Unravelling Expressive Delegations: Complexity and Normative Analysis

Giannis Tyrovolas[1], **Andrei Constantinescu**[2], **Edith Elkind**[3]

[1]Independent
[2]ETH Zurich
[3]University of Oxford
johntyro@hotmail.com, aconstantine@ethz.ch, elkind@cs.ox.ac.uk

Abstract

We consider binary group decision-making under a rich model of liquid democracy recently proposed by Colley, Grandi, and Novaro (2022): agents submit ranked delegation options, where each option may be a function of multiple agents' votes; e.g., "I vote yes if a majority of my friends vote yes." Such ballots are unravelled into a profile of direct votes by selecting one entry from each ballot so as not to introduce cyclic dependencies. We study delegation via monotonic Boolean functions, and two unravelling procedures: MINSUM, which minimises the sum of the ranks of the chosen entries, and its egalitarian counterpart, MINMAX. We provide complete computational dichotomies: MINSUM is hard to compute (and approximate) as soon as any non-trivial functions are permitted, and polynomial otherwise; for MINMAX the easiness results extend to arbitrary-arity logical ORs and ANDs taken in isolation, but not beyond. For the classic model of delegating to individual agents, we give asymptotically near-tight algorithms for carrying out the two procedures, and efficient algorithms for finding optimal unravellings with the highest vote count for a given alternative. These algorithms inspire novel tie-breaking rules for the setup of voting to change a status quo. We then introduce a new axiom, which can be viewed as a variant of the participation axiom, and use algorithmic techniques developed earlier in the paper to show that it is satisfied by MINSUM and a lexicographic refinement of MINMAX (but not MINMAX itself).

1 Introduction

Liquid democracy, also referred to as delegative democracy, is a decision-making mechanism that allows the voters more flexibility than representative democracy: While every voter may vote directly on an issue, voters may also delegate their vote to other voters. Crucially, delegations are *transitive*: if Alice delegates to Bob, and Bob delegates to Claire, then Claire votes with the combined power of all three.

Variants of this idea have been explored by many authors; see, e.g., the survey by Behrens (2017). In particular, Ford (2002) argues that large-scale direct democracy is infeasible and likely undesirable, whereas representative democracy is rather rigid: It requires holding elections every so often, with winners representing their entire constituency and losers gaining no representative power. Further, there is a hard limit on the total number of elected representatives, and every voter can pick from a limited number of candidates. Delegative democracy can then be seen as a balance of the two, by challenging the premise that the number of representatives needs to be kept small. In this model, voters can vote directly on issues if they wish to do so. Passive voters can delegate to representatives called *delegates*. Delegates are not chosen at fixed time points, but need to canvass voters continually, and acquire the combined power of all voters that delegate to them. They can vote on issues directly, or delegate to other delegates. Importantly, delegates need not win competitive elections, and their votes are public for the sake of accountability. Ford introduces several strengthenings of liquid democracy, such as voters delegating fractions of their votes to different delegates. A particularly interesting variant is to allow agents to submit "multiple delegation choices in order of preference." This is done partly to deal with the case of cycles, where Alice delegates to Bob, but also Bob delegates to Alice.

Colley, Grandi, and Novaro (2022) further extend this model, by allowing each delegation choice to be an arbitrary function of other agents' selections. That is, instead of delegating to Bob, Alice can declare that she supports option X if at least one of Bob, Charlie and Diana supports it (for a formal treatment of this model, see Section 2). An agent's ballot is then a ranked list of such functions, in order of preference. This enhancement of the basic model means that cycles can be introduced in much more complicated ways.

Colley et al. propose two "unravelling" procedures to resolve these cycles. MINSUM is a utilitarian method that minimises the sum of preference levels used, and MINMAX is an egalitarian method that minimises the maximum of the preference levels used. They prove that MINSUM is NP-hard if agents are allowed to use arbitrary-arity logical ANDs and MINMAX is NP-hard for arbitrary Boolean functions. Also, they give polynomial-time algorithms to unravel instances where agents can only delegate directly to other agents.

1.1 Our Contribution

We focus on binary issues and strengthen the complexity results of Colley et al. to complete computational dichotomies: MINSUM is hard to compute (and approximate) as soon as any non-trivial functions are permitted, and polynomial otherwise; for MINMAX the easiness results extend to arbitrary-arity logical ORs and ANDs taken in isolation, but not beyond. For the standard model of delegation to in-

dividual agents, we design algorithms that are faster than those of Colley et al. (indeed, nearly optimal). Then, we consider the problem of determining if there exists an unravelling where a particular alternative is selected, giving efficient algorithms in both cases. For MINSUM, we do this by leveraging Fulkerson's primal-dual algorithm for computing minimum-cost arborescences (Fulkerson 1974). Our results suggest novel tie-breaking rules for both unravelling procedures, which are particularly appealing when one of the two input alternatives is seen as a status quo. Interestingly, Fulkerson's ideas also turn out to be useful from a normative perspective: we put forward a new axiom, which can be seen as an adaptation of the classic participation axiom to the setting of liquid democracy, and use our algorithmic techniques to show that it is satisfied by MINSUM and a lexicographic refinement of MINMAX, but not MINMAX itself.

1.2 Related Work

The formal model of liquid democracy with ranked delegations was put forward by Kotsialou and Riley (2020), who were interested in identifying unravelling procedures that avoid cycles and have good normative properties. They study the properties of breadth-first search and depth-first search in this context. Their model inspired a number of recent papers (Colley, Grandi, and Novaro 2022; Brill et al. 2022; Kavitha et al. 2020; Utke and Schmidt-Kraepelin 2023); in particular, the work of Colley et al. extends the model of Kotsialou and Riley (2020) to more expressive ballots and is the direct predecessor of our work. Brill et al. (2022) define the concept of delegation graphs, for which unravelling procedures are called delegation rules. They study two important subclasses: sequence rules and confluent rules; the latter corresponding to selecting an arborescence of the graph. MINSUM and MINMAX are both confluent rules. Brill et al. provide a wide range of normative and experimental results. Of importance to us, MINSUM,[1] corresponding to minimum-cost arborescences, satisfies two attractive axioms—guru-participation and copy-robustness (the latter is similar to our new axiom)—if tie-breaking is performed lexicographically with respect to a fixed ordering of the agents. In contrast, our axiom is formulated for irresolute MINSUM, and can be satisfied by a resolute anonymous rule (i.e., treating all agents symmetrically), at the expense of neutrality. Moreover, building on the popular arborescences framework of Kavitha et al. (2020), the authors experimentally show that on most instances a majority of voters prefer MINSUM arborescences to other arborescences.

There is also a substantial body of work on liquid democracy in the setting with ground truth (Kahng, Mackenzie, and Procaccia 2021; Gölz et al. 2021; Caragiannis and Micha 2019; Becker et al. 2021; Alouf-Heffetz et al. 2022), as well as on game-theoretic aspects of delegation (Bloembergen, Grossi, and Lackner 2019; Escoffier, Gilbert, and Pass-Lanneau 2020; Zhang and Grossi 2021).

[1]Called BordaBranching in their paper to distinguish it from a different rule called MinSum.

2 Preliminaries

A finite set of agents/voters $N = \{1, \dots, n\}$ aim to select an outcome from a finite set D of alternatives. For most of our results, we focus on the *binary* setting $D = \{0, 1\}$.

2.1 Classic Liquid Democracy

We first describe the classic model of liquid democracy with ranked delegations; then, in the next section, we will delve into its generalisation proposed by Colley et al. (2022). We will refer to the latter model simply as *liquid democracy* whenever clear from the context.

The decision-making process proceeds as follows. First, each agent $a \in N$ casts an ordered ballot with size k_a: $B_a = B_a(0) \succ \dots \succ B_a(k_a - 1) \succ \tau_a$, where $B_a(i) \in N \setminus \{a\}$ for each $i = 0, \dots, k_a - 1$ and $\tau_a \in D$. We require $B_a(i) \neq B_a(j)$ for $0 \leq i < j < k_a$. For brevity, we write $B_a(k_a) = \tau_a$. Intuitively, agent a wants to delegate their vote first to $B_a(0)$, as a second option to $B_a(1)$, and so on; their final backup option is to vote directly for $\tau_a \in D$. When $k_a = 0$, the ballot of agent a is $B_a = \tau_a$, which indicates that a will vote directly for τ_a. After the ballots are cast, the electoral authority *unravels* the preferences by turning the collection of ballots $\mathbf{B} = (B_a)_{a \in N}$ into a collection of *concrete* votes $\mathbf{X} = (X_a)_{a \in N} \in D^N$. Then, a deterministic aggregation rule $agg : D^N \to D$ is applied to $\mathbf{X}$ to compute the final decision $agg(\mathbf{X})$. We will sometimes work with *partial* collections of concrete votes $\mathbf{X} \in (D \cup \{\perp\})^N$ where some entries are unspecified (as indicated by $X_a = \perp$).

To unravel the preferences, one can select, for each agent $a \in N$, a position $0 \leq c_a \leq k_a$ from their ballot and use the respective entry $B_a(c_a)$ to compute the concrete votes. This approach has the attractive feature of explainability: The collection of positions $\mathbf{c} = (c_a)_{a \in N}$ forms a *certificate*, which can be reported together with the electoral outcome as an explanation for it. Unravelling procedures that work in this fashion are called *confluent* (Brill et al. 2022). They provide a high degree of accountability, as each agent can see whether their direct vote τ_a was used, or which other agent their vote was delegated to.

Naturally, not all certificates are acceptable: e.g., if Alice and Bob want to delegate to each other as their first preference, the certificate $\mathbf{c} = (0, 0)$ leads to a delegation cycle, and hence to no concrete votes being computed for Alice and Bob. Following the terminology of Colley et al. (2022), we call such certificates *inconsistent*. It is convenient to formalise this concept in the language of graphs.

A tuple $(N, D, (B_a)_{a \in N})$ can be seen as a directed weighted graph called the *delegation graph*. Its vertex set is $N \cup D \cup \{r\}$, where r is a special vertex (the *root*). There is an edge $d \to r$ of weight 0 for each $d \in D$ as well as an edge $a \to B_a(i)$ of weight/cost[2] i for each $a \in A$ and $0 \leq i \leq k_a$. A certificate $\mathbf{c}$ is *consistent* if the set of edges $T_{\mathbf{c}} = \{(a, B_a(c_a)) \mid a \in A\} \cup \{(d, r) \mid d \in D\}$ is an *r-arborescence* (i.e., a directed spanning tree with sink r) of the delegation graph. In particular, every node can reach r along edges in $T_{\mathbf{c}}$, so there are no delegation cycles. For

[2]We use these terms interchangeably.

brevity, we shorten r-arborescence to simply arborescence. A consistent certificate $\mathbf{c}$ induces concrete votes $\mathbf{X}$ as follows: $X_a = d$ whenever d is on the a–r path in $T_{\mathbf{c}}$; we say that agent $a \in N$ *votes* $d \in D$ in $\mathbf{c}$.

When $\mathbf{B}$ admits multiple consistent certificates, it is natural to prefer ones selecting options that are ranked as highly as possible in the ballots. In particular, one can select the certificates that minimise $\sum_{a \in N} c_a$ or $\max_{a \in N} c_a$, i.e., maximise the *utilitarian* or *egalitarian* social welfare. We call such certificates MINSUM and MINMAX certificates, and extend this terminology to unravelling procedures that select certificates optimising the respective quantities. Notice that these procedures are irresolute, as a minimiser of $\sum_{a \in N} c_a$ or $\max_{a \in N} c_a$, need not be unique. In graph-theoretic terms, MINSUM and MINMAX certificates correspond to, respectively, minimum weight and minimum bottleneck arborescences. Given a (potentially irresolute) unravelling procedure F, e.g., MINSUM or MINMAX, and ballots $\mathbf{B}$, we write $F(\mathbf{B})$ for the set of collections of concrete votes $\mathbf{X}$ that can be output by F when executed with input $\mathbf{B}$. When F is resolute, we simply write $F(\mathbf{B}) = \mathbf{X}$ instead of $F(\mathbf{B}) = \{\mathbf{X}\}$.

2.2 Enhanced Liquid Democracy

We now present *smart voting*,[3] as defined by Colley et al. (2022): the extension of the classic delegation model that allows agents to delegate to complex functions of the other agents. For readability, we limit ourselves to the binary setting. We first take a short detour to introduce the notions of Boolean logic needed for our examples and results.

A *k-ary Boolean function* $f\colon \{0,1\}^k \to \{0,1\}$ is a mapping from k-tuples of binary values to binary values. Function f is *monotonic* if for all k-tuples $\mathbf{x}, \mathbf{y}$ such that $x_i \leq y_i$ for $1 \leq i \leq k$ it holds that $f(\mathbf{x}) \leq f(\mathbf{y})$. Two k-ary Boolean functions f and g are equal if they are *extensionally equal*, i.e., $f(\mathbf{x}) = g(\mathbf{x})$ for any k-tuple $\mathbf{x}$. Given a k-ary Boolean function f, substituting values for ℓ of its arguments yields a k'-ary function for $k' = k - \ell$. In a slight abuse of notation, we use 0 and 1 to denote the two constant functions (of arity zero). We express Boolean functions via *Boolean formulas* using connectives such as $\wedge$, $\vee$ and $\neg$; we often write $\neg x$ as $\overline{x}$. For instance, the ternary majority function $\textsc{Maj}(x, y, z)$ can be written as $(x \wedge y) \vee (y \wedge z) \vee (z \wedge x)$. When a Boolean function is monotonic, it can be expressed by a *monotone* Boolean formula, i.e., without negations.

