. Charwat and A. Pfandler. Democratix: A declarative approach to winner determination. In T. Walsh, editor, *Algorithmic Decision Theory*, pages 253–269. Springer, 2015.

[Dokow and Holzman, 2010] E. Dokow and R. Holzman. Aggregation of binary evaluations. *J. Economic Theory*, 145(2):495–511, 2010.

[Eiter and Gottlob, 1995] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[Endriss and de Haan, 2015] U. Endriss and R. de Haan. Complexity of the winner determination problem in judgment aggregation: Kemeny, Slater, Tideman, Young. In *Proceedings of AAMAS'15*, pages 117–125, 2015.

[Endriss and Grandi, 2017] U. Endriss and U. Grandi. Graph aggregation. *Artificial Intelligence*, 245:86–114, 2017.

[Endriss *et al.*, 2012] Ulle Endriss, Umberto Grandi, and Daniele Porello. Complexity of judgment aggregation. *J. Artif. Intell. Res.*, 45:481–514, 2012.

[Endriss *et al.*, 2016] U. Endriss, U. Grandi, R. de Haan, and J. Lang. Succinctness of languages for judgment aggregation. In *KR'16*, pages 176–186, 2016.

[Endriss, 2018] U. Endriss. Judgment aggregation with rationality and feasibility constraints. In *Proceedings of AAMAS' 18*, pages 946–954, 2018.

[Gebser *et al.*, 2011a] M. Gebser, R. Kaminski, and T. Schaub. Complex optimization in answer set programming. *TPLP*, 11(4–5):821–839, 2011.

[Gebser *et al.*, 2011b] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The potsdam answer set solving collection. *AI Communications*, 24(2):107–124, April 2011.

[Gebser *et al.*, 2017] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, Ostrowski. M, J. Romero, T. Schaub, and S Schiele. Potassco user guide (v2.1.0). https://github.com/potassco/guide/releases/tag/v2.1.0, 2017.

[Gebser *et al.*, 2018] M. Gebser, N. Leone, M. Maratea, S. Perri, F. Ricca, and T. Schaub. Evaluation techniques and systems for answer set programming: a survey. In *Proceedings of IJCAI'18*, pages 5450–5456, 7 2018.

[Grossi and Pigozzi, 2014] D. Grossi and G. Pigozzi. *Judgment Aggregation: A Primer*. Morgan and Claypool, 2014.

[de Haan and Slavkovik, 2017] R. de Haan and M. Slavkovik. Complexity results for aggregating judgments using scoring or distance-based procedures. In *Proceedings of AAMAS'17*, 2017.

[de Haan, 2016] R. de Haan. Parameterized complexity results for the kemeny rule in judgment aggregation. In *Proceedings of ECAI'16*, volume 285, pages 1502–1510, 2016.

[de Haan, 2017] R. de Haan. Complexity results for manipulation, bribery and control of the kemeny judgment aggregation procedure. In *Proceedings of AAMAS'17*, 2017.

[de Haan, 2018] R. de Haan. Hunting for tractable languages for judgment aggregation. In *Proceedings of KR'18*. AAAI Press, 2018.

[Jamroga and Slavkovik, 2013] W. Jamroga and M. Slavkovik. Some complexity results for distance-based judgment aggregation. In *LNCS Proceedings PRIMA'13*, pages 313–325, 2013.

[Konczak, 2006] K. Konczak. Voting theory in answer set programming. In *WLP*, volume 1843-06-02 of *INFSYS Research Report*, pages 45–53. Technische Universität Wien, Austria, 2006.

[Lang and Slavkovik, 2013] J. Lang and M. Slavkovik. Judgment aggregation rules and voting rules. In *Proceedings of ADT'13*, volume 8176 of *LNAI*, pages 230–244. Springer, 2013.

[Lang and Slavkovik, 2014] J. Lang and M. Slavkovik. How hard is it to compute majority-preserving judgment aggregation rules? In *Proceedings of ECAI'14*, volume 263, pages 501–506, 2014.

[Lang and Xia, 2016] J. Lang and L. Xia. Voting in combinatorial domains. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. Procaccia, editors, *Handbook of Computational Social Choice*. Cambridge University Press, Cambridge, 2016.

[Lang *et al.*, 2016] J. Lang, M. Slavkovik, and S. Vesic. Agenda separability in judgment aggregation. In *Proceedings of AAAI'16*, pages 1016–1022, 2016.

[Lang *et al.*, 2017] J. Lang, G. Pigozzi., M. Slavkovik, L. van der Torre, and S. Vesic. A partial taxonomy of judgment aggregation rules and their properties. *Social Choice and Welfare*, 48(2):327–356, 2017.

[Slavkovik, 2016] M. Slavkovik. An introductory course to judgment aggregation. *CoRR*, abs/1607.03307, 2016.