Proceedings of the Sixteenth International Conference on
Principles of Knowledge Representation and Reasoning (KR 2018)

Hunting for Tractable Languages for Judgment Aggregation

Ronald de Haan
Institute for Logic, Language and Computation
University of Amsterdam
me@ronalddehaan.eu

Abstract

Judgment aggregation is a general framework for collective decision making that can be used to model many different settings. Due to its general nature, the worst case complexity of essentially all relevant problems in this framework is very high. However, these intractability results are mainly due to the fact that the language to represent the aggregation domain is overly expressive. We initiate an investigation of representation languages for judgment aggregation that strike a balance between (1) being limited enough to yield computational tractability results and (2) being expressive enough to model relevant applications. In particular, we consider the languages of Krom formulas, (definite) Horn formulas, and Boolean circuits in decomposable negation normal form (DNNF). We illustrate the use of the positive complexity results that we obtain for these languages with a concrete application: voting on how to spend a budget (i.e., participatory budgeting).

Introduction

Judgment aggregation is a general framework to study methods for collective opinion forming, that has been investigated in the area of computational social choice (see, e.g., Endriss 2016, Grossi and Pigozzi 2014). The framework is set up in such a general way that it can be used to model an extremely wide range of scenarios—including, e.g., the setting of voting (Dietrich and List 2007). On the one hand, this generality is an advantage: methods studied in judgment aggregation can be employed in all these scenarios. On the other hand, however, this generality severely hinders the use of judgment aggregation methods in applications. Because there are no restrictions on the type of aggregation settings that are modeled, relevant computational tasks across the board are computationally intractable in the worst case. In other words, no performance guarantees are available that warrant the efficient use of judgment aggregation methods for applications—not even for simple settings. For example, computing the outcome of a judgment aggregation scenario is NP-hard for all aggregation procedures studied in the literature that satisfy the rudimentary quality condition of *consistency* (Endriss and De Haan 2015; De Haan and Slavkovik 2017; Lang and Slavkovik 2014).

These negative computational complexity results are in many cases due purely to the expressivity of the language used to represent aggregation scenarios (full propositional logic, or CNF formulas)—not to the structure of the scenario being modeled. In other words, the known negative complexity results draw an overly negative picture

To correct this gloomy and misleading image, a more detailed and more fine-grained perspective is needed on the way that application settings are modeled in the general framework of judgment aggregation. In this paper, we take a first look at the complexity of judgment aggregation scenarios using this more sensitive point of view. That is, we initiate an investigation of representation languages for judgment aggregation that (1) are modest enough to yield positive complexity results for relevant computational tasks, yet (2) are general enough to model interesting and relevant applications.

Concretely, we look at several restricted propositional languages that strike a balance between expressivity and tractability in other settings, and we study to what extent such a balance is attained in the setting of judgment aggregation. In particular, we look at Krom (2CNF), Horn and definite Horn formulas, and we consider the class of Boolean circuits in decomposable negation normal form (DNNF). We study the impact of these restricted languages on the complexity of computing outcomes for a number of judgment aggregation procedures studied in the literature. We obtain a wide range of (positive and negative) results. Most of the results we obtain are summarized in Tables 3, 4 and 5, located in later sections.

In particular, we obtain several interesting positive complexity results for the case where the domain is represented using a Boolean circuit in DNNF. Additionally, we illustrate how this representation language of Boolean circuits in DNNF—that combines expressivity and tractability—can be used to get tractability results for a specific application: voting on how to spend a budget. This application setting can be seen as an instantiation of the setting of *Participatory Budgeting* (see, e.g., Benade et al. 2017).

Related Work Judgment aggregation has been studied in the field of computational social choice from (a.o.) a philosophy, economics and computer science perspective (see, e.g., Dietrich 2007, Endriss 2016, Grossi and Pigozzi 2014, Lang et al. 2017, List and Pettit 2002, Rothe 2016). The complexity

of computing outcomes for judgment aggregation procedures has been studied by, a.o., Endriss, Grandi, and Porello (2012), Endriss et al. (2016), Endriss and De Haan (2015), De Haan and Slavkovik (2017) and Lang and Slavkovik (2014). See Table 2 for complexity results that are relevant for this paper.

Roadmap We begin by explaining the framework of judgment aggregation. We then study to what extent the known languages of Krom and (definite) Horn formulas lead to suitable results for judgment aggregation. We continue with looking at the class of DNNF circuits—studied in the field of knowledge compilation—and we illustrate how results for this class of circuits can be used for a concrete application of judgment aggregation (that of voting on how to allocate a budget). We conclude with outlining some promising ways in which the research path that we set out can be followed.

An overview of notions from propositional logic and computational complexity theory that we use can be found in the appendix. The proofs of some results are omitted from the paper—these results are marked with a star ($\star$). Full proofs can be found in the accompanying technical report (De Haan 2018).

Judgment Aggregation

We begin by introducing the setting of Judgment Aggregation (Dietrich 2007; Endriss 2016; Grossi and Pigozzi 2014; List and Pettit 2002). In this paper, we will use a variant of the framework that has been studied by, e.g., Grandi (2012), Grandi and Endriss (2013) and Endriss et al. (2016).[1]

Let $\mathcal{I} = \{x_1, \ldots, x_n\}$ be a finite set of *issues*, in the form of propositional variables. Intuitively, these issues are the topics about which the individuals want to combine their judgments. A truth assignment $\alpha : \mathcal{I} \to \{0,1\}$ is called a *ballot*, and represents an opinion that individuals and the group can have. We will also denote ballots α by a binary vector $(b_1, \ldots, b_n) \in \{0,1\}^n$, where $b_i = \alpha(x_i)$ for each $i \in [n]$—we use $[n]$ to denote $\{1, \ldots, n\}$ for each $n \in \mathbb{N}$. Moreover, we say that $(p_1, \ldots, p_n) \in \{0,1,\star\}^n$ is a *partial ballot*, and that $(p_1, \ldots, p_n)$ *agrees with* a ballot $(b_1, \ldots, b_n)$ if $p_i = b_i$ whenever $p_i \neq \star$, for all $i \in [n]$. We use an integrity constraint Γ to restrict the set of feasible opinions (for both the individuals and the group). The integrity constraint Γ is a propositional formula (or more generally, a single-output Boolean circuit), whose variables can include $x_1, \ldots, x_n$. We define the set $\mathcal{R}(\mathcal{I}, \Gamma)$ of *rational ballots* to be the ballots (for $\mathcal{I}$) that are consistent with the integrity constraint Γ. We say that finite sequences $\boldsymbol{r} \in \mathcal{R}(\mathcal{I}, \Gamma)^+$ of rational ballots are *profiles*. A profile contains a ballot for each individual participating in the judgment aggregation scenario. Where convenient we equate a profile $\boldsymbol{r} = (r_1, \ldots, r_p)$ with the multiset containing $r_1, \ldots, r_p$.

A *judgment aggregation procedure* (or *rule*), for the set $\mathcal{I}$ of issues and the integrity constraint Γ, is a function F that takes as input a profile $\boldsymbol{r} \in \mathcal{R}(\mathcal{I}, \Gamma)^+$, and that produces a non-empty set of ballots. A procedure F is called *consistent* if for all $\mathcal{I}$, Γ and $\boldsymbol{r}$ it holds that each $r^* \in F(\boldsymbol{r})$ is consistent with Γ. Consistency is a central requirement for judgment aggregation procedures, and all rules that we consider in this paper are consistent.