We will work with n-ary Boolean functions with arguments indexed by the set of agents N, i.e., functions $f : \{0,1\}^N \to \{0,1\}$. When a function does not require some of its arguments, we may omit them, and consider an "equivalent" function of smaller arity. Formally, if f only depends on its arguments indexed by $S \subsetneq N$ then we will view it as $f' : \{0,1\}^S \to \{0,1\}$. Conversely, in contexts requiring it, a function $f' : \{0,1\}^S \to \{0,1\}$ denotes the function $f : \{0,1\}^N \to \{0,1\}$ with redundant arguments added for $N \setminus S$. Given a (possibly partial) list of concrete votes $\mathbf{X} \in \{0,1,\bot\}^N$, we write $f(\mathbf{X})$ for the function obtained from f by substituting $a \mapsto X_a$ for all $a \in N$ with $X_a \neq \bot$.

[3]Not to be confused with the Swiss voting advice application `www.smartvote.ch`.

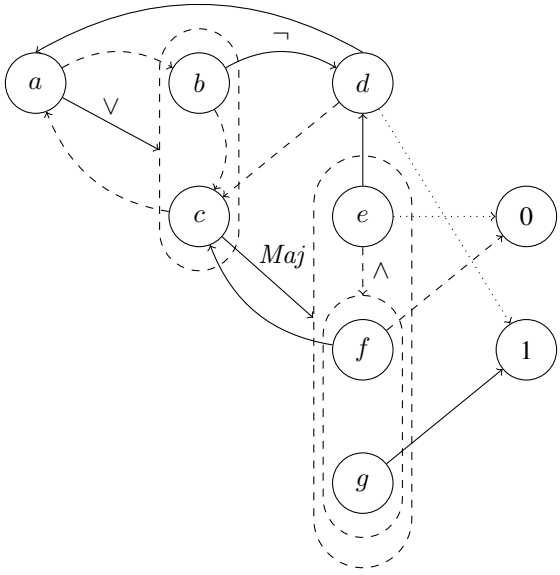


Figure 1: Preference profile in Example 1. Solid lines indicate first preferences, dashed lines indicate second preferences, and dotted lines indicate third preferences. To avoid clutter we have removed the third preferences of a, b and c.

In the enhanced liquid democracy model of Colley et al. (2022), for each $a \in N$ and $0 \leq i < k_a$ the i-th entry $B_a(i)$ in agent a's ballot is a Boolean function with arguments indexed by $N \setminus \{a\}$. The final entries $B_a(k_a) = \tau_a$ naturally correspond to one of the two constant functions.

Example 1. Consider agents $N = \{a, b, c, d, e, f, g\}$ casting ballots $\mathbf{B} = (B_a)_{a \in N}$ as follows (Figure 1):

$$B_a = b \vee c \succ b \succ 0 \qquad B_b = \overline{d} \succ c \succ 1$$
$$B_c = \mathit{Maj}\,(e, f, g) \succ a \succ 1 \qquad B_d = a \succ c \succ 1$$
$$B_e = d \succ f \wedge g \succ 0 \qquad B_f = c \succ 0$$
$$B_g = 1$$

The enhanced model generalises classic liquid democracy in that the latter only allows projection functions (direct delegations) and constant functions (direct votes). Most definitions from the classic model extend in a straightforward way, but a few important points need to be discussed. First, what language is used to express Boolean functions? Note that the validation check $B_a(i) \neq B_a(j)$ for all $0 \leq i < j \leq k_a$[4] has to be replaced with (extensional) function equality: a simple textual comparison would not suffice, as a Boolean function can be expressed by multiple Boolean formulas. We want the function-definition language to allow for this check to be performed efficiently, i.e., in polynomial time; hence, the language of arbitrary Boolean formulas is not suitable. Perhaps most importantly, the definition of the delegation graph does not extend to the enhanced model, and hence we need a new definition of a consistent certificate. As the enhanced model generalises the classic model, this definition

[4]Note the last $\leq$ instead of $<$.

should coincide with the classic one for instances that only use direct delegations and constants. It will be convenient to use the following reformulation of consistency for the classic model (the proof follows by considering arborescence traversals where vertices occur after their parents).

Lemma 2. In the classic model, a certificate $\mathbf{c}$ is consistent iff there exists a linear order $\lhd$ on N such that for all agents $a \in N$ it holds that either $B_a(c_a) \in D$ or $B_a(c_a) \lhd a$.

In other words, a certificate is consistent iff the concrete votes can be computed by starting with $\mathbf{X} = (\bot, \dots, \bot)$ and at each step picking an agent $a \in N$ with $X_a = \bot$ and resolving their vote, i.e., setting X_a either to $B_a(c_a) \in D$ or to $X_{B_a(c_a)}$; for the latter choice we need $X_{B_a(c_a)} \neq \bot$, i.e., $B_a(c_a) \lhd a$, which should be read as "the vote of $B_a(c_a)$ is resolved before that of a". Armed with the idea that "concrete votes can be resolved sequentially," we extend the definition of consistency to the enhanced model as follows.

In the enhanced model, we say that a certificate $\mathbf{c}$ is consistent iff there is a linear order $\lhd$ on N that is *valid*, in the sense that the concrete votes can be computed in the following way. We start with $\mathbf{X} = (\bot, \dots, \bot)$ and go through the agents in order given by $\lhd$. Whenever an agent $a \in N$ is reached, let $f = B_a(c_a)$ be their assigned function in the certificate. Then it should be the case that f simplifies to one of the two constant functions 0 or 1 when the values known for agents that appear earlier in $\lhd$ are substituted in. Formally, either $f(\mathbf{X}) = 0$ or $f(\mathbf{X}) = 1$ should hold, where equality denotes extensional equality. We set $X_a \leftarrow f(\mathbf{X})$ and move on to the next agent in order. That is, the way in which functions are evaluated is not unlike lazy function evaluation in some programming languages.

For instance, let $N = \{a, b, c\}$. Suppose that the function $f = B_a(c_a)$ of agent a assigned by the certificate is $b \vee c$, and consider the order $b \lhd a \lhd c$. If, when a is reached in the order, we have $\mathbf{X} = (\bot, 1, \bot)$, we can compute $f(\mathbf{X}) = f(\bot, 1, \bot) = 1 \vee c = 1$, even though the vote of c is still unknown. However, for $\mathbf{X} = (\bot, 0, \bot)$ the order $\lhd$ is not valid: $f(\mathbf{X}) = f(\bot, 0, \bot) = 0 \vee c = c$ is not extensionally equal to a constant function.

Crucially, the computed concrete votes $\mathbf{X}$ do not depend on the choice of the valid ordering $\lhd$: This is because every assignment performed is, in a sense, "forced."[5] Hence, for a consistent certificate, the concrete votes $\mathbf{X}$ are uniquely determined and are the unique solution to the system of simultaneous equations $X_a = B_a(c_a)(\mathbf{X})$ for all $a \in N$. One can ask whether the converse is also true. Namely, given a certificate $\mathbf{c}$ such that the system of simultaneous equations has a unique solution (i.e., the concrete votes are uniquely determined), is it consistent? If this were the case, this would be an argument in favor of consistent certificates as an attractive ground notion. The next theorem shows that is indeed true when delegations are restricted to monotonic functions (see the appendix for the proof, and all other omitted proofs), and we will argue that there are strong arguments for using monotonic functions from a normative perspective.

[5]To decide if a valid $\lhd$ exists it suffices to attempt to construct it iteratively, by picking, at each step, a new agent a whose vote can be determined based on the previous ones.

Theorem 3. *Assuming delegation is only permitted to monotonic functions, a certificate* $\mathbf{c}$ *is consistent iff the system of simultaneous equations* $X_a = B_a(c_a)(\mathbf{X})$ *for all* $a \in N$ *has a unique solution.*

We now return to Example 1 to illustrate the new definitions, together with MINSUM and MINMAX, whose definitions remain unchanged.

Example 1 (continued)**.** Observe that $\mathbf{c} = (0, \dots, 0)$ is not a consistent certificate. Indeed, the iterative process for this certificate starts with $\mathbf{X} = (\bot, \dots, \bot)$, sets $X_g = 1$ in the first step, but can not proceed further. This is because the knowledge of X_g does not immediately uniquely identify any other value in $\mathbf{X}$, since, considering first preferences only, g is only mentioned in the first preference of c, and $Maj(e, f, g) = Maj(e, f, 1)$ is not uniquely determined without knowing X_e and X_f first.

Hence, if there is a consistent certificate using only the first two preferences of all agents, then it is a MINMAX certificate. One such certificate is $\mathbf{c} = (0, 1, 0, 0, 1, 1, 0)$. To see this, start again the iterative process with $\mathbf{X} = (\bot, \dots, \bot)$. We immediately obtain $X_g \leftarrow 1$ and $X_f \leftarrow 0$. Subsequently, $X_e \leftarrow X_f \wedge X_g = 0 \wedge 1 = 0$. Then, $X_c \leftarrow Maj(X_e, X_f, X_g) = Maj(0, 0, 1) = 0$, which could not have been determined before X_e since $Maj(X_e, 0, 1) = X_e$. Afterwards, $X_b \leftarrow X_c = 0$. Then $X_a \leftarrow X_b \vee X_c = 0 \vee 0 = 0$ and $X_d \leftarrow X_a = 0$. So, the iterative procedure has successfully determined the concrete votes $\mathbf{X} = (0, 0, 0, 0, 0, 0, 1)$, so $\mathbf{c}$ is a MINMAX certificate; i.e., $\max_{a \in N} c_a = 1$. Note that the order used to resolve the votes is $g \lhd f \lhd e \lhd c \lhd b \lhd a \lhd d$. Notationally, the concrete votes satisfy $\mathbf{X} \in \text{MINMAX}(\mathbf{B})$.

The previous certificate had $\sum_{a \in N} c_a = 3$. In contrast, a MINSUM certificate is given by $\mathbf{c} = (0, 0, 2, 0, 0, 0, 0)$, which satisfies $\sum_{a \in N} c_a = 2$. Performing the unravelling in this case would yield the direct votes $\mathbf{X} = (1, 0, 1, 1, 1, 1, 1) \in \text{MINSUM}(\mathbf{B})$. Note that $\mathbf{c}$ is not the only MINSUM certificate: $\mathbf{c}' = (0, 0, 0, 2, 0, 0, 0)$ would be a different one.

We end these preliminaries by returning to the expressivity of the language used to specify functions. For computational tractability, all functions that appear in the ballots should be polynomial-time computable. Moreover, ballot validation, as well as checking whether a given certificate is consistent (say after it has been reported to the voters) both require that checking extensional function equality can be achieved in polynomial time. These are all rather demanding properties, and they notably fail unless P = NP even for functions expressed as monotone formulas in disjunctive normal form (DNF), as reported by Colley et al. (2022). As a resolution, they propose restricting the class of allowable functions to minimised DNF formulas, for which the problems become tractable. We do not make this assumption, but instead note that for our results these conditions will be either trivially satisfied or not necessary (the latter holds for the hardness parts of dichotomy results).

3 Complexity of the Enhanced Model

In this section, we provide complete computational dichotomies for MINSUM and MINMAX for the binary setting assuming that all allowable functions are monotonic. It is easy to see why non-monotonic delegations may cause issues that are more fundamental than computational complexity. E.g., suppose the supporters of a substantial-sized party delegate to the negation of the vote of a high-influence individual who votes directly for 1. This individual is then motivated to vote directly for 0 knowing that this flips the vote of many agents from 0 to 1, adding an undesirable strategic element to the setup.

For $F \in \{\text{MINSUM}, \text{MINMAX}\}$ and a class of Boolean functions $\mathcal{F}$ including projections and constants (i.e., direct delegations and direct votes), we write $F_{\mathcal{F}}$ for F restricted to delegations to functions in $\mathcal{F}$. Notable examples of $\mathcal{F}$ include BOOL, the class of all Boolean functions, MON, the class of all monotonic Boolean functions, LIQUID, which only allows projections and constants (i.e., classic liquid democracy), OR_2, which additionally allows binary disjunctions (similarly AND_2), OR, which allows arbitrary-arity disjunctions (similarly AND) and finally $\text{ORAND}_2 = \text{OR}_2 \cup \text{AND}_2$. We assume $\mathcal{F}$ to be invariant to permuting agents in N; e.g., if agent a can delegate to $(b \vee c) \wedge d$, then a' can delegate to $(b' \vee c') \wedge d'$ for any pairwise distinct a', b', c', d'. This assumption is mild given that anonymity is seen as a desirable preference aggregation axiom.

