

Unravelling multi-agent ranked delegations *

Rachael Colley,¹ Umberto Grandi,¹ Arianna Novaro²
 {rachael.colley, umberto.grandi}@irit.fr, arianna.novaro@univ-paris1.fr

¹ Institut de Recherche en Informatique de Toulouse (IRIT), University of Toulouse

² Centre d'Economie de la Sorbonne (CES), University of Paris 1 Panthéon-Sorbonne

Abstract

We introduce a voting model with multi-agent ranked delegations. This model generalises liquid democracy in two aspects: first, an agent's delegation can use the votes of multiple other agents to determine their own—for instance, an agent's vote may correspond to the majority outcome of the votes of a trusted group of agents; second, agents can submit a ranking over multiple delegations, so that a backup delegation can be used when their preferred delegations are involved in cycles. The main focus of this paper is the study of unravelling procedures that transform the delegation ballots received from the agents into a profile of direct votes, from which a winning alternative can then be determined by using a standard voting rule. We propose and study six such unravelling procedures, two based on optimisation and four using a greedy approach. We study both algorithmic and axiomatic properties, as well as related computational complexity problems of our unravelling procedures for different restrictions on the types of ballots that the agents can submit.

*This paper revises and extends our previous work presented at IJCAI-2020 [19], which was developed from ideas discussed at the Dagstuhl Seminar 19381 on Application-Oriented Computational Social Choice in September 2019. We are grateful for the feedback received by the anonymous reviewers of IJCAI-2020 and JAAMAS, as well as the audience of MPREF-2020 and the COMSOC video seminar. Some of the work in this paper was performed while the third author was affiliated with the Institute for Logic, Language and Computation (ILLC) at the University of Amsterdam.

1 Introduction

In delegative voting, an agent's vote (or their corresponding voting power) can be passed to another voter or candidate. Dodgson [24] was the first to mention a delegative voting process, in the context of multi-winner elections, where a candidate could strategically delegate their excess votes to another candidate of their choosing.

In general, models of delegative democracy bridge the gap between direct and representative democracy, where in the former every member of a community has to vote on every issue that arises, whereas the latter allows elected representatives to decide on behalf of a community. On the one hand, direct democracy is arguably time-consuming and infeasible for large-scale voting, since voters must be informed on every issue to be able to vote on it. On the other hand, representative democracy tends to leave voters under-represented (although it exists in many forms). Delegative democracy can thus balance these problems, since agents can engage either *actively*, by voting directly, or *passively*, by choosing a representative for any issue [28].

Since voters can choose to actively participate into the decision-making, models of delegative democracy can also be seen as examples of *interactive democracy*, i.e., those voting systems that turn collective decisions into more engaging and responsive processes. In particular, Brill [10] argues that the progression of interactive democracy can be done in conjunction with the advancements of technology: for instance, in *e-democracy* the Internet is used to strengthen real-world and online democracies [47]. There are thus arguments in support of delegative democracy (and of upholding it via computerised methods), yet it is unclear how this should be implemented.

Proxy voting and *liquid democracy* are two instances of delegative democracy. Proxy voting allows voters to choose their own representative who votes on their behalf. Depending on the model, the representatives can be predetermined [2, 46]; any voter can be a representative and any agent can vote directly on any issue [32, 44, 49, 50]; or proxy voting is limited only to certain elections [40]. Liquid democracy allows agents to either vote directly on an issue or to delegate their vote to another trusted agent. Unlike in proxy voting, delegations are transitive: i.e., if you have delegated your vote to another agent, they are free to either vote directly or to delegate their own vote (as well as all other delegations they have received) to another agent [6]. However, the transitivity of delegations can lead to an agent's vote being used in a way that the agent would not support; for example, if a vote has been passed through many delegations, the original agent may not agree with

the final agent who votes on their behalf. Another issue arises in determining the outcome when some agents are stuck in a delegation cycle, i.e., when a delegating agent transitively receives their own delegation, as well as all of those in the cycle.