An example of a simple judgment aggregation procedure is the *majority rule* (defined for profiles with an odd number of ballots). We let the majority outcome $m_{\boldsymbol{r}}$ be the partial ballot such that for each $x \in \mathcal{I}$, $m_{\boldsymbol{r}}(x) = 1$ if a strict majority of ballots $r_i \in \boldsymbol{r}$ satisfy $r_i(x) = 1$, $m_{\boldsymbol{r}}(x) = 0$ if a strict majority of ballots $r_i \in \boldsymbol{r}$ satisfy $r_i(x) = 0$, and $m_{\boldsymbol{r}}(x) = \star$ otherwise. The majority rule returns the majority outcome $m_{\boldsymbol{r}}$. The majority rule is efficient to compute, but is not consistent (as shown in Example 1).

Example 1. *Consider the judgment aggregation scenario. where* $\mathcal{I} = \{x_1, x_2, x_3\}$, $\Gamma = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, *and the profile* $\boldsymbol{r} = (r_1, r_2, r_3)$ *is as shown in Table 1. The majority outcome* $\textsc{maj}(\boldsymbol{r})$ *is inconsistent with* Γ.

$\boldsymbol{r}$	x_1	x_2	x_3
r_1	1	1	0
r_2	1	0	1
r_3	0	1	1
$\textsc{maj}(\boldsymbol{r})$	1	1	1

Table 1: Example of a judgment aggregation scenario.

Judgment Aggregation Procedures

Next, we introduce the judgment aggregation rules that we use in this paper. These procedures are all consistent and are many of the ones that have been studied in the literature (for an overview see, e.g., Lang et al. 2017).

Several procedures that we consider can be seen as instantiations of a general template: *scoring procedures*. Let $\mathcal{I}$ be a set of issues and Γ be an integrity constraint. Moreover, let $s : \mathcal{R}(\mathcal{I}, \Gamma) \times \text{Lit}(\mathcal{I}) \to \mathbb{N}$ be a *scoring function* that assigns a value to each literal $l \in \text{Lit}(\mathcal{I})$ with respect to a ballot $r \in \mathcal{R}(\mathcal{I}, \Gamma)$. The scoring judgment aggregation procedure F_s that corresponds to s is defined as follows:

$$F_s(\boldsymbol{r}) = \operatorname*{arg\,max}_{r \in \mathcal{R}(\mathcal{I},\Gamma)} \sum_{r_i \in \boldsymbol{r}} \sum_{\substack{l \in \text{Lit}(\mathcal{I}) \\ r(l)=1}} s(r_i, l).$$

That is, F_s selects the rational ballots $r \in \mathcal{R}(\mathcal{I}, \Gamma)$ that maximize the cumulative score for all literals agreeing with r with respect to all ballots $r_i \in \boldsymbol{r}$.

The *median (or Kemeny) procedure* MED is based on the scoring function and is defined by letting $s_{\text{K}}(r, l) = r(l)$ for each $r \in \mathcal{R}(\mathcal{I}, \Gamma)$ and each $l \in \text{Lit}(\mathcal{I})$. Alternatively, the MED procedure can be defined as the rule that selects the ballots $r^* \in \mathcal{R}(\mathcal{I}, \Gamma)$ that minimize the cumulative Hamming distance to the profile $\boldsymbol{r}$. The *Hamming distance* between two ballots r, r' is $d_{\text{H}}(r, r') = |\{\, x \in \mathcal{I} : r(x) \neq r'(x) \,\}|$.

The *reversal scoring procedure* REV is based on the scoring function $s_{\text{R}}(r, l)$ such that $s_{\text{R}}(r, l) = \min_{r' \in \mathcal{R}(\mathcal{I},\Gamma), r'(l)=0} d_{\text{H}}(r, r')$ for each $r \in \mathcal{R}(\mathcal{I}, \Gamma)$ and each $l \in \text{Lit}(\mathcal{I})$. That is, the score $s_{\text{R}}(r, l)$ of l w.r.t. r is

[1] This framework is also known under the name of *binary aggregation with integrity constraints*, and can be used interchangeably with other Judgment Aggregation frameworks from the literature —as shown by Endriss et al. (2016).

the minimal number of issues whose truth value needs to be flipped to get a rational ballot r' that sets l to false.

The *max-card Condorcet (or Slater) procedure* MCC is also based on the Hamming distance. Let $\boldsymbol{r}$ be a profile. The MCC procedure is defined by letting $\text{MCC}(\boldsymbol{r}) = \arg\min_{r^* \in \mathcal{R}(\mathcal{I},\Gamma)} d_{\text{H}}(r^*, m_{\boldsymbol{r}})$. That is, the MCC procedure selects the rational ballots that minimize the Hamming distance to the majority outcome $m_{\boldsymbol{r}}$.

The *Young procedure* YOUNG selects those ballots that can be obtained as a rational majority outcome by deleting a minimal number of ballots from the profile. Let $\boldsymbol{r}$ be a profile, and let d denote the smallest number such that deleting d individual ballots from $\boldsymbol{r}$ results in a profile $\boldsymbol{r}'$ such that $m_{\boldsymbol{r}'}$ is a complete and rational ballot. We let the outcome $\text{YOUNG}(\boldsymbol{r})$ of the Young procedure be the set of rational ballots r^* such that deleting d individual from $\boldsymbol{r}$ results in a profile $\boldsymbol{r}'$ with $m_{\boldsymbol{r}'} = r^*$.

The *Max-Hamming procedure* MAXHAM is also based on the Hamming distance. Let r be a single ballot, and let $\boldsymbol{r} = (r_1, \ldots, r_p)$ be a profile. We define the max-Hamming distance between r and $\boldsymbol{r}$ to be $d_{\text{max,H}}(r, \boldsymbol{r}) = \max_{r_i \in \boldsymbol{r}} d_{\text{H}}(r, r_i)$. The Max-Hamming procedure is defined by letting $\text{MAXHAM}(\boldsymbol{r}) = \arg\min_{r^* \in \mathcal{R}(\mathcal{I},\Gamma)} d_{\text{max,H}}(r^*, \boldsymbol{r})$. That is, the Max-Hamming procedure selects the rational ballots that minimize the max-Hamming distance to $\boldsymbol{r}$.

The *ranked agenda (or Tideman) procedure* RA is based on the notion of majority strength.[2] Let $\boldsymbol{r}$ be a profile and let $l \in \text{Lit}(\mathcal{I})$. The majority strength $\text{ms}(\boldsymbol{r}, l)$ of l for $\boldsymbol{r}$ is the number of ballots $r \in \boldsymbol{r}$ such that $r(l) = 1$. Let $<_{\text{tb}}$ be a fixed linear order on $\text{Lit}(\mathcal{I})$ (the tie-breaking order). Based on $<_{\text{tb}}$ and the majority strength, we define the linear order $<_{\boldsymbol{r}}$ on $\text{Lit}(\mathcal{I})$. Let $l_1, l_2 \in \text{Lit}(\mathcal{I})$. Then $l_1 <_{\boldsymbol{r}} l_2$ if either (i) $\text{ms}(\boldsymbol{r}, l_1) > \text{ms}(\boldsymbol{r}, l_2)$ or (ii) $\text{ms}(\boldsymbol{r}, l_1) = \text{ms}(\boldsymbol{r}, l_2)$ and $l_1 <_{\text{tb}} l_2$. Then $\text{RA}(\boldsymbol{r}) = \{r^*\}$ where the ballot r^* is defined inductively as follows. Let $l_1, l_2, \ldots, l_{2n}$ be such that for each $i \in [2n-1]$ it holds that $l_i <_{\boldsymbol{r}} l_{i+1}$. Let s_0 be the empty truth assignment. For each $i \in [2n-1]$, check whether both $s_i(l_i) \neq 0$ and s_i' is consistent with Γ, where s_i' is obtained from s_i by setting l_i to true (and keeping the assignments to variables not occurring in l_i unchanged). If both are the case, then let $s_{i+1} = s_i'$. Otherwise, let $s_{i+1} = s_i$. Then $r^* = s_{2n}$. Intuitively, the procedure iterates over the assignments $l_1, l_2, \ldots$ in the order specified by $<_{\boldsymbol{r}}$. Each literal l_i is set to true whenever this does not lead to an inconsistency with previously assigned literals.