For brevity, we adopt the following notational conventions. We say that $\text{MINSUM}_{\mathcal{F}}$ is NP-hard if the decision problem "Given x, decide if there is a consistent certificate with $\sum_{a \in N} c_a \leq x$" is NP-hard. We say that it is poly-time computable if there exists a polynomial-time algorithm outputting a $\text{MINSUM}_{\mathcal{F}}$ certificate. We say that $\text{MINSUM}_{\mathcal{F}}$ is hard to approximate within a constant factor if, unless P = NP, for no constant $C \geq 1$ does there exist a polynomial-time algorithm outputting a consistent certificate $\mathbf{c}$ with $\sum_{a \in N} c_a$ at most C times more than this sum for a MINSUM certificate. Moreover, we say that it is hard to approximate within an $n^{1-\epsilon}$ factor if for no $\epsilon > 0$ the above holds for $C = n^{1-\epsilon}$. We extend this terminology to $\text{MINMAX}_{\mathcal{F}}$.

Colley et al. (2022) show that $\text{MINSUM}_{\text{LIQUID}}$ and $\text{MINMAX}_{\text{LIQUID}}$ are polynomial-time computable while $\text{MINSUM}_{\text{AND}}$ and $\text{MINMAX}_{\text{BOOL}}$ are NP-hard. We strengthen these results by giving a full characterisation of when the problems admit polynomial-time algorithms.

Complexity of $\text{MINSUM}_{\mathcal{F}}$. We start by showing that if either $\text{OR}_2 \subseteq \mathcal{F}$ or $\text{AND}_2 \subseteq \mathcal{F}$, then $\text{MINSUM}_{\mathcal{F}}$ is NP-hard and hard to approximate within an $n^{1-\epsilon}$ factor, even for ballots of maximum size two.

Theorem 4. *$\text{MINSUM}_{\text{OR}_2}$ and $\text{MINSUM}_{\text{AND}_2}$ are hard to $n^{1-\epsilon}$-factor approximate even for maximum ballot size 2.*

We prove this result by reducing from the decision version of VERTEXCOVER. We construct voter gadgets for each vertex and edge. Edge gadgets can be resolved using the voters' first preferences if and only if the edge is covered. We then use gap amplification to show hardness of approximation. The $n^{1-\epsilon}$ bound is tight for ballots of size at most 2 since there exists a simple n-approximation: unless everyone can get their first (cost 0) choice, the optimal solution has cost at least 1, so giving everyone their second choice is an n-approximation.

Complexity of $\text{MINMAX}_{\mathcal{F}}$. Next, we show that if $\text{ORAND}_2 \subseteq \mathcal{F}$ then $\text{MINMAX}_{\mathcal{F}}$ is NP-hard even for ballots of maximum size three,[6] and hard to approximate within a constant factor. In contrast, we prove that $\text{MINMAX}_{\text{OR}}$ and $\text{MINMAX}_{\text{AND}}$ are polynomial-time computable, showing the need for both $\wedge$ and $\vee$ to obtain hardness. We begin with the easiness result.

Theorem 5. *$\text{MINMAX}_{\text{OR}}$ and $\text{MINMAX}_{\text{AND}}$ are polynomial-time computable (for arbitrary ballot sizes).*

We now turn to the hardness results. The proofs are similar in spirit to those for MINSUM. We reduce from a version of 3SAT.

Theorem 6. *$\text{MINMAX}_{\text{ORAND}_2}$ is NP-hard even for maximum ballot size 3, and hard to constant-factor approximate.*

Computational Dichotomies. So far, we have proven results concerning $\vee$ and $\wedge$. We now extend our results to full computational dichotomies establishing tractability/intractability for every class of monotonic functions $\mathcal{F} \subseteq \text{MON}$. We implicitly assume $\text{LIQUID} \subseteq \mathcal{F}$ since LIQUID alone entails tractability anyway.

Theorem 7. *Assume $\mathcal{F} \subseteq \text{MON}$. If $\mathcal{F} = \text{LIQUID}$, $\text{MINSUM}_{\mathcal{F}}$ is poly-time computable. Otherwise, $\text{MINSUM}_{\mathcal{F}}$ is NP-hard and hard to $n^{1-\epsilon}$-approximate even for ballots of size at most 2. Also, If $\mathcal{F} \subseteq \text{OR}$ or $\mathcal{F} \subseteq \text{AND}$, $\text{MINMAX}_{\mathcal{F}}$ is poly-time computable. Otherwise, $\text{MINMAX}_{\mathcal{F}}$ is NP-hard even for ballots of size at most 3, and hard to constant-factor approximate.*

As an immediate corollary, if agents are allowed to delegate to some odd majority then MINSUM and MINMAX are hard, resolving a question left open by Colley et al. (2022).

4 Complexity of the Classic Model

We now revisit the classic model of liquid democracy with ranked delegations, as described in Section 2.1. In particular, we study the MINSUM and MINMAX unravelling procedures, corresponding to minimum cost and minimum bottleneck arborescences in the delegation graph.

4.1 Efficient Computation of MINSUM and MINMAX Certificates

Our first results concern efficiently computing an arbitrary optimal certificate. Compared to previous works, our algorithm for MINMAX is asymptotically tight, while the one for MINSUM is tight up to a logarithmic factor. In this section only, we do not make the assumption that $D = \{0, 1\}$,

[6] Note the contrast with $\text{MINSUM}_{\mathcal{F}}$, where two sufficed for hardness. For maximum size two, finding a MINMAX certificate reduces to checking whether $\mathbf{c} = (0, \ldots, 0)$ is consistent, whose difficulty is given solely by the difficulty of testing function equivalence, which is easy for ORAND_2.

since our analysis applies to general domains with no additional complications. We work with a modified delegation graph: we remove the vertices that correspond to $D \subseteq V$, and redirect the edges pointing to them to point directly to the root r.[7] We then have $|V| = n+1$; we write $m = |E| = n + \sum_{a \in N} k_a$ for the total number of edges and ℓ for the maximum delegation ballot size. The algorithms proposed by Colley, Grandi, and Novaro (2022) run in time $\mathcal{O}(nm)$ for MINSUM and $\mathcal{O}(n^2\ell^2)$ for MINMAX.

Theorem 8. *A* MINSUM *certificate can be computed in time* $O(n \log n + m)$*, and a* MINMAX *certificate can be computed in time* $O(n + m)$.

The algorithm we propose for MINMAX can be described in purely graph-theoretic terms: given a directed graph with integer edge weights between 0 and W, it computes a minimum bottleneck arborescence with a given root in time $O(n + m + W)$. Despite the simplicity of the algorithm, we have not been able to find it in prior work.

4.2 Control for MINSUM and MINMAX

Thus far, we have concerned ourselves with the irresolute variants of MINSUM and MINMAX, where the electoral authority can choose any arborescence minimising the respective quantity and then apply an aggregation function. But what if they want to favour a specific alternative $d \in D = \{0, 1\}$? Without loss of generality, assume they want to favour alternative 1. Recall that for any arborescence T, the agents $a \in N$ that vote 1 in the concrete votes are the descendants of 1 in T. For both unravelling procedures, we will show that, surprisingly, if N_1 is the set of agents who vote for 1 in at least one optimal arborescence, then there is an optimal arborescence where all agents in N_1 vote for 1, and such an arborescence can be computed in polynomial time. This has immediate implications for the electoral authority attempting to exercise control: they can compute and select such an arborescence. The respective concrete votes subsume all possible concrete votes for 1 found in any optimal arborescence, so this leads to the best possible outcome for the authority, no matter which monotonic aggregation function is applied to the concrete votes. For MINMAX, the proof is not difficult.

Theorem 9. *Let $N_1 \subseteq N$ be the agents who vote 1 in some minimum bottleneck arborescence (i.e.,* MINMAX *certificate). Then, we can compute in $O(n+m)$ time a minimum bottleneck arborescence where all agents in N_1 vote 1.*

Proof. For convenience, we work on the transposed graph. Let w^* be the maximum edge weight used in a minimum bottleneck arborescence. This can be computed in $O(n+m)$ time using Theorem 8. Write $E_{\leq x} = \{e \in E \mid w(e) \leq x\}$ for the set of edges of weight at most x and $G_{\leq x} = (V, E_{\leq x})$ for the graph restricted to such edges. Then, any arborescence of $G_{\leq w^*}$ is a minimum bottleneck arborescence, and vice-versa. For readability, assume we remove all other edges and set $G = G_{\leq w^*}$. Consider the set S of vertices reachable from 1 in G. Note that in all arborescences vertices in $(V \setminus \{r\}) \setminus S$ will be descendants of 0, so, if we can find an arborescence where all vertices in S are descendants of 1, then the conclusion follows. Such an arborescence can indeed easily be constructed by running a depth-first search starting from the root and recursing along the edge to 1 before the edge to 0. This takes $O(n + m)$ time. Note that we implicitly get that $N_1 = S \setminus \{1\}$. □

[7] This gives slightly tighter and cleaner time bounds, whilst not impacting the certificate-arborescence correspondence.

The proof for MINSUM, while similar in spirit, requires insights from Fulkerson's primal-dual algorithm for computing minimum cost arborescences (Fulkerson 1974). Modern accounts of the algorithm are available in (Kamiyama 2014; Utke and Schmidt-Kraepelin 2023; Bernáth and Pap 2013). The algorithm outputs a collection $E' \subseteq E$ of edges of the graph, called the "tight" edges, as well as a *laminar family* $\mathcal{L}$ of subsets of $V \setminus \{r\}$. Laminarity means that for any $L_1, L_2 \in \mathcal{L}$ either $L_1 \subseteq L_2$, or $L_2 \subseteq L_1$ or $L_1 \cap L_2 = \varnothing$. Laminar families can be understood through their associated forest, where each set $L \in \mathcal{L}$ has a set of children sets $ch_{\mathcal{L}}(L)$ consisting of those $L' \in \mathcal{L}$ such that $L' \subsetneq L$ and there is no $L'' \in \mathcal{L}$ satisfying $L' \subsetneq L'' \subsetneq L$. The set of *roots* of the forest is denoted $rt_{\mathcal{L}}$ and consists of all maximal sets in $\mathcal{L}$. Fulkerson's algorithm is given in the appendix. For this section, it suffices to understand a few key properties of its output known from previous works. Most importantly, the relationship between minimum cost arborescences and $(E', \mathcal{L})$ is given by the following powerful lemma, for which given a set of vertices $S \subseteq V$, we write $\rho(S) = \{(u, v) \in E \mid u \in S \text{ and } v \notin S\}$.

Lemma 10. An arborescence T has minimum cost iff the following two conditions hold: (i) $T \subseteq E'$; (ii) for any $L \in \mathcal{L}$ it holds that $|\rho(L) \cap T| = 1$.

This gives a combinatorial characterisation of minimum-cost arborescences without mentioning edge weights, enabling us to adapt our approach for MINMAX to MINSUM.

Theorem 11. *Let $N_1 \subseteq N$ be the set of agents who vote 1 in some min-cost arborescence (i.e.,* MINSUM *certificate). Then we can compute in polynomial time a min-cost arborescence where all agents in N_1 vote 1. Moreover, $a \in N_1$ iff 1 is reachable from a along edges in E' output by Fulkerson's algorithm.*

On the positive side, Theorems 9 and 11 suggest biased tie-breaking rules for MINMAX and MINSUM, which we denote by MINMAX^1 and MINSUM^1, selecting the collection of concrete votes $(X_a)_{a \in N}$ with the highest possible number of ones. Our results show that these procedures are resolute and subsume all possible votes for 1 among MINMAX (MINSUM) certificates. One can also define MINMAX^0 and MINSUM^0 analogously. These biased procedures can find use in settings where the decision to be made is for changing a status quo, where one can select between biasing for or against the status quo prior to the vote. We note the following attractive corollary of our results, for which given two collections of concrete votes $\mathbf{X}, \mathbf{Y}$ we write $\mathbf{X} \leq \mathbf{Y}$ to mean that $X_a \leq Y_a$ for all $a \in N$:

Corollary 12. For any ballot list $\mathbf{B}$ and $\mathbf{X} \in \text{MINMAX}(\mathbf{B})$, $\mathbf{X}' \in \text{MINSUM}(\mathbf{B})$, we have $\text{MINMAX}^0(\mathbf{B}) \leq \mathbf{X} \leq \text{MINMAX}^1(\mathbf{B})$ and $\text{MINSUM}^0(\mathbf{B}) \leq \mathbf{X}' \leq \text{MINSUM}^1(\mathbf{B})$.

5 Axiomatic Results

In this section we study MINSUM and MINMAX from a normative standpoint. We introduce a natural axiom, which we call *cast monotonicity*, and establish that it is satisfied by MINSUM and a lexicographic refinement of MINMAX (but not MINMAX itself). This holds both for the irresolute procedures and for their biased resolute variants. Unless stated otherwise, in this section we consider the classic model.