In this paper we tackle these problems by introducing a model of delegative democracy where voters can give multi-agent ranked delegations, thus generalising liquid democracy in two aspects. Firstly, we allow agents to submit a ranking over multiple possible delegations they would support, in order to ensure that their vote can be determined (should a delegation cycle arise). Secondly, voters can use the votes of multiple delegates in order to determine their own vote: for instance, they may state that their vote should coincide with the majority decision of a group of trusted delegates. Agents are thus able (but not required) to express complex delegations.

A natural question which arises when allowing for ranked delegations is when and how should the ranked delegations be used to break cycles. In this paper we define and analyse six of what we call *unravelling procedures*, which are used to find outcomes from delegations that could contain cycles.

1.1 Our contribution

In Section 2 we introduce our model of multi-agent ranked delegations, which we call *smart voting*, building upon the model introduced in previous work [19]. Our *smart ballots* give more expressivity to the agents than those from previous models of delegative democracy. Starting from these complex ballots, an *unravelling procedure* returns a standard voting profile from which a collective decision is taken. We introduce six unravelling procedures: MINSUM minimises the preference levels globally used for the agents; MINMAX minimises the maximum preference level used in the unravelling; the remaining four procedures use a greedy approach to guarantee tractability. We then introduce two restricted languages for smart ballots on binary issues: BOOL is our general language, where delegations are arbitrary Boolean functions expressed in complete DNF, while LIQUID allows for ranked single-agent delegation.

In Section 3 we prove our main results: the decision problems needed to compute MINSUM and MINMAX on the BOOL language are NP-complete, though they are polynomial for LIQUID ballots. Moreover, we prove that our four greedy unravelling procedures always terminate on valid smart ballots in polynomial time.

In Section 4 we compare our unravelling procedures, showing that all six can give outcomes which differ from each other and from the procedures by Kotsialou

and Riley [38]. However, the four greedy procedures and MINSUM coincide on standard liquid democracy ballots. We also study the axioms of cast-participation and guru-participation, as well as a notion of Pareto dominance. We conclude in Section 5.

1.2 Related work

In this section we present the related literature on delegative democracy, starting from the previous work that inspired our twofold generalisation of liquid democracy.

Multi-agent delegations. Our first generalisation is to allow voters to express delegations involving many other agents. In the same spirit, the model by Degraeve [22] allows for an agent’s delegation to be split among possibly many delegates: for example, a voter delegating to three agents could give half of their vote to one agent and a quarter to the other two delegates. Abramowitz and Mattei [1] give a similar model, but for proxy voting, where agents assign weights to a fixed set of representatives for each issue: agents can thus choose to spread their vote over many representatives.

In pairwise liquid democracy on ordinal elections [11], ballots are rankings of candidates and they can be completed by delegating a decision on distinct pairs of candidates to different delegates. Gözl et al. [29] address the problem of a small number of agents holding a lot of power by studying the fluid mechanics of liquid democracy, trying to balance influence like liquid in a vessel.

Ranked delegations. The second generalisation is to allow for ranked delegations, to avoid cycles (as in our case) or to avoid the delegation to an abstaining agent. Ford [28] introduces the notion of *delegation chains*, which we call *ranked delegations*, where an agent submits an ordering of many delegates, from the most trusted to one who still represents the agent, but less than the delegates coming before.

The model proposed by Kotsialou and Riley [38] includes ranked delegations that are used to avoid delegating to an abstaining agent and delegation cycles. They propose two procedures, breadth-first and depth-first, which find an outcome in a similar manner as we suggest; yet they do not allow delegations to abstaining agents. A similar approach has been proposed by Brill et al. [12] in ongoing work.

Kahng, Mackenzie, and Procaccia [34] suggest an alternative method of removing delegation cycles, by assigning a competency level to agents and forbidding them to delegate to someone with a lower competency level than themselves. Boldi et al. [8] propose *viscous democracy*, a model of liquid democracy that can be extended to include ranked delegations, where they make use of a dampening factor to ensure short chains of delegations. Behrens and Swierczek [3] propose a model of ranked liquid democracy as well as seven desirable properties for a liquid democracy system, showing that no system can satisfy all of them.