Outcome Determination

When given a judgment aggregation scenario (i.e., an agenda, an integrity constraint, and a profile of individual opinions), an important computational task is to compute a possible collective opinion, for a fixed judgment aggregation procedure. This task is often referred to as *outcome determination*. Moreover, often it makes sense to seek possible collective opinions that satisfy certain properties (e.g., whether or not a given issue is accepted in the collective opinion).

Essentially, this is a search problem: the task is to find one of (possibly) multiple solutions. However, to make the theoretical complexity analysis easier, we will consider the following decision variant of this problem.

OUTCOME(F)
Instance: A set $\mathcal{I}$ of issues with an integrity constraint Γ a profile $\boldsymbol{r} \in \mathcal{R}(\mathcal{I}, \Gamma)^+$ and a partial ballot s (for $\mathcal{I}$).
Question: Is there a ballot $r^* \in F(\boldsymbol{r})$ such that s agrees with r^*?

An outcome r^* witnessing a yes-answer can be obtained by solving this decision problem a linear number of times. In addition to the basic task of finding one outcome (that agrees with a given partial ballot s), one could consider other computational tasks, e.g., representing the set $F(\boldsymbol{r})$ of outcomes in a succinct way that admits certain queries/operations to be performed efficiently. For example, it might be desirable to enumerate all (possibly exponentially many) outcomes with polynomial delay. It could also be desirable to check whether all outcomes agree with a given partial ballot s (*skeptical reasoning*). For the sake of simplicity, in this paper we will stick to the decision problem described above. All tractability results that we obtain for the decision problem can straightforwardly be extended to tractability results for the above computational tasks.

For the judgment aggregation procedures F that we considered above, OUTCOME(F) is Θ_2^{p}-hard. For an overview, see Table 2.

F	complexity of OUTCOME(F)	
MED	Θ_2^{p}-c	(Lang and Slavkovik 2014)
REV	Θ_2^{p}-c	(De Haan and Slavkovik 2017)
MCC	Θ_2^{p}-c	(Lang and Slavkovik 2014)
YOUNG	Θ_2^{p}-c	(Endriss and De Haan 2015)
MAXHAM	Θ_2^{p}-c	(De Haan and Slavkovik 2017)
RA	Δ_2^{p}-c	(Endriss and De Haan 2015)

Table 2: The computational complexity of outcome determination for various procedures F.

Krom and (Definite) Horn Formulas

In this section, we consider the fragments of Krom (2CNF), Horn and definite Horn formulas—for a formal definition of these fragments, see the appendix. These fragments can be used to express settings where only basic dependencies between issues play a role—see Example 2 for an indication.

Example 2. *Krom (2CNF) formulas can be used to express dependencies of the form "if we decide to use software tool 1 (s_1) or software tool 2 (s_2), then we need to purchase the entire package (p):"* $(s_1 \vee s_2) \rightarrow p \equiv (\neg s_1 \vee p) \wedge (\neg s_2 \vee p)$.

Definite Horn formulas can be used to express dependencies of the form "if we hire both researcher 1 (r_1) and researcher 2 (r_2), then we need to rent another office o:" $(r_1 \wedge r_2) \rightarrow o \equiv (\neg r_1 \vee \neg r_2 \vee o)$.

[2] Here, we consider a variant of the ranked agenda procedure that works with a fixed tie-breaking order. Other variants, where all possible tie-breaking orders are considered in parallel, have also been studied in the literature (see, e.g., Lang et al. 2017).

For some judgment aggregation rules these fragments make computing outcomes tractable, and for other judgment aggregation rules they do not. We begin with considering the rules MED and MCC. Computing outcomes for these rules is tractable when restricted to Krom formulas, but not when restricted to (definite) Horn formulas.

Proposition* 1. OUTCOME(MED) *is* Θ_2^{p}*-hard even when restricted to the case where* $\Gamma \in$ DEFHORN.

Proposition* 2. OUTCOME(MCC) *is* Θ_2^{p}*-hard even when restricted to the case where* $\Gamma \in$ DEFHORN.

The following result refers to the notion of majority consistency (see, e.g., Lang and Slavkovik 2014). A profile $\boldsymbol{r}$ is *majority consistent* (with respect to an integrity constraint Γ) if the majority outcome $m_{\boldsymbol{r}}$ is consistent with Γ. A judgment aggregation procedure is *majority consistent* if for each integrity constraint Γ and each profile $\boldsymbol{r}$ that is majority consistent (w.r.t. Γ), the procedure outputs all and only those complete ballots that agree with the (partial) ballot $m_{\boldsymbol{r}}$.

Theorem 3. *For all judgment aggregation procedures F that are majority consistent, e.g., $F \in \{\text{MED}, \text{MCC}\}$,* OUTCOME(F) *is polynomial-time solvable when* $\Gamma \in$ KROM.

Proof. The general idea behind this proof is to use the property that when $\Gamma \in$ KROM, the majority outcome $m_{\boldsymbol{r}}$ is always Γ-consistent. Let $(\mathcal{I}, \Gamma, \boldsymbol{r}, s)$ be an instance of OUTCOME(F) with $\Gamma \in$ KROM. Let $\boldsymbol{r} = (r_1, \ldots, r_p)$. We consider the majority outcome $r^* = m_{\boldsymbol{r}}$.

We show that the partial ballot r^* is consistent with Γ. Suppose, to derive a contradiction, that r^* is inconsistent with Γ. Then there must be some clause $(l_1 \vee l_2)$ of size 2 such that $\Gamma \models (l_1 \vee l_2)$ and r^* sets both l_1 and l_2 to false. By definition of r^*, then a strict majority of the ballots in $\boldsymbol{r}$ set l_1 to false, and a strict majority of the ballots in $\boldsymbol{r}$ set l_2 to false. By the pigeonhole principle then there must be some ballot r_i in $\boldsymbol{r}$ that sets both l_1 and l_2 to false. However, since $\Gamma \models (l_1 \vee l_2)$, we get that r_i does not satisfy Γ, which is a contradiction with our assumption that all ballots in the profile satisfy Γ. Thus, we can conclude that r^* is consistent with Γ.

Since F is majority consistent, we know that $F(\boldsymbol{r})$ contains all ballots that are consistent with both r^* and Γ. Since $\Gamma \in$ KROM, deciding if $F(\boldsymbol{r})$ contains a ballot that is consistent with s can be done in polynomial time. □

We continue with the MAXHAM procedure for which computing outcomes is not tractable when restricted to Krom formulas nor when restricted to definite Horn formulas.

Proposition* 4. OUTCOME(MAXHAM) *is* Θ_2^{p}*-hard even when restricted to the case where* $\Gamma = \top$.

OUTCOME(MAXHAM) restricted to the case where $\Gamma = \top$ coincides with a problem known as CLOSEST STRING for binary alphabets (see, e.g., Li, Ma, and Wang 2002). To the best of our knowledge, this is the first time that the exact complexity of (this variant of) this problem has been identified. OUTCOME(MAXHAM) is also very similar to the problem of computing outcomes for the minimax rule in approval voting (Brams, Kilgour, and Sanver 2004).

Corollary 5. OUTCOME(MAXHAM) *is* Θ_2^{p}*-hard even when restricted to the case where* $\Gamma \in$ DEFHORN $\cap$ KROM.

Finally, we consider the procedure RA, for which computing outcomes is tractable for both Krom and Horn formulas.