Fix a set of agents N and $D = \{0, 1\}$. Let F be a (possibly irresolute) unravelling procedure mapping collections of ballots $\mathbf{B} = (B_a)_{a \in N}$ to collections of concrete votes $\mathbf{X} = (X_a)_{a \in N}$. Given an aggregation function $agg : \{0, 1\}^N \to \{0, 1\}$, the set of possible winning alternatives with respect to F on input $\mathbf{B}$ is $agg(F(\mathbf{B})) = \{agg(\mathbf{X}) \mid \mathbf{X} \in F(\mathbf{B})\}$.

Definition 13. An unravelling procedure F is *cast-monotonic* with respect to a monotonic aggregation function agg if for every ballot list $\mathbf{B}$, every agent $a \in N$, each alternative $d \in D$ and the ballot list $\mathbf{B}'$ obtained from $\mathbf{B}$ by changing B_a to a direct vote for d we have: (i) $d \in agg(F(\mathbf{B}))$ implies $d \in agg(F(\mathbf{B}'))$ and (ii) $1 - d \notin agg(F(\mathbf{B}))$ implies $1 - d \notin agg(F(\mathbf{B}'))$. We say that F is *(unconditionally) cast-monotonic* if it is cast-monotonic with respect to all monotonic aggregation functions agg.

Intuitively, cast monotonicity says that if an agent likes an alternative d, then she is best off voting for it directly: changing from an arbitrary ballot to a direct vote for d can neither remove d from the set of winners nor add $1 - d$ to the set of winners.

Given two lists of concrete votes $\mathbf{X}, \mathbf{X}'$, recall that $\mathbf{X} \leq \mathbf{X}'$ if $X_a \leq X'_a$ for all $a \in N$. We write $\leq_d$ to mean $\leq$ for $d = 1$ and $\geq$ for $d = 0$. Then, we can give the following characterisation of unconditional cast monotonicity.

Lemma 14. An unravelling procedure F is cast-monotonic iff for every ballot list $\mathbf{B}$, every agent $a \in N$, every alternative $d \in D$, and the ballot list $\mathbf{B}'$ obtained from $\mathbf{B}$ by changing B_a to a direct vote for d, the following conditions hold: (iii) for every $\mathbf{X} \in F(\mathbf{B})$ there is an $\mathbf{X}' \in F(\mathbf{B}')$ with $\mathbf{X} \leq_d \mathbf{X}'$; (iv) for every $\mathbf{X}' \in F(\mathbf{B}')$ there is an $\mathbf{X} \in F(\mathbf{B})$ with $\mathbf{X} \leq_d \mathbf{X}'$. Note that if F is resolute, (iii) is equivalent to (iv).

It is not hard to show that MINMAX and its biased variants are not cast-monotonic.

Theorem 15. MINMAX *fails cast monotonicity. The same holds for its resolute variants* MINMAX^p *with* $p \in \{0, 1\}$.

Next, we show that MINSUM and its variants are cast-monotonic. Our proof (see the appendix) relies on somewhat technical arguments regarding Fulkerson's algorithm.

Theorem 16. MINSUM, MINSUM^0, MINSUM^1 *satisfy cast monotonicity.*

The reader may wonder whether cast monotonicity results for MINSUM extend to the more general setting where agents can delegate to monotonic Boolean functions. Sadly, this is not the case, even if we only allow binary $\vee$ or $\wedge$.

Theorem 17. $\text{MINSUM}_{\mathcal{F}}$ *fails cast monotonicity as soon as either* $\text{OR}_2 \subseteq \mathcal{F}$ *or* $\text{AND}_2 \subseteq \mathcal{F}$. *The same holds for the resolute variants* $\text{MINSUM}^p_{\mathcal{F}}$ *for* $p \in \{0, 1\}$.

To end the section, we observe that the lexicographic refinement of MINMAX satisfies cast monotonicity. In this refinement, we seek a certificate that minimises the number of agents $a \in N$ with $c_a = n - 1$, breaking ties by the number of agents with $c_a = n - 2$, and so on. Formally:

Definition 18 (LEXIMIN)**.** Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be two certificates. We say that $\mathbf{c}_1 \leq_{lex} \mathbf{c}_2$ if, after sorting the entries in $\mathbf{c}_1$ and $\mathbf{c}_2$ in non-increasing order, $\mathbf{c}_1$ is lexicographically no larger than $\mathbf{c}_2$. The unravelling procedure LEXIMIN returns a minimum consistent certificate with respect to $\leq_{lex}$.

LEXIMIN can be viewed as a variant of MINSUM where the edge weights $0, 1, 2, \ldots, n - 1$ in the delegation graph are replaced with weights $1, X, X^2, \ldots, X^{n-1}$ for a large enough X (e.g., $X = n + 2$ suffices). Indeed, our results for MINSUM do not require the edge weights to be drawn from $\{0, \ldots, n - 1\}$; rather, they hold whenever the weights form a strictly increasing sequence. Accordingly, the two biased versions of LEXIMIN are resolute, easy to compute, and satisfy cast monotonicity. For more expressive function classes, LEXIMIN inherits its intractability from that of MINSUM, as it coincides with it on ballots of maximum size 2.

6 Summary and Future Work

We have extended the work of Colley et al. (2022), by strengthening their complexity-theoretic results to a complete dichotomy for monotone Boolean functions. For the standard model, we provided near-optimal algorithms for the two unravelling procedures and efficient algorithms for computing an outcome where a given alternative wins, leading to attractive tie-breaking rules for the setup where one alternative is the status quo. We introduced the cast monotonicity axiom, which MINSUM and the lexicographic refinement of MINMAX satisfy, while traditional MINMAX does not; these results extend to the biased variants of the rules.

One interesting direction for future work is to extend the model by allowing agents to assign cardinal values to their preferences, instead of just ordering them. The cardinal voting model can be seen as a meaningful generalisation of the preferential voting model we have explored. Moreover, our results show that computing optimal unravellings is NP-hard for most reasonable function classes, but there is a case relevant to the real world that is left uncovered. Namely, one implementation requires agents to choose before the election if they want to be delegates or not. Delegates have to disclose their votes while non-delegates can keep their vote secret. Suppose we allow non-delegates to vote for a single Boolean function of delegates, but delegates can only delegate directly to other delegates. Then, hardness of unravelling no longer arises as cycles are contained in the pure fragment of the model, increasing expressive power while maintaining tractability. An interesting question in this setting is under what conditions an agent prefers to join the delegates as opposed to remaining a voter.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive feedback and useful suggestions contributing to improving this paper. This work was supported by the AI Programme of The Alan Turing Institute. A preliminary version of this work has been presented at the Games, Agents, and Incentives Workshop of AAMAS'23, based on the first author's Master's thesis, which would not have been possible without the third author's supervision. We thank the workshop attendees for their interesting questions and remarks. The first author would additionally like to thank his good friends Victoria Walker, Dimitrios Iatrakis and Radostin Chonev for their assistance in this work.

References

Alouf-Heffetz, S.; Armstrong, B.; Larson, K.; and Talmon, N. 2022. How should we vote? A comparison of voting systems within social networks. In *31st International Joint Conference on Artificial Intelligence*, 31–38.

Becker, R.; D'Angelo, G.; Delfaraz, E.; and Gilbert, H. 2021. Unveiling the truth in liquid democracy with misinformed voters. In *7th International Conference on Algorithmic Decision Theory*, 132–146.

Behrens, J. 2017. The origins of liquid democracy. *The Liquid Democracy Journal*, (5): 7–17.

Bernáth, A.; and Pap, G. 2013. Blocking Optimal Arborescences. In Goemans, M.; and Correa, J., eds., *Integer Programming and Combinatorial Optimization*, 74–85. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-36694-9.

Bloembergen, D.; Grossi, D.; and Lackner, M. 2019. On rational delegations in liquid democracy. In *33rd AAAI Conference on Artificial Intelligence*, 1796–1803.

Brill, M.; Delemazure, T.; George, A.-M.; Lackner, M.; and Schmidt-Kraepelin, U. 2022. Liquid democracy with ranked delegations. In *36th AAAI Conference on Artificial Intelligence*, 4884–4891.

Caragiannis, I.; and Micha, E. 2019. A contribution to the critique of liquid democracy. In *28th International Joint Conference on Artificial Intelligence*, 116–122.

Colley, R.; Grandi, U.; and Novaro, A. 2022. Unravelling multi-agent ranked delegations. *Autonomous Agents and Multi-Agent Systems*, 36(1): 1–35.

Escoffier, B.; Gilbert, H.; and Pass-Lanneau, A. 2020. Iterative delegations in liquid democracy with restricted preferences. In *34th AAAI Conference on Artificial Intelligence*, 1926–1933.

Ford, B. A. 2002. Delegative democracy. Technical report.

Fulkerson, D. R. 1974. Packing Rooted Directed Cuts in a Weighted Directed Graph. *Math. Program.*, 6(1): 1–13.

Gabow, H.; Galil, Z.; Spencer, T.; and Tarjan, R. 1986. Efficient Algorithms for Finding Minimum Spanning Tree in Undirected and Directed Graphs. *Combinatorica*, 6: 109–122.

Gölz, P.; Kahng, A.; Mackenzie, S.; and Procaccia, A. D. 2021. The fluid mechanics of liquid democracy. *ACM Transactions on Economics and Computation*, 9(4): 1–39.

Kahng, A.; Mackenzie, S.; and Procaccia, A. 2021. Liquid democracy: An algorithmic perspective. *Journal of Artificial Intelligence Research*, 70: 1223–1252.

Kamiyama, N. 2014. Arborescence Problems in Directed Graphs: Theorems and Algorithms. *Interdisciplinary Information Sciences*, 20(1): 51–70.

Kavitha, T.; Király, T.; Matuschke, J.; Schlotter, I.; and Schmidt-Kraepelin, U. 2020. Popular Branchings and Their Dual Certificates. In Bienstock, D.; and Zambelli, G., eds., *Integer Programming and Combinatorial Optimization*, 223–237. Cham: Springer International Publishing. ISBN 978-3-030-45771-6.

Kotsialou, G.; and Riley, L. 2020. Incentivising participation in liquid democracy with breadth-first delegation. In *19th International Conference on Autonomous Agents and Multiagent Systems*, 638–644.

Utke, M.; and Schmidt-Kraepelin, U. 2023. Anonymous and Copy-Robust Delegations for Liquid Democracy. *arXiv 2307.01174*.

Zhang, Y.; and Grossi, D. 2021. Power in liquid democracy. In *35th AAAI Conference on Artificial Intelligence*, 5822–5830.

A Technical Appendix to "Unravelling Expressive Delegations: Complexity and Normative Analysis"

Abstract

This appendix contains the proofs omitted from the main body as well as additional supporting material. We moreover include a short section on axiomatic dichotomies and an observation concerning a hardness proof from previous work.

A.1 Preliminaries

Theorem 3. *Assuming delegation is only permitted to monotonic functions, a certificate* $\mathbf{c}$ *is consistent iff the system of simultaneous equations* $X_a = B_a(c_a)(\mathbf{X})$ *for all* $a \in N$ *has a unique solution.*

Proof. Consider the iterative procedure checking whether $\mathbf{c}$ is consistent. If it finishes with a complete collection of concrete votes $\mathbf{X}$, then this is a solution to the system, and it is the unique one given that all assignments were "forced." Otherwise, assume the procedure terminates with an incomplete collection $\mathbf{X}$ where entries in $N_\perp = \{a \in N \mid X_a = \perp\}$ are undetermined. Then, create two completions of $\mathbf{X}$, one where the entries for $N_\perp$ are assigned to 0 and one where they are assigned to 1, and call them $\mathbf{X^0}$ and $\mathbf{X^1}$ respectively. Note that $\mathbf{X^0} \leq \mathbf{X^1}$ holds, which is defined to mean that $X^0_a \leq X^1_a$ for all $a \in N$. Hence, because delegation is only allowed to monotonic functions, $B_a(c_a)(\mathbf{X}^0) \leq B_a(c_a)(\mathbf{X}^1)$ holds. We will show that $\mathbf{X^0}$ and $\mathbf{X^1}$ are two (distinct) solutions to the system. To do so, it is enough to check that for any $a \in N_\perp$ we have $B_a(c_a)(\mathbf{X}^0) = 0$ and $B_a(c_a)(\mathbf{X}^1) = 1$. This amounts to checking that $B_a(c_a)(\mathbf{X}^0) = B_a(c_a)(\mathbf{X}^1)$ would lead to a contradiction. This is indeed the case, as it would imply by monotonicity that for any completion $\mathbf{X}'$ of $\mathbf{X}$ the value $B_a(c_a)(\mathbf{X}')$ only depends on $\mathbf{X}$, contradicting that $\mathbf{X}$ was the output of the algorithm. □

A.2 Complexity of MinSum$_\mathcal{F}$

Theorem 4. MinSum$_{\text{Or}_2}$ *and* MinSum$_{\text{And}_2}$ *are hard to* $n^{1-\epsilon}$*-factor approximate even for maximum ballot size 2.*

Proof. Follows from Lemmas 19 to 21 below. □

Lemma 19. MinSum$_{\text{Or}_2}$ is NP-hard and hard to constant-factor approximate, even for maximum ballot size 2.