As an alternative to extending liquid democracy with ranked delegations, Escoffier, Gilbert, and Pass-Lanneau [27] introduce iterative delegations, to circumvent situations where outcomes cannot be determined. For example, an agent may want to change their vote if they are in a delegation cycle or if they do not approve of the *guru* (i.e., the final receiver of the delegation) that votes on their behalf. However, stable states may not exist: Escoffier, Gilbert, and Pass-Lanneau [26] study how the profile of preferences impacts the existence of a stable state, showing that some structures on profiles (such as single-peaked preferences) can ensure an equilibrium.

Other models of delegative democracy. Inspired by Kahng, Mackenzie, and Procaccia [34], Caragiannis and Micha [14] study the interplay of liquid democracy and truth-tracking. Assuming that voting is about finding some ground-truth, a delegation to another agent implies that you believe they will be better than you at finding the truth. Finding optimal delegations (obtaining the highest probability of finding the ground-truth) is however not a tractable problem and it is also hard to approximate.

Analogously, Cohensius and Meir [17] use proxy voting to approximate an underlying ground-truth, when voter participation would otherwise be low, and show via empirical data that the outcomes are more accurate than with full voter participation.

Christoff and Grossi [15] give a model of liquid democracy on multiple binary issues connected by constraints that reflect consistent sets of opinions. In this extension of liquid democracy an agent’s vote acquired via delegation must be consistent with the constraint and their votes on other issues. They also include an embedding of transitive delegative voting on binary issues into binary aggregation with abstentions. Bloembergen, Grossi, and Lackner [5] give a game-theoretic analysis of delegation games via pure-Nash equilibria, where the agent’s utility is the accuracy of their vote. They complement this study with agent-based simu-

lations that verify their theoretical best-response dynamic. Cohensius et al. [18] consider a game-theoretic model, where only a small number of agents are able or motivated to vote. They compare voting with and without proxies, studying which method approximates the optimal outcome. Their results support proxy voting when agents are placed on a line; however, in the general setting, some agents may gain too much power.

Zhang and Grossi [52] also study delegation games and provide a generalization of the Banzhaf index in liquid democracy to investigate the power of the agents who vote directly on the issues. Meir et al. [43] study power in proxy voting, trying to limit the power of sybil-voters in their system. They find bounds on the highest number of sybil-voters a system could have while still upholding two forms of sybil-resistance.

Social choice on social networks. Delegative democracy can be seen as a form of social choice on social networks [30], since delegations create a social network where an edge represents the trust one agent has for another. This is also seen in models of opinion diffusion: in both cases, trusted agents can affect either the opinion or the vote of another agent. Threshold models of opinion diffusion [31] are closely related to multi-agent delegations using quota rules; one such model is that of Brederbeck and Elkind [9], where opinions change to be in agreement with the majority of the agent’s neighbours. The pairwise diffusion model of Brill et al. [13] is closely related to the one of pairwise liquid democracy [11]: agents give an ordering over all alternatives for one issue, and the structure of both the network and the agent’s preferences are central to determine the termination of the diffusion process.

Advancements in voting technology. A challenge of delegative democracy is the technology required to implement it. Miller [44] in 1969 proposed that voters would need “*a special metal key, a coded combination, or even a thumbprint*” to ensure a safe voting process. Since then, the advancement of technology has surpassed these notions, yet the question remains of how to safeguard voting when relying on technology.

A suggestion is to use blockchain technology, such as smart contracts. Dhillon et al. [23] give a detailed plan of the infrastructure required for a decentralised on-line voting platform, using distributed ledgers—showing the strengths and weaknesses of such a system. Zhang and Zhou [51] give a model of *statement voting* that uses the Universal Composability framework to circumvent the issues of im-

plementing *e*-democracies. Their model allows for approval voting, STV, and liquid democracy.

Along with theoretical advancements, voting platforms have also become more commonplace—including those designed for liquid democracy. Mancini [42] advocates for political systems to be brought into line with the technology available, and that liquid democracy could “*rewire from the inside how politics works*”. Liquid democracy has also been promoted by political parties, such as the *Partido de Internet* in Mexico, the *Net Party* in Argentina, and most notably the *Pirate party of Germany* who implemented the Liquid Feedback software [4] for internal deliberation. Voting behaviour has been studied on the platform [36], and the party’s structure has been altered by its use [41]. Liquid Feedback introduced *pseudonymity* to have some level of anonymity in the system via pseudonyms [48].