Theorem 6. *Let $\mathcal{C}$ be a class of propositional formulas (or Boolean circuits) with the following two properties:*

- *$\mathcal{C}$ is closed under instantiation, i.e., for any $\Gamma \in \mathcal{C}$ and any partial truth assignment $\alpha : \mathrm{Var}(\Gamma) \to \{0, 1\}$ it holds that $\Gamma[\alpha] \in \mathcal{C}$; and*
- *satisfiability of formulas in $\mathcal{C}$ is polynomial-time solvable.*

Then OUTCOME(RA) *is polynomial-time solvable when restricted to the case where* $\Gamma \in \mathcal{C}$.

Proof (sketch). Let $\mathcal{C}$ be a class of propositional formulas that satisfies the conditions stated above, and let $\Gamma \in \mathcal{C}$. We can then compute OUTCOME(RA) $= \{r^*\}$ by directly using the iterative definition of r^* given in the description of the ranked agenda procedure. This definition iteratively constructs partial ballots $s_0, \ldots, s_{2n}$. Ballot s_0 is the empty ballot, and for each $i > 0$, ballot s_i is constructed from s_{i-1} by using only the operations of instantiating the integrity constraint and checking satisfiability of the resulting formula. Due to the properties of $\mathcal{C}$, these operations are all polynomial-time solvable. Thus, constructing $r^* = s_{2n}$ can be done in polynomial time. □

Corollary 7. *For each $\mathcal{C} \in \{\text{KROM}, \text{HORN}\}$,* OUTCOME(RA) *is polynomial-time solvable when restricted to the case where* $\Gamma \in \mathcal{C}$.

An overview of the complexity results that we established in this section can be found in Table 3.

F	complexity of OUTCOME(F) restricted to HORN / DEFHORN	
MED	Θ_2^{p}-c	(Proposition 1)
MCC	Θ_2^{p}-c	(Proposition 2)
MAXHAM	Θ_2^{p}-c	(Corollary 5)
RA	in P	(Corollary 7)
F	complexity of OUTCOME(F) restricted to KROM	
MED	in P	(Theorem 3)
MCC	in P	(Theorem 3)
MAXHAM	Θ_2^{p}-c	(Corollary 5)
RA	in P	(Corollary 7)

Table 3: The computational complexity of outcome determination for several procedures F restricted to the case where $\Gamma \in$ KROM, the case where $\Gamma \in$ HORN, and the case where $\Gamma \in$ DEFHORN.

The results that we obtained for Horn formulas can all be straightforwardly extended to the fragment of renamable Horn formulas—e.g., the fragment of renamable Horn formulas satisfies the requirements of Theorem 6. A propositional formula φ is *renamable Horn* if there is a set $R \subseteq \mathrm{Var}(\varphi)$ of variables such that φ becomes Horn when all literals over R are complemented.

Boolean Circuits in DNNF

Next, we consider the case where the integrity constraints are restricted to Boolean circuits in Decomposable Negation Normal Form (DNNF). This is a class of Boolean circuits studied in the area of knowledge compilation. We illustrate how this class of circuits is useful for judgment aggregation.

Knowledge Compilation

Knowledge compilation (see, e.g., Darwiche and Marquis 2002, Darwiche 2014, Marquis 2015) refers to a collection of approaches for solving reasoning problems in the area of artificial intelligence and knowledge representation and reasoning that are computationally intractable in the worst-case asymptotic sense. These reasoning problems typically involve knowledge in the form of a Boolean function—often represented as a propositional formula. The general idea behind these approaches is to split the reasoning process into two phases: (1) compiling the knowledge into a different format that allows the reasoning problem to be solved efficiently, and (2) solving the reasoning problem using the compiled knowledge. Since the entire reasoning problem is computationally intractable, at least one of these two phases must be intractable. Indeed, typically the first phase does not enjoy performance guarantees on the running time—upper bounds on the size of the compiled knowledge are often desired instead. One of the advantages of this methodology is that one can reuse the compiled knowledge for many instances, which could lead to a smaller overall running time.

A prototypical example of a problem studied in the setting of knowledge compilation is that of *clause entailment* (see, e.g., Darwiche and Marquis 2002, Cadoli et al. 2002). In this problem, one is given a knowledge base, say in the form of a propositional formula φ in CNF, together with a clause δ. The question is to decide whether $\varphi \models \delta$. This problem is co-NP-complete in general. The knowledge compilation approach to solve this problem would be to firstly compile the CNF formula φ into an equivalent expression in a different format. For example, one could consider the formalism of Boolean circuits in *Decomposable Negation Normal Form (DNNF)* (or *DNNF circuits*, for short).

DNNF circuits are a particular class of Boolean circuits in *Negation Normal Form (NNF)*. A Boolean circuit C in NNF is a direct acyclic graph with a single root (a node with no ingoing edges) where each leaf is labelled with $\top$, $\bot$, x or $\neg x$ for a propositional variable x, and where each internal node is labelled with $\wedge$ or $\vee$. (An arc in the graph from N_1 to N_2 indicates that N_2 is a child node of N_1.) The set of propositional variables occurring in C is denoted by $\mathrm{Var}(C)$. For any truth assignment $\alpha : \mathrm{Var}(C) \to \{0, 1\}$, we define the truth value $C[\alpha]$ assigned to C by α in the usual way, i.e., each node is assigned a truth value based on its label and the truth value assigned to its children, and the truth value assigned to C is the truth value assigned to the root of the circuit. DNNF circuits are Boolean circuits in NNF that satisfy the additional property of decomposability. A circuit C is *decomposable* if for each conjunction in the circuit, the conjuncts do not share variables. That is, for each node d in C that is labelled with $\wedge$ and for any two children d_1, d_2 of this node, it holds that $\mathrm{Var}(C_1) \cap \mathrm{Var}(C_2) = \emptyset$, where C_1, C_2 are the subcircuits of C that have d_1, d_2 as root, respectively. An example of a DNNF circuit is given in Figure 1.

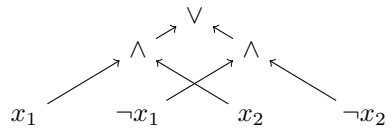


Figure 1: An example of a DNNF circuit.

The problem of clause entailment can be solved in polynomial time when the propositional knowledge is given as a DNNF circuit (Darwiche and Marquis 2002). Moreover, every CNF formula can be translated to an equivalent DNNF circuit—without guarantees on the size of the circuit. Thus, one could solve the problem of clause entailment by first compiling the CNF formula φ into an equivalent DNNF circuit C (without guarantees on the running time or size of the result) and then solving $C \models \delta$ in time polynomial in $|C|$.

Next, we will show how representation languages such as DNNF circuits can be used in the setting of Judgment Aggregation, and we will argue how Judgment Aggregation can benefit from the approach of first compiling knowledge (without performance guarantees) before using the compiled knowledge to solve the initial problem.

Algebraic Model Counting

We will use the technique of algebraic model counting (Kimmig, Van den Broeck, and De Raedt 2017) to execute several judgment aggregation procedures efficiently using the structure of DNNF circuits. Algebraic model counting is a generalization of the problem of counting models of a Boolean function that uses the addition and multiplication operators of a commutative semiring.