Proof. We reduce from the decision version of Vertex-Cover. For consistency with some of our later proofs, we work with an equivalent formulation of Vertex Cover based on 2SAT. Namely, the input is a Boolean formula $\Phi = \bigwedge_{i=1}^{k} C_i$ over variables $x_1, \ldots, x_n$, where each clause C_i is a disjunction of two positive literals, as well as a number K. We will construct an instance of MinSum$_{\text{Or}_2}$ for which a consistent certificate of cost at most K exists iff Φ has a satisfying assignment with at most K variables set to 1.

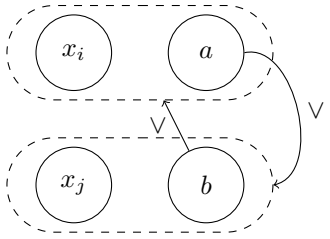


Figure 2: Gadget of clause $x_a \vee x_b$ for MinSum$_{\text{Or}_2}$.

For each variable x_i introduce a voter x_i with voting profile $B_{x_i} = 0 \succ 1$. Note that this is not a valid smart ballot, but can be simulated by introducing a constant voter zero with $B_{\text{zero}} = 0$ and using $B_{x_i} = \text{zero} \succ 1$ instead. Intuitively, variable x_i will evaluate to 1 precisely when voter x_i gets preference level 1.

For each clause $C = x_i \vee x_j$ in Φ we construct $K + 1$ identical gadgets with the intuitive property that they incur no additional cost if C is satisfied and a cost of at least $K+1$ otherwise. One gadget consists of two fresh voters a and b with smart profiles $B_a = x_j \vee b \succ 0$ and $B_b = x_i \vee a \succ 0$. This is illustrated in Figure 2.

We now prove the correctness of the construction. First, assume $\mathbf{y} = y_1, \ldots, y_n$ is a satisfying assignment of Φ with at most K variables set to 1. Then, we construct a certificate $\mathbf{c}$ of cost at most K, as follows. For voters x_i, set $c_{x_i} = y_i$, while for other voters v set $c_v = 0$. We now show that $\mathbf{c}$ is consistent. To do so, we need to show that the preferences of gadget voters can be unravelled into votes according to $\mathbf{c}$. Consider a clause $C = x_i \vee x_j$. Because $\mathbf{y}$ is a satisfying assignment, one of y_i or y_j is 1. By symmetry, it suffices to consider $y_i = 1$. Then, the vote of b can be computed as $x_i \vee a = y_i \vee a = 1 \vee a = 1$. Similarly, the vote of a can be computed as $x_j \vee b = x_j \vee 1 = 1$. Hence, $\mathbf{c}$ is consistent.

Conversely, assume Φ is not satisfiable and let $\mathbf{c}$ be a consistent certificate. We will show that the cost of c is at least $K + 1$. From certificate $\mathbf{c}$, construct an assignment $\mathbf{y} = y_1, \ldots, y_n$ by setting $y_i = c_{x_i}$. Because $\mathbf{y}$ does not satisfy Φ, consider a clause $C = x_i \vee x_j$ which is not satisfied by $\mathbf{y}$; i.e., $y_i = y_j = 0$. We now show that all $K + 1$ gadgets corresponding to C each incur a cost of at least one. Consider one such a gadget and assume for a contradiction that it incurs a cost of zero; i.e., its voters a and b both get their first preference, namely a gets $x_j \vee b$ and b gets $x_i \vee a$. However, since $y_i = y_j = 0$, these preferences resolve to $0 \vee b = b$ and $0 \vee a = a$, which is a preference cycle, contradicting the consistency of c.

We now apply gap amplification to our proof to also get hardness of approximation. Suppose that in the reduction above instead of $K + 1$ copies of each gadget we used $c(K + 1)$ copies for some $c \geq 1$. Then, by the same argument as above, if Φ is satisfiable with at most K variables set to 1, there is a consistent certificate of cost at most K, and otherwise all consistent certificates incur a cost of at least $c(K + 1)$. A polynomial-time c-approximation algorithm for MinSum$_{\text{Or}_2}$ could be used to produce a solution of cost a most $cK < c(K + 1)$ when Φ is satisfiable with at

most K variables set to 1, and hence can not exist unless P = NP. □

Lemma 20. $\text{MINSUM}_{\text{OR}_2}$ is hard to $n^{1-\epsilon}$-factor approximate even for maximum ballot size 2.

Proof. To get the stronger inapproximability result, we will set $c = n^k$ in our previous proof, with k to be chosen later. Care needs to be taken here because, to talk about a c-approximation, c needs to be expressed as a function of the $\text{MINSUM}_{\text{OR}_2}$ instance size.[8] To avoid notational clash, in this proof we use n to denote the number of variables in Φ and X to denote the number of voters in instances of $\text{MINSUM}_{\text{OR}_2}$. Assume for a contradiction that $\text{MINSUM}_{\text{OR}_2}$ admits a polynomial time $X^{1-\epsilon}$-approximation algorithm $\mathcal{A}$ for some $\epsilon > 0$. Assuming Φ consists of m clauses, the instance produced by our reduction consists of $X = n+1+2n^k(K+1)m \leq n+1+2n^k(n+1)n^3 = O(n^{k+4})$ voters, which for some $C > 0$ is at most Cn^{k+4}. Therefore, for our instance algorithm $\mathcal{A}$ approximates within a factor of $X^{1-\epsilon} \leq C^{1-\epsilon}n^{(k+4)(1-\epsilon)}$. To get a contradiction, we want that this is at most $c = n^k$; i.e., $C^{1-\epsilon}n^{(k+4)(1-\epsilon)} \leq n^k$, which is the same as $C^{1-\epsilon} \leq n^{k-(k+4)(1-\epsilon)} = n^{\epsilon(k+4)-4}$. By picking $k \geq 5\epsilon^{-1} - 4$, the latter will hold for n large enough, yielding a polynomial-time algorithm for deciding Vertex Cover, subject to n being large enough, which is not possible unless P = NP. □

Lemma 21. $\text{MINSUM}_{\text{AND}_2}$ is hard to $n^{1-\epsilon}$-factor approximate even for maximum ballot size 2.

Proof. We only outline the changes required compared to the proofs for $\vee$. Namely, variable voters now have $B_{x_i} = \underline{\text{one}} \succ 0$, where $\underline{\text{one}}$ is a constant voter with $B_{\underline{\text{one}}} = 1$. Moreover, the clause gadgets now consists of two voters a and b with smart ballots $B_a = x_j \wedge b \succ 0$ and $B_b = x_i \wedge a \succ 0$. The rest of the proof goes through as written, except for when it comes to calculating votes, where $\wedge$ replaces $\vee$, and caution needs to be observed when substituting in values for x_i because of the now "inverted" smart ballots used for the variable voters. For instance, where before we had calculated the vote of b in the first implication as $x_i \vee a = y_i \vee a = 1 \vee a = 1$, we now do it as $x_i \wedge a = B_b(c_{x_i}) \wedge a = B_b(y_i) \wedge a = (1 - y_i) \wedge a = 0 \wedge a = 0$. □

A.3 Complexity of $\text{MINMAX}_{\mathcal{F}}$

Theorem 5. $\text{MINMAX}_{\text{OR}}$ *and* $\text{MINMAX}_{\text{AND}}$ *are polynomial-time computable (for arbitrary ballot sizes).*

Proof. We show this for $\text{MINMAX}_{\text{OR}}$, the case of $\text{MINMAX}_{\text{AND}}$ is completely symmetric. We binary search for the smallest w such that a consistent certificate $\mathbf{c}$ with $\max_{a \in N} c_a \leq w$ exists. To check for a given w whether such a certificate exists and compute one if so, let us restrict the ballots to the first w preference levels, namely performing $k_a \leftarrow \min\{k_a, w\}$. Now, given the new ballots $\mathbf{B}$ we want to find any consistent certificate (which might not be possible if we deleted too many backup votes).

[8] So we do *not* get that an n^k-approximation does not exist.

The key observation is that each certificate $\mathbf{c}$ corresponds to a directed graph with vertex set N and nodes labeled with elements from the set $\{0, 1, \perp\}$, constructed as follows: label all nodes with $\perp$, except those $a \in N$ such that $B_a(c_a) = \tau_a$, which are labelled with $\tau_a \in \{0, 1\}$. For all other nodes, $B_a(c_a) = b_1 \vee b_2 \vee \ldots \vee b_k$ holds for some $k \geq 1$. Add to the graph all the edges in the set $E_a(c_a) = \{(a, b_i) \mid 1 \leq i \leq k\}$. Then, running the iterative algorithm computing the concrete votes $\mathbf{X}$ for $\mathbf{c}$ will lead to the following: nodes $a \in N$ that can reach a node labelled with 1 will satisfy $X_a = 1$; from the other nodes, those part of no cycle will satisfy $X_a = 0$ and the other ones $X_a = \perp$. Hence, a certificate is consistent if and only if nodes that can not reach a node labelled 1 in the corresponding graph form an acyclic induced subgraph.

Consequently, to find a consistent certificate, we should first try to maximize the set of nodes reaching a node labelled 1. To do this, we begin by assigning the backup vote to all agents still having it, if it happens to be a 1 (hence maximizing the set of nodes labelled 1). Then, consider another graph with vertex set N, the nodes already assigned labelled 1 and the rest labelled $\perp$, and with the edge set containing $E_a(i)$ for all a labelled $\perp$ and $0 \leq i \leq k_a$. Find the nodes that can reach a node labelled 1 in this graph and compute a directed spanning forest T of the subgraph induced by these nodes with sinks the nodes labelled 1 (this can be done by running a depth-first search on the transposed graph from all sinks and returning the tree edges). For each non-sink node a in this subgraph, let $(a, x) \in T$ be the corresponding parent edge in T. Subsequently, find some i for which $(a, x) \in E_a(i)$ and set $c_a = i$. We have now ensured that the maximum set of nodes can reach a node labelled 1 in the certificate. These nodes $N_1 \subseteq N$ are precisely the nodes $a \in N$ satisfying $c_a \neq \perp$[9] thus far. For the remaining nodes $N \setminus N_1$, we can only hope to make their induced subgraph in the certificate acyclic. To do this, assign the backup vote to agents in $N \setminus N_1$ still having it (which has to be a 0). Then, proceed iteratively: whenever an agent with $c_a = \perp$ exists such that for some $0 \leq i \leq k_a$ all edges $(a, b) \in E_a(i)$ satisfy $c_b \neq \perp$, assign $c_a = i$. If this process exhausts all of $N \setminus N_1$, then we have found a consistent certificate, otherwise none exist. Intuitively, this iterative process computes a topological sorting of the subgraph induced by $N \setminus N_1$ in the certificate, one agent at a time. □

Theorem 6. $\text{MINMAX}_{\text{OR}\wedge\text{AND}_2}$ *is NP-hard even for maximum ballot size 3, and hard to constant-factor approximate.*

Proof. Follows from Lemmas 22 and 23 below. □

Lemma 22. $\text{MINMAX}_{\text{OR}\wedge\text{AND}_2}$ is NP-hard even for maximum ballot size 3.

Proof. We reduce from the version of 3SAT where all clauses consist of either three positive or three negative literals, which is NP-hard by Schaefer's Dichotomy Theorem. The input is a Boolean formula $\Phi = \bigwedge_{i=1}^{k} C_i$ over variables

[9] We took the liberty in this proof to use $c_a = \perp$ to denote an unset value for c_a.

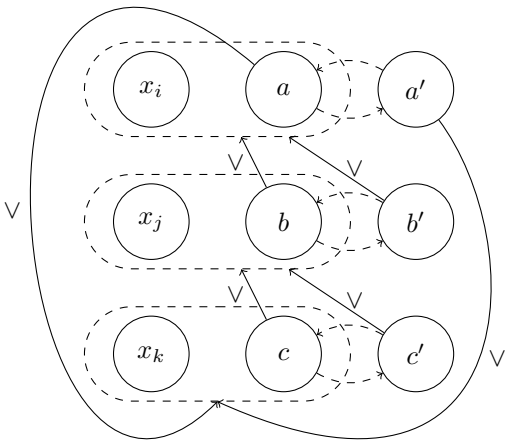


Figure 3: Gadget of clause $x_i \vee x_j \vee x_k$ for MINMAX$_{\text{ORAND}_2}$.

$x_1, \ldots, x_n$, where each clause C_i consists of either three positive or three negative literals. We will construct an instance of MINMAX$_{\text{ORAND}_2}$ for which a consistent certificate $\mathbf{c}$ with $\max \mathbf{c} \leq 1$ exists iff Φ is satisfiable.