Finally, *Adhocracy* has a voting platform that uses proxy voting for communities, whereas *Ethereum* has a voting platform for liquid democracy. Moreover, Google tested liquid democracy in their internal social network via *Google Votes* [33].

2 Smart voting

In this section we recall the definitions of our model for multi-agent ranked delegations in voting, which has been previously introduced as *smart voting* [19]. The model allows agents to decide if they want to vote directly on the issues at stake, or to give (possibly complex and/or multiple) delegations to determine their vote from the votes of others in the electorate.

In a smart voting election, the following four main stages will take place:

1. Each agent participating in the election creates and sends their own smart ballot (as described in Section 2.1), which could be restricted to a specific format;
2. Once the central authority has received all the ballots, they check whether the ballots abide by the aforementioned restrictions (i.e., whether they are *valid* ballots);
3. Since smart ballots may include delegations, they need to be unravelled via some procedure to obtain a standard voting profile (as presented in Section 2.2);

4. Finally, a classical voting rule, such as the majority or plurality rule, is used on the resulting standard voting profile to obtain the collective decision.

In the next sections, we will study problems which pertain to the steps 1–3 of a smart voting election. For the final step 4, any standard voting rule can be used.

2.1 Smart ballots

In a smart voting election, a finite set \mathcal{N} of n agents (or voters) has to take a collective decision over a finite set \mathcal{I} of m independent issues. The possible values, or *alternatives*, for each issue $i \in \mathcal{I}$ range over a non-empty finite domain $\mathcal{D}(i)$, which can also include abstentions (denoted by the symbol $*$).

Each agent $a \in \mathcal{N}$ expresses her vote over an issue $i \in \mathcal{I}$ by submitting what we call a *smart ballot* B_{ai} , defined as follows:

Definition 1 (Smart ballot) *A smart ballot of agent a on an issue $i \in \mathcal{I}$ is an ordering $((S^1, F^1) > \dots > (S^k, F^k) > x)$ where $k \geq 0$ and for $h \leq k$ we have that $S^h \subseteq \mathcal{N}$ is a set of agents, $F^h : \mathcal{D}(i)^{S^h} \rightarrow \mathcal{D}(i)$ is a resolute aggregation function and $x \in \mathcal{D}(i)$ is an alternative.*

For simplicity, we will focus on a single issue $i \in \mathcal{I}$. We thus drop the index i throughout to simplify notation—writing, e.g., \mathcal{D} instead of $\mathcal{D}(i)$ and B_a instead of B_{ai} . All our results can be easily generalised to multiple independent issues.

In a smart ballot, an agent is expressing a preference ordering over their desired delegations—i.e., the k domain-function pairs (S, F) —which ends with a direct backup vote x for an alternative in the domain of the issue. Note that while it is required for an agent to provide a backup vote as their last choice, that vote could be an abstention (if it is in the domain of the issue). Moreover, an agent can submit a smart ballot with $k = 0$, i.e., they vote directly on the issue without any delegation.

In most of our examples, a delegating function F either takes the form of a single-agent delegation, an aggregation rule (such as the majority rule), or a Boolean function. In the latter case we assume that the function is represented as a Boolean formula built from a set of propositional variables $\{a \mid a \in \mathcal{N}\}$, representing the use of agent a 's vote to determine the delegation, and the standard logical connectives for negation \neg , conjunction \wedge , and disjunction \vee , on a binary domain.

When computing a function F on an incomplete input we use the notion of a *necessary winner* [37]. For example, consider an agent $a \in \mathcal{N}$ having a delegation

$Maj(b, c, d)$ —i.e., the majority over the votes of b , c and d —on a binary issue for which b and c have a direct vote for 1, while there is no vote of d yet. We can still compute $Maj(b, c, d)$ without the vote of d , since there is already a majority for 1.

From our general definition of smart ballots, we further distinguish a class of *valid* smart ballots satisfying some desirable properties:

Definition 2 (Valid smart ballot) *A valid smart ballot of agent a is a smart ballot B_a such that, for all $1 \leq s \neq t \leq k$, we have that (i) if $S^s \cap S^t \neq \emptyset$ then F^s is not equivalent to F^t , and (ii) $a \notin S^s$.*

Condition (i) imposes that an agent is not submitting the same delegation at multiple preference levels of their smart ballot. This removes the possibility of manipulating the process by delegating to the same agents with the same delegation function multiple times, thus ensuring that the more preferred delegation or an equivalent formulation will be chosen. Condition (ii) ensures that agents cannot include themselves in the set of delegates, as this would immediately lead to a delegation cycle.