Definition 1 (Commutative semiring)**.** *A* semiring *is a structure* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$*, where:*

- *addition* $\oplus$ *is an associative and commutative binary operation over the set* $\mathcal{A}$*;*
- *multiplication* $\otimes$ *is an associative binary operation over the set* $\mathcal{A}$*;*
- $\otimes$ *distributes over* $\oplus$*;*
- $e^{\oplus} \in \mathcal{A}$ *is the neutral element of* $\oplus$*, i.e., for all* $a \in \mathcal{A}$*,* $a \oplus e^{\oplus} = a$*;*
- $e^{\otimes} \in \mathcal{A}$ *is the neutral element of* $\otimes$*, i.e., for all* $a \in \mathcal{A}$*,* $a \otimes e^{\otimes} = a$*; and*
- $e^{\oplus}$ *is an annihilator for* $\otimes$*, i.e., for all* $a \in \mathcal{A}$*,* $e^{\oplus} \otimes a = a \otimes e^{\oplus} = e^{\oplus}$*.*

When $\otimes$ *is commutative, we say that the semiring is commutative. When* $\oplus$ *is idempotent, we say that the semiring is idempotent.*

Definition 2 (Algebraic model counting)**.** *Given:*

- *a Boolean function* f *over a set* $\mathcal{I}$ *of propositional variables;*
- *a commutative semiring* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$*, and*

- *a labelling function* $\lambda : \mathrm{Lit}(\mathcal{I}) \to \mathcal{A}$ *mapping literals over the variables in* $\mathcal{I}$ *to values in the set* $\mathcal{A}$*,*

the task of algebraic model counting (AMC) *is to compute:*

$$\boldsymbol{A}(f) = \bigoplus_{\substack{\alpha:\mathcal{I}\to\{0,1\} \\ f(\alpha)=1}} \bigotimes_{\substack{l\in\mathrm{Lit}(\mathcal{I}) \\ \lambda(l)=1}} \lambda(l).$$

We can solve the task of algebraic model counting efficiently for DNNF circuits when the semiring satisfies an additional condition.

Definition 3 (Neutral $(\oplus, \alpha)$)**.** *Let* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ *be a semiring, and let* $\lambda : \mathrm{Lit}(\mathcal{I}) \to \mathcal{A}$ *be a labelling function for some set* $\mathcal{I}$ *of propositional variables. A pair* $(\oplus, \lambda)$ *is* neutral *if for all* $x \in \mathcal{I}$ *it holds that* $\lambda(x) \oplus \lambda(\neg x) = e^{\otimes}$*.*

Theorem 8 (Kimmig, Van den Broeck, and De Raedt 2017, Thm 5)**.** *When* f *is represented as a DNNF circuit, and the semiring* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ *and the labelling function* λ *have the properties that (i)* $\oplus$ *is idempotent, and (ii)* $(\oplus, \lambda)$ *is neutral, then the algebraic model counting problem is polynomial-time solvable—when given* f *and* λ *as input, and when the operations of addition (*$\oplus$*) and multiplication (*$\otimes$*) over* $\mathcal{A}$ *can be performed in polynomial time.*

We will use the result of Theorem 8 to show that outcome determination for several judgment aggregation procedures is tractable for the case where Γ is a DNNF circuit. To do so, we will consider the following commutative, idempotent semiring (also known as the *max-plus algebra*). We let $\mathcal{A} = \mathbb{Z} \cup \{-\infty, \infty\}$, we let $\oplus = \max$, $\otimes = +$, $e^{\oplus} = -\infty$, and $e^{\otimes} = 1$. Whenever we have a labelling function α such that $(\oplus, \lambda)$ is neutral—i.e., such that $\max(\lambda(x), \lambda(\neg x)) = 0$ for each $x \in \mathcal{I}$—we satisfy the conditions of Theorem 8.

Theorem 9. OUTCOME(MED) *and* OUTCOME(MCC) *are polynomial-time computable when* Γ *is a DNNF circuit.*

Proof. We prove the statement for OUTCOME(MED). The case for OUTCOME(MCC) is analogous. Let $(\mathcal{I}, \Gamma, \boldsymbol{r}, s)$ be an instance of OUTCOME(MED). We solve the problem by reducing it to the problem of algebraic model counting. For $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$, we use the max-plus algebra described above. We construct the labelling function λ as follows. For each $x \in \mathcal{I}$, we count the number $n_{x,1}$ of ballots $r \in \boldsymbol{r}$ such that $r(x) = 1$ and we count the number $n_{x,0}$ of ballots $r \in \boldsymbol{r}$ such that $r(x) = 0$. That is, we let $n_{x,0}$ and $n_{x,1}$ be the majority strength of $\neg x$ and x, respectively, in the profile $\boldsymbol{r}$. We pick a constant c_x such that $\max\{n'_{x,0}, n'_{x,1}\} = 0$ where $n'_{x,0} = n_{x,0} + c_x$ and $n'_{x,1} = n_{x,1} + c_x$. We then let $\lambda(x) = n'_{x,1}$ and $\lambda(\neg x) = n'_{x,0}$. This ensures that $(\oplus, \lambda)$ satisfies the condition of neutrality (i.e., that $\lambda(x) \oplus \lambda(\neg x) = e^{\otimes}$ for each $x \in \mathcal{I}$).

This choice of $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ and λ has the property that the ballots $r^* \in \mathrm{MED}(\boldsymbol{r})$ are exactly those complete ballots r^* that satisfy Γ and for which holds that $\boldsymbol{A}(\Gamma) = \bigotimes_{l\in\mathrm{Lit}(\mathcal{I}), r^*(l)=1} \lambda(l)$. That is, the set $\mathrm{MED}(\boldsymbol{r})$ consists of those rational ballots that achieve the solution of the algebraic model counting problem $\boldsymbol{A}(\Gamma)$. We can solve the instance of decision problem OUTCOME(MED) by solving the algebraic model counting problem twice: once for Γ and once for $\Gamma[s]$. The instance is a yes-instance if and only if $\boldsymbol{A}(\Gamma) = \boldsymbol{A}(\Gamma[s])$. By Theorem 8, this can be done in polynomial time.

To make this algorithm work for the case of OUTCOME(MCC), one only needs to adapt the values of $n_{x,0}$ and $n_{x,1}$. Instead of setting $n_{x,0}$ and $n_{x,1}$ to the majority strength of $\neg x$ and x, respectively, we let $n_{x,0} = 0$ if a strict majority of ballots $r \in \boldsymbol{r}$ have that $r(x) = 1$, and we let $n_{x,0} = 1$ otherwise. Similarly, we let $n_{x,1} = 0$ if a strict majority of ballots $r \in \boldsymbol{r}$ have that $r(x) = 0$, and we let $n_{x,0} = 1$ otherwise. □

Representing the integrity constraint as a DNNF circuit makes it possible to perform more tasks efficiently than just the decision problem OUTCOME(F). For example, the algorithms for algebraic model counting can be used to produce a DNNF circuit that represents the set $F(\boldsymbol{r})$ of outcomes, allowing further operations to be carried out efficiently.

Theorem 10. OUTCOME(REV) *is polynomial-time computable when* Γ *is a DNNF circuit.*

Proof (sketch). The polynomial-time algorithm for OUTCOME(REV) is analogous to the algorithm described for OUTCOME(MED) described in the proof of Theorem 9. The only modification that needs to be made to make this algorithm work for OUTCOME(REV) is to adapt the numbers $n_{x,0}$ and $n_{x,1}$, for each $x \in \mathcal{I}$. Instead of identifying these numbers with the majority strength of $\neg x$ and x, respectively, we identify them with the total reversal score of x and $\neg x$, over the profile $\boldsymbol{r}$. That is, we let $n_{x,0} = \sum_{r\in\boldsymbol{r}} s_{\mathrm{R}}(r, \neg x)$ and we let $n_{x,1} = \sum_{r\in\boldsymbol{r}} s_{\mathrm{R}}(r, x)$. For general propositional formulas Γ, the reversal scoring function s_{R} is NP-hard to compute. However, since Γ is given as a DNNF circuit, we can compute the scoring function s_{R}, and thereby $n_{x,0}$ and $n_{x,1}$, in polynomial time—by using another reduction to the problem of algebraic model counting. We omit the details of this latter reduction. □

Intuitively, the results of Theorems 9 and 10 are a consequence of the fact that DNNF circuits allow polynomial-time weighted maximal model computation, and that the judgment aggregation procedures MED, MCC and REV are based on weighted maximal model computation. These results can therefore also straightforwardly be extended to other judgment aggregation procedures that are based on weighted maximal model computation.