For each variable x_i introduce a voter x_i with voting profile $B_{x_i} = 0 \succ 1$. Note that this is not a valid smart ballot but can be simulated by introducing a constant voter zero with $B_{\text{zero}} = 0$ and using $B_{x_i} = \text{zero} \succ 1$ instead. Intuitively, variable x_i will evaluate to 1 precisely when voter x_i gets preference level 1.

For each clause C in Φ we construct a gadget as follows: if $C = x_i \vee x_j \vee x_k$, introduce six fresh voters a, a', b, b', c, c' with preference profiles as follows, illustrated in Figure 3:

$$B_a = x_k \vee c \succ a' \succ 0 \qquad B_{a'} = x_c \vee c \succ a \succ 0$$
$$B_b = x_i \vee a \succ b' \succ 0 \qquad B_{b'} = x_i \vee a \succ b \succ 0$$
$$B_c = x_j \vee b \succ c' \succ 0 \qquad B_{c'} = x_j \vee b \succ c \succ 0$$

If, instead, $C = \overline{x_i} \vee \overline{x_j} \vee \overline{x_k}$, then the gadget is the same but with $\vee$ exchanged for $\wedge$. Intuitively, the clause gadget can be unravelled with only first and second preferences iff C is satisfied by the assignment of the variable voters. This is made more rigorous later.

We now prove the correctness of the construction. First, assume $\mathbf{y} = y_1, \ldots, y_n$ is a satisfying assignment of Φ. Then, we construct a consistent certificate $\mathbf{c}$ with $\max \mathbf{c} \leq 1$, as follows. For voters x_i, set $c_{x_i} = y_i$. Now, consider a clause $C = x_i \vee x_j \vee x_k$ (the negative literals case is symmetric). Because $\mathbf{y}$ is a satisfying assignment, one of y_i, y_j, y_k is 1. By symmetry, it suffices to consider $y_i = 1$. Then, the votes of b and b' can be computed using their first preferences as $x_i \vee a = y_i \vee a = 1 \vee a = 1$. Similarly, the votes of c and c' can be computed using their first preferences as $x_j \vee b = x_j \vee 1 = 1$. Finally, the votes of a and a' can be computed using their first preferences as $x_k \vee c = x_k \vee 1 = 1$. The described reasoning has produced a consistent certificate $\mathbf{c}$ with $\max \mathbf{c} \leq 1$.

Conversely, assume $\mathbf{c}$ is a consistent certificate with $\max \mathbf{c} \leq 1$. From certificate $\mathbf{c}$ construct an assignment $\mathbf{y} = y_1, \ldots, y_n$ by setting $y_i = c_{x_i}$. We will show that $\mathbf{y}$ satisfies Φ. Assume for a contradiction some clause $C = x_i \vee x_j \vee x_k$ (the negative literals case is symmetric) is not satisfied by $\mathbf{y}$; i.e., $y_i = y_j = y_k = 0$. Then, the preferences of the voters in the corresponding gadget resolve to the following by substituting x_i, x_j, x_k with 0:

$$B_a = c \succ a' \succ 0 \qquad B_{a'} = c \succ a \succ 0$$
$$B_b = a \succ b' \succ 0 \qquad B_{b'} = a \succ b \succ 0$$
$$B_c = b \succ c' \succ 0 \qquad B_{c'} = b \succ c \succ 0$$

Clearly, unless there exists $v \in \{a, a', b, b', c, c'\}$ such that $c_v = 2$, trying to unravel the preferences according to $\mathbf{c}$ will lead to a preference cycle, making $\mathbf{c}$ not consistent. However, this means that $\max \mathbf{c} \geq 2$, a contradiction. □

Lemma 23. MINMAX$_{\text{ORAND}_2}$ is hard to constant-factor approximate.

Proof. This can be proven by modifying the proof of Lemma 22. We will still introduce three voters a, b, c per clause. However, instead of voters a', b', c' we will introduce k voters of each kind: $(a'_\ell)_{\ell=1}^k, (b'_\ell)_{\ell=1}^k, (c'_\ell)_{\ell=1}^k$. We set the ballots of a, b, c as follows:

$$B_a = x_k \vee c \succ a'_1 \succ \ldots \succ a'_k \succ 0$$
$$B_b = x_i \vee a \succ b'_1 \succ \ldots \succ b'_k \succ 0$$
$$B_c = x_j \vee b \succ c'_1 \succ \ldots \succ c'_k \succ 0$$

We set the ballots of $a'_\ell, b'_\ell, c'_\ell$ as follows for $1 \leq \ell \leq k$:

$$B_{a'_\ell} = x_k \vee c \succ a'_1 \succ \ldots \succ a'_{\ell-1} \succ a \succ a'_{\ell+1} \succ \ldots \succ a'_k \succ 0$$
$$B_{b'_\ell} = x_i \vee a \succ b'_1 \succ \ldots \succ b'_{\ell-1} \succ b \succ b'_{\ell+1} \succ \ldots \succ b'_k \succ 0$$
$$B_{c'_\ell} = x_j \vee b \succ c'_1 \succ \ldots \succ c'_{\ell-1} \succ c \succ c'_{\ell+1} \succ \ldots \succ c'_k \succ 0$$

For this instance, similar reasoning to the proof of Theorem 6 shows that, if Φ is satisfiable, there is a consistent certificate with $\max \mathbf{c} \leq 1$, and otherwise all consistent certificates satisfy $\max \mathbf{c} = k + 1$. A polynomial-time α-approximation algorithm for MINMAX$_{\text{ORAND}_2}$ could be used to produce a consistent certificate $\mathbf{c}$ with $\max \mathbf{c} \leq \alpha$ (which is less than $k + 1$ if k is chosen large enough) whenever Φ is satisfiable, and hence can not exist unless P = NP. □

A.4 Computational Dichotomies

Given a Boolean function $f\colon \{0,1\}^k \to \{0,1\}$ and a partial assignment of values to its variables $\nu : \{0,1\}^k \to \{0,1,\bot\}$ we write $f(\nu)$ for the function obtained from f by substituting in the values where $\nu_i \neq \bot$. Then, we have the following lemma.

Lemma 24. Let $f\colon \{0,1\}^k \to \{0,1\}$ be a monotonic function. Denote its arguments by $x_1, \ldots, x_k$ and write $K = \{1, \ldots, k\}$. Then, unless $f = \bigvee_{i \in \mathcal{I}} x_i$ for some $\mathcal{I} \subseteq K$, there exist indices $p \neq q$ and a partial valuation ν assigning values to all arguments except x_p and x_q such that $f(\nu) = x_p \wedge x_q$. The same holds changing $(\bigvee, \wedge)$ to $(\bigwedge, \vee)$.

Proof. We prove the first part, as the second follows by applying the first to $f' = \overline{f(\overline{x_1}, \ldots, \overline{x_k})}$. Write f as a minimal monotone formula in DNF. Namely, let $C_1, \ldots, C_\ell \subseteq K$ be such that $f = \bigvee_{i=1}^{\ell} \bigwedge_{j \in C_i} x_j$. By minimality, for no $i \neq j$ does it hold that $C_i \subseteq C_j$. If $\ell = 0$ or $|C_i| = 1$ for all i then we are in the "unless" case. Otherwise, without loss of generality assume $|C_1| > 1$ and take $p \neq q$ such that $p, q \in C_1$. We construct ν by setting setting all variables for $K \setminus C_1$ to 0 and all variables for $C_1 \setminus \{p, q\}$ to 1. We now show $f(\nu) = x_p \wedge x_q$. Note that $\bigwedge_{j \in C_i} x_j$ evaluates to 0 under ν for all $i > 1$ because $C_i \not\subseteq C_1$, from which there exists $b \in C_i \setminus C_1 \subseteq K \setminus C_1$ implying $\nu_b = 0$. It remains to see that $\bigwedge_{j \in C_1} x_j = x_p \wedge x_q$ under ν, which is immediate given that ν sets the other variables for C_1 to 1. □

As an example, $\textsc{Maj}(x, y, 0) = x \wedge y$ and $\textsc{Maj}(x, y, 1) = x \vee y$. Note that the two "unless" cases above correspond to $f \in \textsc{Or}$ and $f \in \textsc{And}$. We are now ready to prove our full dichotomies.

Theorem 7. *Assume $\mathcal{F} \subseteq \textsc{Mon}$. If $\mathcal{F} = \textsc{Liquid}$, $\textsc{MinSum}_{\mathcal{F}}$ is poly-time computable. Otherwise, $\textsc{MinSum}_{\mathcal{F}}$ is NP-hard and hard to $n^{1-\epsilon}$-approximate even for ballots of size at most 2. Also, If $\mathcal{F} \subseteq \textsc{Or}$ or $\mathcal{F} \subseteq \textsc{And}$, $\textsc{MinMax}_{\mathcal{F}}$ is poly-time computable. Otherwise, $\textsc{MinMax}_{\mathcal{F}}$ is NP-hard even for ballots of size at most 3, and hard to constant-factor approximate.*

Proof. Follows from Lemmas 25 and 26 below. □

Lemma 25. Assume $\mathcal{F} \subseteq \textsc{Mon}$. If $\mathcal{F} = \textsc{Liquid}$, $\textsc{MinSum}_{\mathcal{F}}$ is poly-time computable. Otherwise, $\textsc{MinSum}_{\mathcal{F}}$ is NP-hard and hard to $n^{1-\epsilon}$-approximate even for ballots of size at most 2.

Proof. The easiness part follows from previous work (and also from Theorem 8). For the hardness part, assume $\textsc{Liquid} \subsetneq \mathcal{F}$ and consider some $f \in \mathcal{F} \setminus \textsc{Liquid}$. Notice that $\textsc{Liquid} = \textsc{Or} \cap \textsc{And}$, so $f \in (\mathcal{F} \setminus \textsc{Or}) \cup (\mathcal{F} \setminus \textsc{And})$. First, consider the case $f \in \mathcal{F} \setminus \textsc{Or}$. By Lemma 24, let ν be a partial valuation assigning values to all variables of f except say x_p and x_q such that $f(\nu) = x_p \wedge x_q$. Assuming we introduce constant voters $\underline{\text{zero}}$ and $\underline{\text{one}}$ with $B_{\underline{\text{zero}}} = 0$ and $B_{\underline{\text{one}}} = 1$ if not already present, we can now simulate the proof of Lemma 21 replacing 0/1 in ν by $\underline{\text{zero}}/\underline{\text{one}}$ and $\wedge$ by $f(\nu)$. The case $f \in \mathcal{F} \setminus \textsc{And}$ is analogous by using the other half of Lemma 24, and Lemmas 19 and 20 instead of Lemma 21. □

Lemma 26. Assume $\mathcal{F} \subseteq \textsc{Mon}$. If $\mathcal{F} \subseteq \textsc{Or}$ or $\mathcal{F} \subseteq \textsc{And}$, $\textsc{MinMax}_{\mathcal{F}}$ is poly-time computable. Otherwise, $\textsc{MinMax}_{\mathcal{F}}$ is NP-hard even for ballots of size at most 3, and hard to constant-factor approximate.

Proof. The easiness part is Theorem 5. For the hardness part, there exist $f \in \mathcal{F} \setminus \textsc{Or}$ and $f' \in \mathcal{F} \setminus \textsc{And}$. Applying Lemma 24 to f and f' we can simulate binary $\wedge$ and $\vee$ in the proof of Theorem 6 similarly to the proof of Lemma 25. □

A.5 Efficient Computation of MinSum and MinMax Certificates

Theorem 8. *A $\textsc{MinSum}$ certificate can be computed in time $O(n \log n + m)$, and a $\textsc{MinMax}$ certificate can be computed in time $O(n + m)$.*

Proof. Follows from Lemmas 27 and 28 below. □

Lemma 27. A $\textsc{MinSum}$ certificate can be computed in time $O(n \log n + m)$.

Proof. Direct consequence of applying the algorithm of Gabow et al. (1986) to compute a minimum cost arborescence. This is essentially a clever implementation of the Chu–Liu-Edmonds algorithm, the latter running in time $O(nm)$. □

We leave open whether the running time can be improved using the fact that edge weights are small non-negative integers (i.e., less than n). Note that for instances with average ballot size $\Omega(\log n)$, this is already asymptotically optimal.

Lemma 28. A $\textsc{MinMax}$ certificate can be computed in time $O(n + m)$.

Proof. For convenience, we work on the transposed graph. Write $E_x = \{e \in E \mid w(e) = x\}$ for the set of edges of weight x and $E_{\leq x} = \cup_{x'=0}^{x} E_{x'}$. An arborescence using edges with weights at most x exists iff all vertices can reach the root r in $G_{\leq x} = (V, E_{\leq x})$; i.e., $G_{\leq x}$ admits an arborescence. As edge weights are between 0 and $n-1$, this already gives an $O((n+m) \log n)$ algorithm: perform binary search for the smallest $0 \leq x < n$ such that $G_{\leq x}$ admits an arborescence and report some arborescence of $G_{\leq x}$.