The following example illustrates various instances of possible smart ballots:

Example 1 *Six agents $\mathcal{N} = \{a, b, c, d, e, f\}$ face the decision of whether to order dinner from one of two new restaurants. Let us denote by 1 the first restaurant, by 0 the second one, and assume that agents can also abstain, i.e., $\mathcal{D} = \{1, 0, *\}$.*

We now present some options of valid smart ballots that agent a could submit:

(i) $B_a = (1)$

This smart ballot represents the direct vote of a for the first restaurant.

(ii) $B_a = ((\{b, c, d, e, f\}, RMaj) > 0)$

Here, agent a wants their vote to be the relative majority $RMaj$ (i.e., plurality between 0 and 1, with $$ in case of a tie) of the choices of the agents b, c, d, e, f . If this first delegation of a leads to a cycle, they will vote for the second restaurant.*

(iii) $B_a = ((\{d\}, id) > (\{e\}, id) > *)$

Here, a 's first preference is to delegate to agent d (id indicates the identity function); if this causes a delegation cycle, then a chooses to delegate to e ; and if this also causes a cycle, then a abstains from the vote.

Suppose now that we have a binary issue¹ with domain $\mathcal{D} = \{0, 1\}$ where 1 represents ordering take-away while 0 represents cooking at home.

$$(iv) \ B_a = ((\{b, f\}, b \vee f) > (\{b, c, e\}, (c \wedge b) \vee (\neg e \wedge b)) > 1)$$

In this case, a 's first choice is to delegate using the Boolean function $b \vee f$, i.e., a will vote for take-away if either b or f want to; if this creates a delegation cycle, then a will vote to try the take-away if both b and c also want to, or b wants to while e prefers to cook at home. If this still causes a cycle, then a votes for 1.

$$(v) \ B_a = ((\{b, c, f\}, Maj) > (\{b, c, e\}, (c \wedge b) \vee (\neg e \wedge b)) > 1)$$

This smart ballot differs from (iv) only in that a 's first preference is to have their vote coincide with the majority of the agents b, c and f 's votes.

One can easily check that all the ballots in Example 1 are valid as per Definition 2.

Each linear order of delegations (plus the backup vote) in a smart ballot indicates a preference over possible delegations. We write B_a^h to indicate the h^{th} preference level given by agent a in their smart ballot B_a . Hence, we have $B_a^h = (S_a^h, F_a^h)$ when the h^{th} preference level is a delegation, or $B_a^h = x$ with $x \in \mathcal{D}$ when the h^{th} preference level is a direct vote. In Example 1, e.g., $B_a^2 = (\{e\}, id)$ for ballot (iii).

We collect the n smart ballots from each agent in \mathcal{N} into a (smart) profile, i.e., a vector $\mathbf{B} = (B_1, \dots, B_n)$. A valid (smart) profile is a smart profile where each smart ballot is valid, according to Definition 2.

2.2 Unravelling procedures

An *unravelling procedure* is a function which allows us to turn a smart profile into a standard voting profile, i.e., a vector in \mathcal{D}^n of direct votes for the issue.

Definition 3 (Unravelling procedure) An unravelling procedure \mathcal{U} for the agents in \mathcal{N} is any computable function

$$\mathcal{U} : (B_1 \times \dots \times B_n) \rightarrow \mathcal{D}^n.$$

¹Note that in order to use Boolean functions to express a delegation, the domain of the alternatives for the issue must be a Boolean algebra. We restrict this to a two-element Boolean algebra, namely $\{0, 1\}$.

Thus, an unravelling procedure \mathcal{U} takes a smart profile \mathbf{B} and it returns a voting profile in \mathcal{D}^n . When \mathcal{U} and \mathbf{B} are clear from context, we often write just X to denote an outcome of an unravelling procedure: i.e., $X \in \mathcal{U}(\mathbf{B})$ for $X \in \mathcal{D}^n$.