Other Results

We can extend some previously established results (Proposition 4 and Theorem 6) to the case of DNNF circuits.

Corollary 11. OUTCOME(RA) *is polynomial-time computable when restricted to the case where* Γ *is a DNNF circuit.*

Corollary 12. OUTCOME(MAXHAM) *is* Θ_2^{p}*-complete when restricted to the case where* Γ *is a DNNF circuit.*

A similar result for YOUNG follows from a result that we will establish in the next section (Proposition 18).

Corollary 13. OUTCOME(YOUNG) *is* Θ_2^{p}*-complete when restricted to the case where* Γ *is a DNNF circuit.*

An overview of the results established so far in this section can be found in Table 4.

F	complexity of OUTCOME(F)	
MED	in P	(Theorem 9)
REV	in P	(Theorem 10)
MCC	in P	(Theorem 9)
YOUNG	Θ_2^{p}-c	(Corollary 13)
MAXHAM	Θ_2^{p}-c	(Corollary 12)
RA	in P	(Corollary 11)

Table 4: The computational complexity of outcome determination for various procedures F restricted to the case where Γ is a DNNF circuit.

A Compilation Approach

The results of Theorems 9 and 10 and Corollary 11 pave the way for another approach towards finding cases where judgment aggregation procedures can be performed efficiently. The idea behind this approach is to compile the integrity constraint into a DNNF circuit—regardless of whether this compilation process enjoys a polynomial-time worst-case performance guarantee. There are several off-the-shelf tools available that compile CNF formulas into DNNF circuits using optimized methods based on SAT solving algorithms (Darwiche 2004; Muise et al. 2012; Oztok and Darwiche 2014b). Since the class of DNNF circuits is expressively complete—i.e., every Boolean function can be expressed using a DNNF circuit—it is possible to compile any integrity constraint Γ into a DNNF circuit C_Γ.

The downside is that the circuit C_Γ could be of exponential size, or it could take exponential time to compute it. However, once the circuit C_Γ is computed and stored in memory, one can use several judgment aggregation procedures efficiently: MED, MCC, REV and RA.

Thus, this approach restricts the computational bottleneck to the *compilation phase*, before any judgments are solicited from the individuals in the judgment aggregation scenario. Once the compilation phase has been completed, there are polynomial-time guarantees on the *aggregation phase* (polynomial in the size of the compiled DNNF circuit C_Γ).

CNF Formulas of Bounded Treewidth

The tractability results for DNNF circuits can be leveraged to get parameterized tractability results for the case where the integrity constraint is a CNF formula with a 'treelike' structure.

Parameterized Complexity Theory & Treewidth In order to explain the results that follow, we briefly introduce some relevant concepts from the theory of parameterized complexity. For more details, we refer to textbooks on the topic (see, e.g., Cygan et al. 2015, Downey and Fellows 2013). The central notion in parameterized complexity is that of *fixed-parameter tractability*—a notion of computational tractability that is more lenient than the traditional notion of polynomial-time solvability. In parameterized complexity running times are measured in terms of the input size n as well as a problem parameter k. Intuitively, the parameter is used to capture structure that is present in the input and that can be exploited algorithmically. The smaller the value of the problem parameter k, the more structure the input exhibits. Formally, we consider *parameterized problems* that capture the computational task at hand as well as the choice of parameter. A parameterized problem Q is a subset of $\Sigma^* \times \mathbb{N}$ for some fixed alphabet Σ. An instance (x, k) of Q contains the problem input $x \in \Sigma^*$ and the parameter value $k \in \mathbb{N}$. A parameterized problem is *fixed-parameter tractable* there is a deterministic algorithm that for each instance (x, k) decides whether $(x, k) \in Q$ and that runs in time $f(k)|x|^c$, where f is a computable function of k, and c is a fixed constant. Algorithms running within such time bounds are called *fpt-algorithms*. The idea behind these definitions is that fixed-parameter tractable running times are scalable whenever the value of k is small.

A commonly used parameter is that of the treewidth of a graph. Intuitively, the treewidth measures the extent to which a graph is like a tree—trees and forests have treewidth 1, cycles have treewidth 2, and so forth. The notion of treewidth is defined as follows. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{T}, (B_t)_{t \in T})$ where $\mathcal{T} = (T, F)$ is a tree and $(B_t)_{t \in T}$ is a family of subsets of V such that:

- for every $v \in V$, the set $B^{-1}(v) = \{\, t \in T : v \in B_t \,\}$ is nonempty and connected in $\mathcal{T}$; and
- for every edge $\{v, w\} \in E$, there is a $t \in T$ such that $v, w \in B_t$.

The *width* of the decomposition $(\mathcal{T}, (B_t)_{t \in T})$ is the number $\max\{\, |B_t| : t \in T \,\} - 1$. The *treewidth* of G is the minimum of the widths of all tree decompositions of G. Let G be a graph and k a nonnegative integer. There is an fpt-algorithm that computes a tree decomposition of G of width k if it exists, and fails otherwise (Bodlaender 1996).

Encoding Results We can then use results from the literature to establish tractability results for computing outcomes of various judgment aggregation procedures for integrity constraints whose variable interactions have a treelike structure. Let $\Gamma = c_1 \wedge \cdots \wedge c_m$ be a CNF formula. The *incidence graph of* Γ is the graph (V, E), where $V = \mathrm{Var}(\Gamma) \cup \{c_1, \ldots, c_u\}$ and $E = \{\, \{c_j, x\} : 1 \leq j \leq m, x \in \mathrm{Var}(\Gamma), x \text{ occurs in the clause } c_j \,\}$. The *incidence treewidth of* Γ is defined as the treewidth of the incidence graph of Γ.

We can leverage the results of Theorems 9 and 10 and Corollary 11 to get fixed-parameter tractability results for computing outcomes of MED, MCC, REV and RA for integrity constraints with small incidence treewidth.

Proposition 14 (Oztok and Darwiche 2014a, Bova et al. 2015)**.** *Let Γ be a CNF formula of incidence treewidth k. Constructing a DNNF circuit Γ' that is equivalent to Γ can be done in fixed-parameter tractable time.*

Corollary 15. *The problems* OUTCOME(MED)*,* OUTCOME(MCC)*,* OUTCOME(REV) *and* OUTCOME(RA) *are fixed-*

parameter tractable when parameterized by the incidence treewidth of Γ.

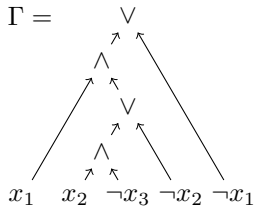
Case Study: Budget Constraints

In this section, we illustrate how the results of the previous section can contribute to providing a computational complexity analysis for an application setting. The setting that we consider as an example is that of budget constraints. This setting is closely related to that of *Participatory Budgeting* (see, e.g., Benade et al. 2017), where citizens propose projects and vote on which projects get funded by public money. In the setting that we consider, each issue $x \in \mathcal{I}$ represents whether or not some measure is implemented. Each such measure has an implementation cost c_x associated with it. Moreover, there is a total budget B that cannot be exceeded—that is, each ballot (individual or collective) can set a set of variables x to true such that the cumulative cost of these variables is at most B (and set the remaining variables to false). The integrity constraint Γ encodes that the total budget B cannot be exceeded by the total cost of the variables that are set to true. (For the sake of simplicity, we assume that all costs and the total budget are all positive integers.)

The concepts and tools from judgment aggregation are useful and relevant in this setting. This is witnessed, for instance, by the fact that simply taking a majority vote will not always lead to a suitable collective outcome. Consider the example where there are three measures that are each associated with cost 1, and where there is a budget of 2. Moreover, suppose that there are three individuals. The first individual votes to implement measures 1 and 2; the second votes for measures 1 and 3, and the third for 2 and 3. Each of the individuals' opinions is consistent with the budget. However, taking a majority measure-by-measure vote results in implementing all three issues, which exceeds the budget. (In other words, the individual opinions r_1, r_2, r_3 are all rational, whereas the collective majority opinion $m_{\boldsymbol{r}}$ is not.) This example is illustrated in Figure 2—in this figure, we encode the budget constraint using a DNNF circuit Γ.