To improve on this, compute the sets $(E_x)_{0 \leq x < n}$ and then proceed starting with $x = 0$ and increase x by one until all vertices in $G_{\leq x}$ can reach the root. To do this efficiently, maintain a set S of nodes that can reach the root at any point in time. Initially, $S = \{r\}$, and we stop whenever $S = V$. Moreover, maintain the current graph $G_{\leq x}$ using adjacency lists and add the new edges E_{x+1} to it one by one whenever x is incremented. When adding an edge (u, v), add it to u's adjacency list and check whether $u \in S$ and $v \notin S$. If so, launch a graph traversal (e.g., depth-first search) from v restricted to vertices in $V \setminus S$ and edges already in the adjacency lists, adding to S any newly discovered vertices. Because the graph traversals will enter each vertex exactly once and hence each edge $e \in E$ will be considered at most once in the traversals, the total time complexity is $O(n+m)$. □

A.6 Control for MinSum

Fulkerson's primal-dual algorithm for computing minimum cost arborescences (Fulkerson 1974; Kamiyama 2014; Utke and Schmidt-Kraepelin 2023; Bernáth and Pap 2013) is given in Algorithm 1.[10] To prove our result for control of $\textsc{MinSum}$, we will need a number of key properties of the output $(E', \mathcal{L})$ of the algorithm known from previous works. The most prominent one is Lemma 10, stated in the main text, which we repeat here for convenience.

[10] Technically, only the first phase of the algorithm. The second phase builds a min-cost arborescence similarly to Lemma 30 below.

Algorithm 1: Fulkerson's Algorithm

Input: Directed graph $G = (V, E)$ rooted at $r \in V$, weight function $w : E \to \mathbb{R}_{\geq 0}$.
Output: Set of tight edges $E' \subseteq E$ and laminar family $\mathcal{L}$ of subsets of $V \setminus \{r\}$.

$E', \mathcal{L} \leftarrow \varnothing$
while there exists a strongly connected component L of $G' = (V, E')$ such that $r \notin L$ and $\rho(L) \cap E' = \varnothing$ **do**
  Pick one such L arbitrarily.
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{L\}$
  $w_{min} \leftarrow \min_{e \in \rho(L)} w(e)$
  $w(e) \leftarrow w(e) - w_{min}$ for all $e \in \rho(L)$.
  $E' \leftarrow E' \cup \{e \in \rho(L) \mid w(e) = 0\}$
end while
return E' and $\mathcal{L}$.

Lemma 10. An arborescence T has minimum cost iff the following two conditions hold: (i) $T \subseteq E'$; (ii) for any $L \in \mathcal{L}$ it holds that $|\rho(L) \cap T| = 1$.

This gives a combinatorial characterization of minimum cost arborescences without mentioning edge weights. It can be shown that the output of the algorithm does not depend on the nondeterministic choice of L made on Algorithm 1, so it is meaningful and convenient to talk about "the output" of the algorithm, rather than "an output," but our proofs can all in principle be written without using this fact. We will also need the following property, which follows by directly examining the algorithm.

Lemma 29. For any $L \in \mathcal{L}$, the subgraph induced by L is strongly connected with respect to tight edges.

We begin with the following auxiliary result.

Lemma 30. For any $L \in \mathcal{L}$ and $v \in L$, there exists a v-arborescence of the induced subgraph $G[L]$ using only tight edges and satisfying condition (ii) for all $L' \in \mathcal{L}$ such that $L' \subsetneq L$. Moreover, it can be computed in polynomial time.

Proof. For clarity of exposition, remove all non-tight edges from G and restrict our attention to $G[L]$. Moreover, also for clarity, and without affecting the correctness of any of our previous claims, assume that $\mathcal{L}$ contains all singleton sets $\{v\}$ for $v \in V \setminus \{r\}$.

We construct the required arborescence recursively. First, enumerate $ch_{\mathcal{L}}(L) = \{L_1, \ldots, L_k\}$. Note that $L = \cup_{i=1}^{k} L_i$. Without loss of generality, assume $v \in L_1$. For $1 \leq i \leq k$ contract L_i to a single new node called x_i, keeping track of the correspondence between new edges and old edges.[11] This gives a new graph with vertex set $\{x_1, \ldots, x_k\}$. By Lemma 29, the subgraph induced by L is strongly connected, and this is not altered by contractions, so the new graph is also strongly connected. As a result, a graph traversal started from x_1; e.g., depth-first search; can be used to find an arborescence rooted at x_1 in the contracted graph. This arborescence corresponds to edges $(a_i, b_i)_{i=2}^{k}$ in the original graph $G[L]$, where $a_i \in L_i$. We make these edges part of the answer arborescence. To get the remaining edges, recurse inside each L_i to get an arborescence in $G[L_i]$ rooted at a_i with the required properties, and add it to the answer arborescence. □

[11] This can introduce multi-edges, we keep them and their costs.

We can now show our result for control of MINSUM by building on the proof idea for MINMAX.

Theorem 11. *Let $N_1 \subseteq N$ be the set of agents who vote 1 in some min-cost arborescence (i.e.,* MINSUM *certificate). Then we can compute in polynomial time a min-cost arborescence where all agents in N_1 vote 1. Moreover, $a \in N_1$ iff 1 is reachable from a along edges in E' output by Fulkerson's algorithm.*

Proof. Run Fulkerson's algorithm, and let $(E', \mathcal{L})$ be the output. The minimum-cost arborescences are precisely those satisfying conditions (i) and (ii) in Lemma 10. For readability, assume we remove all edges in $E \setminus E'$ from the graph. Then, we are interested in arborescences satisfying condition (ii) only. Consider the set S of vertices that can reach 1 in G. Note that, in all arborescences, vertices in $(V \setminus \{r\}) \setminus S$ will be descendants of 0, so, if we can find an arborescence satisfying (ii) where all vertices in S are descendants of 1, then the conclusion follows. We next show this last fact.

First, note that for any $L \in \mathcal{L}$, either $L \subseteq S$, or $L \subseteq (V \setminus \{r\}) \setminus S$. To see this, note that if $u, v \in L$ are such that $u \in S$ and $v \in (V \setminus \{r\}) \setminus S$, then, by Lemma 29, there would be a path from v to u, implying that $v \in S$, which can not be the case. To construct the arborescence, the approach is largely similar to the one taken in the proof of Lemma 30. Contract all sets in $rt_{\mathcal{L}}$ into single nodes and let x be the node corresponding to 1. Because any $L \in rt_{\mathcal{L}}$ is strongly connected and satisfies either $L \subseteq S$ or $L \subseteq (V \setminus \{r\}) \setminus S$, it follows that the nodes in the contracted graph that can reach x are precisely those corresponding to L's for which $L \subseteq S$. Then, construct an arborescence in the contracted graph by using (this time purposefully) depth-first search on the transposed graph starting from r and recursing to x before the node corresponding to 0. Then, as before, expand back the contracted nodes, marking the edges in the original graph corresponding to edges in the arborescence to be included in the answer arborescence. Finally, recurse inside each $L \in rt_{\mathcal{L}}$ with roots chosen as before using Lemma 30 to complete the arborescence. It can be checked that the output is a min-cost arborescence where agents in $S \setminus \{1\}$ vote for 1, from which implicitly we also get that $N_1 = S \setminus \{1\}$. □

A.7 Axiomatic Results

We will need the following auxiliary lemma:

Lemma 31. Consider an unravelling procedure F, two arbitrary collections of ballots $\mathbf{B}, \mathbf{B}'$ and some $d \in D$. Assume that for some $\mathbf{X} \in F(\mathbf{B})$ and all $\mathbf{X}' \in F(\mathbf{B}')$ it does not hold that $\mathbf{X} \leq_d \mathbf{X}'$. Then, there exists a monotonic aggregation function $agg : D^N \to D$ such that $d \in agg(F(\mathbf{B}))$ and $d \notin agg(F(\mathbf{B}'))$.

Proof. We consider the case $d = 1$, as the other case is fully analogous. Write $S = \{a \in N \mid X_a = 1\}$,

then we define the required monotonic aggregation function $agg = \bigwedge_{a \in S} a$. Note that for any input $\mathbf{Y}$, it holds that $agg(\mathbf{Y}) = 1$ iff $\mathbf{X} \leq \mathbf{Y}$. Hence, it follows that $agg(\mathbf{X}) = 1$ and $agg(\mathbf{X}') = 0$ for all $\mathbf{X}' \in F(\mathbf{B}')$. As a result, $1 \in agg(F(\mathbf{B}))$ and $1 \notin agg(F(\mathbf{B}'))$, as desired. □

Lemma 14. An unravelling procedure F is cast-monotonic iff for every ballot list $\mathbf{B}$, every agent $a \in N$, every alternative $d \in D$, and the ballot list $\mathbf{B}'$ obtained from $\mathbf{B}$ by changing B_a to a direct vote for d, the following conditions hold: (iii) for every $\mathbf{X} \in F(\mathbf{B})$ there is an $\mathbf{X}' \in F(\mathbf{B}')$ with $\mathbf{X} \leq_d \mathbf{X}'$; (iv) for every $\mathbf{X}' \in F(\mathbf{B}')$ there is an $\mathbf{X} \in F(\mathbf{B})$ with $\mathbf{X} \leq_d \mathbf{X}'$. Note that if F is resolute, (iii) is equivalent to (iv).

Proof. Assume $\mathbf{B}, a, d$ are such that (iii) and (iv) hold. We want to show that (i) and (ii) also hold for any monotonic aggregation function agg.

First, for (i), assume $d \in agg(F(\mathbf{B}))$. By definition, there exists $\mathbf{X} \in F(\mathbf{B})$ such that $agg(\mathbf{X}) = d$. By (iii), there exists $\mathbf{X}' \in F(\mathbf{B}')$ such that $\mathbf{X} \leq_d \mathbf{X}'$. By monotonicity, $agg(\mathbf{X}) \leq_d agg(\mathbf{X}')$, so $d \leq_d agg(\mathbf{X}')$. Hence, $agg(\mathbf{X}') = d$, implying that $d \in agg(F(\mathbf{B}'))$, as required.

Then, for (ii), write $d' = 1 - d$ and assume $d' \in agg(F(\mathbf{B}'))$, then we want to show that $d' \in agg(F(\mathbf{B}))$. Since $d' \in agg(F(\mathbf{B}'))$, let $X' \in F(\mathbf{B}')$ be such that $d' = agg(\mathbf{X}')$. By (iv), there exists $\mathbf{X} \in F(\mathbf{B})$ such that $\mathbf{X} \leq_d \mathbf{X}'$. By monotonicity, $agg(\mathbf{X}) \leq_d agg(\mathbf{X}')$, so $agg(\mathbf{X}) \leq_d d'$. Since $d' = 1 - d$ this means that $agg(\mathbf{X}) = d'$, so $d' \in agg(F(\mathbf{B}))$, as required.

Conversely, now assume $\mathbf{B}, a, d$ are such that one of (iii) or (iv) fails. We want to show that either (i) or (ii) fails for some monotonic aggregation function agg.

First, assume (iii) fails, then it follows that (i) also fails by directly applying Lemma 31.

Then, assume (iv) fails. Consider some $\mathbf{X}' \in F(\mathbf{B}')$ such that for all $\mathbf{X} \in F(\mathbf{B})$ it does not hold that $\mathbf{X} \leq_d \mathbf{X}'$. Write $d' = 1 - d$, and note that $\mathbf{X} \leq_d \mathbf{X}' \iff \mathbf{X}' \leq_{d'} \mathbf{X}$. Hence, applying Lemma 31 with d replaced by d' and with the roles of $(\mathbf{B}, \mathbf{X})$ and $(\mathbf{B}', \mathbf{X}')$ reversed, we get the existence of a monotone aggregation function agg such that $d' \in agg(F(\mathbf{B}'))$ and $d' \notin agg(F(\mathbf{B}))$, failing (ii), as required. □

Theorem 15. MINMAX *fails cast monotonicity. The same holds for its resolute variants* MINMAX^p *with* $p \in \{0, 1\}$.

Proof. Consider agents $N = \{a, a', \underline{\text{zero}}, u_1, \ldots, u_{n-3}\}$ casting ballots $\mathbf{B}$ as follows: $B_a = a' \succ 1, B_{a'} = a \succ 1, B_{\underline{\text{zero}}} = 0, B_{u_i} = \underline{\text{zero}} \succ 1$ for all $1 \leq i \leq n-3$. A certificate $\mathbf{c}$ is consistent iff at least one of $c_a = 1$ and $c_{a'} = 1$ holds, and, since all ballots have size at most two, all consistent certificates are MINMAX certificates. As a result, one can check that $\mathbf{X} \in \text{MINMAX}(\mathbf{B})$ iff $X_a = X_{a'} = 1$.

Now, consider ballots $\mathbf{B}'$ where the only difference is a switches their vote to $B'_a = 1$. Here, the unique MINMAX certificate has everyone delegating to their first preferences, leading to the corresponding concrete votes $\mathbf{X}' = (1, 1, 0, \ldots)$ from which $\text{MINMAX}(\mathbf{B}') = \{\mathbf{X}'\}$.