$\boldsymbol{r}$	x_1	x_2	x_3
r_1	1	1	0
r_2	1	0	1
r_3	0	1	1
$\textsc{maj}(\boldsymbol{r})$	1	1	1

(a) The profile $\boldsymbol{r}$



(b) The integrity constraint Γ

Figure 2: Example of an aggregation scenario with a budget constraint (for $B = 2$ and $c_x = 1$ for all $x \in \mathcal{I}$), where the budget constraint is represented as a DNNF circuit Γ.

Encoding into a Polynomial-Size DNNF Circuit

To use the framework of judgment aggregation to model settings with budget constraints, we need to encode budget constraints using integrity constraints Γ. One can do this in several ways. We consider an encoding using DNNF circuits (as in Figure 2b). Let $\mathcal{I}$ be a set of issues, let $\{c_x\}_{x\in\mathcal{I}}$ be a vector of implementation costs, and let $B \in \mathbb{N}$ be a total budget. We say that an integrity constraint Γ encodes the budget constraint for $\{c_x\}_{x\in\mathcal{I}}$ and B if for each complete ballot $r : \mathcal{I} \to \{0, 1\}$ it holds that r satisfies Γ if and only if $\sum_{x\in\mathcal{I},r(x)=1} c_x \leq B$.

We can encode budget constraints efficiently using DNNF circuits by expressing them as binary decision diagrams. A *binary decision diagram (BDD)* is a particular type of NNF circuit. Let Γ be an NNF circuit. We say that a node N of Γ is a *decision node* if (i) it is a leaf or (ii) it is a disjunction node expressing $(x\wedge\alpha)\vee(\neg x\wedge\beta)$, where $x \in \mathrm{Var}(\Gamma)$ and α and β are decision nodes. A binary decision diagram is an NNF circuit whose root is a decision node. A *free binary decision diagram (FBDD)* is a BDD that satisfies decomposability (see, e.g., Darwiche and Marquis 2002, Gergov and Meinel 1994).

Theorem 16. *For each $\mathcal{I}$, $\{c_x\}_{x\in\mathcal{I}}$ and B, we can construct a DNNF circuit Γ encoding the budget constraint for $\{c_x\}_{x\in\mathcal{I}}$ and B in time polynomial in $B + |\mathcal{I}|$.*

Proof. We construct an FBDD Γ encoding the budget constraint for $\{c_x\}_{x\in\mathcal{I}}$ and B as follows. Without loss of generality, suppose that $c_x > 0$ for each $x \in \mathcal{I}$. Let $\mathcal{I} = \{x_1, \ldots, x_n\}$. We introduce a decision node $N_{i,j}$ for each $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, B\}$. Take arbitrary $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, B\}$. If $i = n$, we let $N_{i,j} = \top$. If $i < n$, we distinguish two cases: either (i) $j' \leq B$ or (ii) $j' > B$, where $j' = j + c_{x_i}$.. In case (i), we let $N_{i,j} = (x_i \wedge N_{i+1,j'}) \vee (\neg x_i \wedge N_{i+1,j})$. In case (ii), we let $N_{i,j} = (x_i \wedge \bot) \vee (\neg x_i \wedge N_{i+1,j})$. We let the root of the FBDD be the node $N_{0,0}$—and we remove all nodes that are not descendants of $N_{0,0}$. Intuitively, the subcircuit rooted at $N_{i,j}$ represents all truth assignments to the variables $x_{i+1}, \ldots, x_n$ that fit within a budget of $B - j$. For each node $N_{i,j}$ it holds that the variables in the leaves reachable from $N_{i,j}$ are among $x_{i+1}, \ldots, x_n$. Therefore, we constructed an FBDD. Moreover, each complete ballot r satisfies the circuit Γ if and only if $\sum_{x\in\mathcal{I},r(x)=1} c_x \leq B$. Thus, Γ is a DNNF circuit constructed in time polynomial in $B + |\mathcal{I}|$ encoding the budget constraint for $\{c_x\}_{x\in\mathcal{I}}$ and B. □

An example of a DNNF circuit resulting from the encoding described in the proof of Theorem 16—after some simplifications—can be found in Figure 2b.

Complexity Results

Using the encoding result of Theorem 16, we can establish polynomial-time solvability results for computing outcomes for several judgment aggregation procedures in the setting of budget constraints.

Corollary 17. OUTCOME(MED), OUTCOME(MCC), OUTCOME(REV), *and* OUTCOME(RA) *are polynomial-time computable when restricted to the case where Γ expresses a budget constraint.*

Proof. The result follows from Theorems 9, 10 and 16, and Corollary 11. □

For the YOUNG and MAXHAM procedures, we obtain intractability results for the case of budget constraints—for both procedures computing outcomes is Θ_2^{p}-hard.

Proposition* 18. OUTCOME(YOUNG) *is Θ_2^{p}-hard when restricted to the case where Γ expresses a budget constraint.*

Corollary 19. OUTCOME(MAXHAM) *is Θ_2^{p}-hard when restricted to the case where Γ expresses a budget constraint.*

Proof. The result follows directly from Proposition 4. □

An overview of the complexity results that we established in this section can be found in Table 5.

F	complexity of OUTCOME(F)	
MED	in P	(Corollary 17)
REV	in P	(Corollary 17)
MCC	in P	(Corollary 17)
YOUNG	Θ_2^{p}-c	(Proposition 18)
MAXHAM	Θ_2^{p}-c	(Corollary 19)
RA	in P	(Corollary 17)

Table 5: The computational complexity of outcome determination for various procedures F restricted to the case where Γ is a budget constraint.

Directions for Future Research

In this paper, we provided a set of initial results for restricted languages for judgment aggregation, but these results are only the tip of the iceberg that is to be explored. We outline some directions for interesting future work on this topic.

One first direction is to establish the complexity of OUTCOME(F) for cases that are left open in this paper—for example, for YOUNG and REV for the case of Krom and (definite) Horn formulas. Another direction is to pinpoint the complexity of OUTCOME(F) for the languages that we considered for other judgment aggregation rules studied in the literature (see, e.g., Lang et al. 2017).

Yet another direction is to extend tractability results obtained in this paper—e.g., for Krom and Horn formulas—to formulas that are 'close' to Krom or Horn formulas. One could use the notion of backdoors for this (see, e.g., Gaspers and Szeider 2012).

Finally, further restricted languages of propositional formulas or Boolean circuits need to be studied, to get a more complete picture of where the boundaries of the expressivity-tractability balance lie in the setting of judgment aggregation. A good source for additional languages is the field of knowledge compilation (see, e.g., Darwiche and Marquis 2002, Darwiche 2014, Marquis 2015), where many restricted languages have been studied with respect to their expressivity and support for performing various operations tractably.

Conclusion

In this paper, we initiated the hunt for representation languages for the setting of judgment aggregation that strike a balance between (1) allowing relevant computational tasks to be performed efficiently and (2) being expressive enough to model interesting and relevant application settings. Concretely, we considered Krom and (definite) Horn formulas, and we studied the class of Boolean circuits in DNNF. We studied the impact of these languages on the complexity of computing outcomes for a number of judgment aggregation procedures studied in the literature. Additionally, we illustrated the use of these languages for a specific application setting: voting on how to spend a budget.

Appendix: Preliminaries

We give an overview of some notions from propositional logic and computational complexity that we use in the paper.

Propositional Logic

Propositional formulas are constructed from propositional variables using the Boolean operators $\wedge, \vee, \rightarrow$, and $\neg$. A *literal* is a propositional variable x (a *positive literal*) or a negated variable $\neg x$ (a *negative literal*). A *clause* is a finite set of literals, not containing a complementary pair $x, \neg x$, and is interpreted as the disjunction of these literals. A formula in *conjunctive normal form (CNF)* is a finite set of clauses, interpreted as the conjunction of these clauses. For each $r \geq 1$, an *r-clause* is a clause that contains at most r literals, and rCNF denotes the class of all CNF formulas consisting only of r-clauses. 2CNF is also denoted by KROM, and 2CNF formulas are also known as *Krom formulas*. A *Horn clause* is a clause that contains at most one positive literal. A *definite Horn clause* is a clause that contains exactly one positive literal. We let HORN denote the class of all CNF formulas that contain only Horn clauses (*Horn formulas*), and we let DEFHORN denote the class of all CNF formulas that contain only definite Horn clauses (*definite Horn formulas*).

For a propositional formula φ, $\mathrm{Var}(\varphi)$ denotes the set of all variables occurring in φ. Moreover, for a set X of variables, $\mathrm{Lit}(X)$ denotes the set of all literals over variables in X, i.e., $\mathrm{Lit}(X) = \{ x, \neg x : x \in X \}$. We use the standard notion of *(truth) assignments* $\alpha : \mathrm{Var}(\varphi) \rightarrow \{0, 1\}$ for Boolean formulas and *truth* of a formula under such an assignment. For any formula φ and any truth assignment α, we let $\varphi[\alpha]$ denote the formula obtained from φ by instantiating variables s in the domain of α with $\alpha(x)$ and simplifying the formula accordingly. By a slight abuse of notation, if α is defined on all $\mathrm{Var}(\varphi)$, we let $\varphi[\alpha]$ denote the truth value of φ under α.

Computational Complexity Theory

We assume the reader to be familiar with the complexity classes P and NP, and with basic notions such as polynomial-time reductions. For more details, we refer to textbooks on computational complexity theory (see, e.g., Arora and Barak 2009).

In this paper, we also refer to the complexity classes Θ_2^{p} and Δ_2^{p} that consist of all decision problems that can be solved by a polynomial-time algorithm that queries an NP oracle $O(\log n)$ or $n^{O(1)}$ times, respectively.

Acknowledgments. This work was supported by the Austrian Science Fund (FWF), project J4047.

References

Arora, S., and Barak, B. 2009. *Computational Complexity – A Modern Approach*. Cambridge University Press.

Benade, G.; Nath, S.; Procaccia, A. D.; and Shah, N. 2017. Preference elicitation for participatory budgeting. In *Proc. of the 31st AAAI Conf. on Artificial Intelligence (AAAI 2017)*, 376–382. AAAI Press.

Bodlaender, H. L. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.* 25(6):1305–1317.

Bova, S.; Capelli, F.; Mengel, S.; and Slivovsky, F. 2015. On compiling CNFs into structured deterministic DNNFs. In *Proc. of the 18th Intern. Conf. on Theory and Applications of Satisfiability Testing (SAT 2015)*, 199–214.

Brams, S. J.; Kilgour, D. M.; and Sanver, M. R. 2004. A minimax procedure for negotiating multilateral treaties. In *Proc. of the 2004 Annual Meeting of the American Political Science Association*.

Cadoli, M.; Donini, F. M.; Liberatore, P.; and Schaerf, M. 2002. Preprocessing of intractable problems. *Inf. Comput.* 176(2):89–120.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *J. Artif. Intell. Res.* 17:229–264.

Darwiche, A. 2004. New advances in compiling CNF into decomposable negation normal form. In de Mántaras, R. L., and Saitta, L., eds., *Proc. of the 16th European Conf. on Artificial Intelligence, (ECAI 2004)*, 328–332. IOS Press.

Darwiche, A. 2014. Tractable knowledge representation formalisms. In Bordeaux, L.; Hamadi, Y.; and Kohli, P., eds., *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press. 141–172.

Dietrich, F., and List, C. 2007. Arrow's theorem in judgment aggregation. *Social Choice and Welfare* 29(1):19–33.

Dietrich, F. 2007. A generalised model of judgment aggregation. *Social Choice and Welfare* 28(4):529–565.

Downey, R. G., and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Springer Verlag.

Endriss, U., and de Haan, R. 2015. Complexity of the winner determination problem in judgment aggregation: Kemeny, Slater, Tideman, Young. In *Proc. of the 14th Intern. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2015)*.

Endriss, U.; Grandi, U.; de Haan, R.; and Lang, J. 2016. Succinctness of languages for judgment aggregation. In *Proc. of the 15th Intern. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2016)*. AAAI Press.

Endriss, U.; Grandi, U.; and Porello, D. 2012. Complexity of judgment aggregation. *J. Artif. Intell. Res.* 45:481–514.

Endriss, U. 2016. Judgment aggregation. In Brandt, F.; Conitzer, V.; Endriss, U.; Lang, J.; and Procaccia, A., eds., *Handbook of Computational Social Choice*. Cambridge University Press, Cambridge.

Gaspers, S., and Szeider, S. 2012. Backdoors to satisfaction. In Bodlaender, H. L.; Downey, R.; Fomin, F. V.; and Marx, D., eds., *The Multivariate Algorithmic Revolution and Beyond*, 287–317. Springer Verlag.

Gergov, J., and Meinel, C. 1994. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers* 43(10):1197–1209.

Grandi, U., and Endriss, U. 2013. Lifting integrity constraints in binary aggregation. *Artificial Intelligence* 199:45–66.

Grandi, U. 2012. *Binary Aggregation with Integrity Constraints*. Ph.D. Dissertation, University of Amsterdam.

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

de Haan, R., and Slavkovik, M. 2017. Complexity results for aggregating judgments using scoring or distance-based procedures. In *Proc. of the 16th International Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2017)*.

de Haan, R. 2018. Hunting for tractable languages for judgment aggregation. *CoRR* abs/1808.03043. https://arxiv.org/abs/1808.03043.

Kimmig, A.; Van den Broeck, G.; and De Raedt, L. 2017. Algebraic model counting. *J. of Applied Logic* 22:46–62.

Lang, J., and Slavkovik, M. 2014. How hard is it to compute majority-preserving judgment aggregation rules? In *Proc. of the 21st European Conf. on Artificial Intelligence (ECAI 2014)*. IOS Press.

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

Li, M.; Ma, B.; and Wang, L. 2002. On the closest string and substring problems. *J. of the ACM* 49(2):157–171.

List, C., and Pettit, P. 2002. Aggregating sets of judgments: An impossibility result. *Economics and Philosophy* 18(1):89–110.

Marquis, P. 2015. Compile! In Bonet, B., and Koenig, S., eds., *Proc. of the 29th AAAI Conf. on Artificial Intelligence (AAAI 2015)*, 4112–4118. AAAI Press.

Muise, C. J.; McIlraith, S. A.; Beck, J. C.; and Hsu, E. I. 2012. Dsharp: Fast d-DNNF compilation with sharpSAT. In Kosseim, L., and Inkpen, D., eds., *Proc. of the 25th Canadian Conf. on Artificial Intelligence (Canadian AI 2012)*, 356–361. Springer Verlag.

Oztok, U., and Darwiche, A. 2014a. CV-width: A new complexity parameter for CNFs. In *Proc. of the 21st European Conf. on Artificial Intelligence (ECAI 2014)*, 675–680. IOS Press.

Oztok, U., and Darwiche, A. 2014b. On compiling CNF into decision-DNNF. In *Proc. of the 20th Intern. Conf. on Principles and Practice of Constraint Programming (CP 2014)*, 42–57. Springer Verlag.

Rothe, J. 2016. *Economics and Computation*. Springer.