Since $\mathbf{X} = (1, 1, 1, \ldots) \in \text{MINMAX}(\mathbf{B})$ and $\mathbf{X} \notin \mathbf{X}'$, by Lemma 14, MINMAX fails cast monotonicity. For instance, this is the case for any odd $n \geq 5$ and the majority aggregation function agg. By the same reasoning, since $\text{MINMAX}^1(\mathbf{B}) = \mathbf{X}$ and $\text{MINMAX}^1(\mathbf{B}') = \mathbf{X}'$, MINMAX^1 also fails cast monotonicity. To show the same also holds for MINMAX^0 consider the same instance with the roles of 0 and 1 inverted. □

To show that MINSUM and its variants satisfy cast monotonicity, we require the following technical lemma about Fulkerson's algorithm concerning delegation graphs differing only in edges exiting a single node; we find it of independent interest.

Lemma 32. For each vertex $a \in V \setminus \{r, 0, 1\}$ in the delegation graph there exists a set $E'_a \subseteq E$ such that, no matter how the edges exiting a and their costs are altered, the set of tight edges E' output by Fulkerson's algorithm satisfies: (i) $E'_a \subseteq E'$; (ii) for any $(x, y) \in E' \setminus E'_a$ vertex x can reach a but not r along edges in E'_a; (iii) if there is a single edge exiting a, namely (a, d) for some $d \in D$, then $E' = E'_a \cup \{(a, d)\}$.

Proof. Consider running Fulkerson's algorithm with the following modification: change the loop condition to additionally require $a \notin L$ and never pick L with $a \in L$ in Algorithm 1. Let $(E'_a, \mathcal{L}_a)$ be the respective output. Note that, irrespective of the edges exiting node a and their weights, if $(E', \mathcal{L})$ is the output of the full algorithm, then $E'_a \subseteq E'$ and $\mathcal{L}_a \subseteq \mathcal{L}$, implying (i). To see this, recall that the output of the full algorithm does not depend on the nondeterministic choices made on Algorithm 1, so one can arrange for the full algorithm to be carried out by first executing the modified algorithm (which does not depend on edges exiting a) and the performing any supplementary steps.

Moreover, note that, with respect to edges in E'_a, every vertex can reach either r or a. To see this, assume that this was not the case for some vertex $x \in V \setminus \{r, a\}$. Then, all vertices reachable from x also fail the property. In particular, let C be a sink strongly connected component; i.e., $\rho(C) \cap E'_a = \varnothing$; reachable from x, then it follows that $r, a \notin C$. Hence, C satisfies the conditions to execute one more iteration of the while loop of the modified algorithm with $L = C$, contradicting the algorithm's termination. As a result, assuming the full algorithm is carried out by first executing the modified algorithm and then performing any necessary supplementary steps, the additional steps will always select L satisfying $a \in L$ and $L \subseteq V_a$, where we wrote $V_a \subseteq V \setminus \{r\}$ for the set of vertices that can reach a but not r along edges in E'_a. To see why this is the case, note that for all additional steps, the only strongly connected component excluding r and satisfying $\rho(L) \cap E' = \varnothing$ will be that of a as all vertices can reach either a or r along tight edges. Moreover, the process terminates the first time such an L is selected and an edge $(x, y) \in \rho(V_a)$ becomes tight, as then all vertices will be able to reach r. This implies (ii), namely that all edges (x, y) added during supplementary steps satisfy $x \in V_a$.

To see (iii) also holds, note that if (a, d) is the only edge exiting a, then, after executing the modified algorithm, the full algorithm will terminate in the next step by making this edge tight. This is because the strongly connected component of a after terminating the modified algorithm consists solely of a since no previous steps have made any edges exiting a tight. □

Theorem 16. MINSUM, MINSUM^0, MINSUM^1 *satisfy cast monotonicity.*

Proof. Follows from Lemmas 33 and 34 below. □

Lemma 33. MINSUM^0 and MINSUM^1 are cast-monotonic.

Proof. Since both rules are resolute, we show this by verifying condition (iii) of Lemma 14. Consider any collection of ballots $\mathbf{B}$, any agent $a \in N$ and, without loss of generality, $d = 1$. Write $\mathbf{B}'$ for $\mathbf{B}$ with B_a changed to a direct vote for 1. We want to show that $\text{MINSUM}^p(\mathbf{B}) \leq \text{MINSUM}^p(\mathbf{B}')$ for $p \in \{0, 1\}$. Let $E'_{\mathbf{B}}$ and $E'_{\mathbf{B}'}$ be the tight edges output by Fulkerson's algorithm executed on the delegation graphs of $\mathbf{B}$ and $\mathbf{B}'$ respectively. Note that the two delegation graphs only differ in the edges exiting $a \in N = V \setminus \{r, 0, 1\}$, so apply Lemma 32 to get a set $E'_a \subseteq E'_{\mathbf{B}} \cap E'_{\mathbf{B}'}$. Moreover, write $V_a \subseteq V \setminus \{r\}$ for the set of vertices that can reach a but not r along edges in E'_a. Then any $(x, y) \in (E'_{\mathbf{B}} \cup E'_{\mathbf{B}'}) \setminus E'_a$ satisfies $x \in V_a$. Finally, since $(a, 1)$ is the only edge exiting a for $\mathbf{B}'$, we have $E'_{\mathbf{B}'} = E'_a \cup \{(a, 1)\}$.

First, we show that $\text{MINSUM}^1(\mathbf{B}) \leq \text{MINSUM}^1(\mathbf{B}')$. By the last part of Theorem 11, this amounts to showing that for any $x \in N$, if 1 is reachable from x along edges in $E'_{\mathbf{B}}$, then it is also reachable from x along edges in $E'_{\mathbf{B}'} = E'_a \cup \{(a, 1)\}$. Assume a path P from x to 1 along edges in $E'_{\mathbf{B}}$ exists. If P only uses edges in E'_a, then the conclusion is immediate. Otherwise, P uses some edge in $E'_{\mathbf{B}} \setminus E'_a$. Let $(i, j) \in E'_{\mathbf{B}} \setminus E'_a$ be the first such edge encountered along P if traversed from x to 1. This implies that $i \in V_a$, so there is a path along edges in E'_a from i to a. Combining the fragment of P before edge (i, j), the path from i to a along edges in E'_a and the edge $(a, 1)$, we get a path from x to 1 using only edges in $E'_{\mathbf{B}'} = E'_a \cup \{(a, 1)\}$, as desired.

Next, we show that $\text{MINSUM}^0(\mathbf{B}) \leq \text{MINSUM}^0(\mathbf{B}')$. By the last part of Theorem 11 with the roles of 0 and 1 inverted, this amounts to showing that for any $x \in N$, if 0 is reachable from x along edges in $E'_{\mathbf{B}'} = E'_a \cup \{(a, 1)\}$, then it is also reachable from x along edges in $E'_{\mathbf{B}}$. Assume a path P from x to 0 along edges in $E'_{\mathbf{B}'} = E'_a \cup \{(a, 1)\}$ exists. By construction of the delegation graph, P can not use the edge $(a, 1)$, so it really is a path along edges in E'_a. Since $E'_a \subseteq E'_{\mathbf{B}}$, we get the conclusion. □

Lemma 34. MINSUM satisfies cast monotonicity.

Proof. In short, this follows by combining the results for the resolute variants in Lemma 33 with Corollary 12.

We show this by verifying the conditions (iii) and (iv) of Lemma 14. Consider any collection of ballots $\mathbf{B}$, any agent $a \in N$ and, without loss of generality, $d = 1$. Write $\mathbf{B}'$ for $\mathbf{B}$ with B_a changed to a direct vote for 1. We want to show that: (iii) for any $\mathbf{X} \in \text{MINSUM}(\mathbf{B})$ there is $\mathbf{X}' \in \text{MINSUM}(\mathbf{B}')$ such that $\mathbf{X} \leq \mathbf{X}'$; (iv) for any $\mathbf{X}' \in \text{MINSUM}(\mathbf{B}')$ there is $\mathbf{X} \in \text{MINSUM}(\mathbf{B})$ such that $\mathbf{X} \leq \mathbf{X}'$.

For (iii), by Corollary 12, any $\mathbf{X} \in \text{MINSUM}(\mathbf{B})$ satisfies $\mathbf{X} \leq \text{MINSUM}^1(\mathbf{B})$. By Lemma 33 and Lemma 14 for resolute rules, $\text{MINSUM}^1(\mathbf{B}) \leq \text{MINSUM}^1(\mathbf{B}')$. By transitivity, $\mathbf{X} \leq \text{MINSUM}^1(\mathbf{B}')$, as required.

For (iv), by Corollary 12, any $\mathbf{X}' \in \text{MINSUM}(\mathbf{B}')$ satisfies $\text{MINSUM}^0(\mathbf{B}') \leq \mathbf{X}'$. By Lemma 33 and Lemma 14 for resolute rules, $\text{MINSUM}^0(\mathbf{B}) \leq \text{MINSUM}^0(\mathbf{B}')$. By transitivity, $\text{MINSUM}^0(\mathbf{B}) \leq \mathbf{X}'$, as required. □

Theorem 17. MINSUM$_{\mathcal{F}}$ *fails cast monotonicity as soon as either* OR$_2 \subseteq \mathcal{F}$ *or* AND$_2 \subseteq \mathcal{F}$. *The same holds for the resolute variants* MINSUM$^p_{\mathcal{F}}$ *for* $p \in \{0, 1\}$.

Proof. We show this for OR$_2$, as the case of AND$_2$ is analogous. Consider agents $N = \{a, b, c, d, e, f, \underline{\text{zero}}\}$ with ballots $B_a = c \succ d \succ 1, B_b = \underline{\text{zero}} \succ 1, B_c = a \vee b \succ d \succ 1, B_d = a \vee b \succ c \succ 1, B_e = b \succ 1, B_f = b \succ 1$ and $B_{\underline{\text{zero}}} = 0$. First, note that the certificate $\mathbf{c} = (0, 1, 0, 0, 0, 0, 0)$ of sum 1 is consistent, leading to concrete votes $\mathbf{X} = (1, 1, 1, 1, 1, 1, 0)$. Moreover, one can check that any consistent certificate $\mathbf{c}'$ satisfying $c'_b = 0$ requires $c'_a + c'_c + c'_d \geq 2$. Hence, $\mathbf{c}$ is the unique MINSUM certificate with corresponding concrete votes $\mathbf{X}$, from which $\text{MINSUM}_{\mathcal{F}}(\mathbf{B}) = \{\mathbf{X}\}$.

Now, consider ballots $\mathbf{B}'$ where the only difference is a switching their vote to $B'_a = 1$. Here, the unique MINSUM certificate has everyone delegating to their first preferences, leading to the corresponding concrete votes $\mathbf{X}' = (1, 0, 1, 1, 0, 0, 0)$, from which $\text{MINSUM}_{\mathcal{F}}(\mathbf{B}') = \{\mathbf{X}'\}$.

Since $\mathbf{X} \not\leq \mathbf{X}'$ this fails cast monotonicity by Lemma 14. For instance, this is the case for the majority aggregation function agg. The results concerning MINSUM$^p_{\mathcal{F}}$ for $p \in \{0, 1\}$ follow since MINSUM$_{\mathcal{F}}$ is resolute for $\mathbf{B}, \mathbf{B}'$ □

A.8 Axiomatic Dichotomies

One can combine Theorems 16 and 17 with reasoning similar to that in the proof of Lemma 25 to get that $\mathcal{F} =$ LIQUID is the only class of monotonic functions for which MINSUM$_{\mathcal{F}}$ and its biased variants satisfy cast monotonicity.[12] This gives the following interesting axiomatic dichotomy.

Corollary 35. Assume $\mathcal{F} \subseteq$ MON. If $\mathcal{F} =$ LIQUID,[13] then MINSUM$_{\mathcal{F}}$, MINSUM$^0_{\mathcal{F}}$, MINSUM$^1_{\mathcal{F}}$ satisfy cast monotonicity. Otherwise, none of them satisfies cast monotonicity.

The same dichotomy can also be proven for LEXIMIN by noting that the instance constructed in the proof of Theorem 17 still applies.

[12] Negations do not help either: the beginning of Section 3 gives an example inline.

[13] Recall that we implicitly assume LIQUID $\subseteq \mathcal{F}$.

A.9 Note on a Previous Hardness Proof

It is worth noting that, upon closer inspection, the proof of Colley et al. (2022) that $\text{MinSum}_{\text{AND}}$ is NP-hard reduces from Feedback Arc Set. If a variant of Feedback Arc Set where each node has out-degree at most 2 were to be used, then NP-hardness for $\text{MinSum}_{\text{AND}_2}$ would follow, but not its hardness of approximation. In the present paper, we get to this result through a different reduction that also enables proving the hardness of approximation. A proof that this variant of Feedback Arc Set is NP-hard is relatively difficult to identify in the literature: we were only able to identify one in unrefereed sources,[14] although it is mentioned without proof in refereed works.

[14] https://cs.stackexchange.com/questions/117291/is-finding-the-minimum-feedback-arc-set-on-graph-with-two-outgoing-arcs-for-each