After an unravelling procedure returns a profile of direct votes, the agents may want to know how their smart ballot was unravelled: i.e., which preference level of their ballot was actually used to compute their direct vote. For this purpose, we introduce the notion of a *certificate*.

Definition 4 (Certificate) A certificate $\mathbf{c} \in \mathbb{N}^n$ for profile \mathbf{B} is a vector where, for all $a \in \mathcal{N}$ such that $B_a = (B_a^1 > \dots > B_a^{k_a})$, the entry $\mathbf{c}_a \in [1, k_a]$ corresponds to a preference level for agent a .

Within the class of all possible certificates of Definition 4, we are interested in those that satisfy the following property: a certificate is *consistent* if there is an ordering of the agents such that an agent's vote can be determined using the preference level in the certificate, given the votes of the agents that come prior in the order.²

Definition 5 (Consistent certificate) Given a profile \mathbf{B} of valid ballots, a certificate \mathbf{c} is consistent if there exists an ordering $\sigma : \mathcal{N} \rightarrow \mathcal{N}$ of the agents which, starting from vector $X^0 = \{\Delta\}^n$ with placeholder values Δ for all agents, iteratively constructs an outcome vector of direct votes $X \in \mathcal{D}^n$ as follows, for $\sigma(a) = z \in [1, n]$:

$$X_a^z = \begin{cases} B_a^{\mathbf{c}_a} & \text{if } B_a^{\mathbf{c}_a} \in \mathcal{D} \\ F_a^{\mathbf{c}_a}(X^{z-1} \upharpoonright_{S_a^{\mathbf{c}_a}}) & \text{otherwise} \end{cases}$$

where X_a represents a 's entry in X and $X \upharpoonright_S = \Pi_{s \in S} X_s$.

We let $\mathcal{C}(\mathbf{B})$ be the set of all consistent certificates of a profile \mathbf{B} , and the a^{th} entry of \mathbf{c} corresponds to $B_a^{\mathbf{c}_a}$ being used by the unravelling procedure. Moreover, when outcomes of unravelling procedures can be determined by certificates we will use $\mathcal{C}_{\mathcal{U}}(\mathbf{B})$ to denote all consistent certificates given by the unravelling procedure \mathcal{U} . We show next that each consistent certificate \mathbf{c} has a unique corresponding outcome vector $X_{\mathbf{c}}$ of direct votes.

²This notion, when restricted to ballots with single-agent delegations, corresponds to the definition of confluent sequence rules by Brill et al. [12].

Proposition 1 *If a consistent certificate \mathbf{c} can be given by two orderings σ and σ' of the agents (as per Definition 5), then the orderings yield the same outcome $X_{\mathbf{c}} \in \mathcal{D}^n$.*

Proof 1 *Consider an arbitrary profile \mathbf{B} and a consistent certificate $\mathbf{c} \in \mathcal{C}(\mathbf{B})$. Assume for a contradiction that \mathbf{c} can yield two distinct vectors of direct votes $X \neq X'$, which are given by two orderings σ and σ' of \mathcal{N} , respectively. To reach a contradiction, we show by induction on the ordering σ that for each agent $a \in \mathcal{N}$ we have $X_a = X'_a$.*

For the base case, consider agent $a \in \mathcal{N}$ such that $\sigma(a) = 1$. As a 's vote was added to $X_{\mathbf{c}}$ without any other vote, \mathbf{c}_a must refer to a direct vote. Therefore, the direct vote of a will be added to X and X' (although it may be that $\sigma'(a) \neq 1$). We assume for our inductive hypothesis that for all agents $b \in \mathcal{N}$ where $\sigma(b) \leq k$ it is the case that $X_b = X'_b$. We will show that for agent d such that $\sigma(d) = k + 1$ we have $X_d = X'_d$. In case $B_d^{\mathbf{c}_d}$ is a direct vote, the same reasoning as for the base case applies. Else, by definition we have a necessary winner for $F_d^{\mathbf{c}_d}(X \upharpoonright_{S_d^{\mathbf{c}_d}}) = X_d$. If $X_d \neq X'_d$, then $F_d^{\mathbf{c}_d}(X \upharpoonright_{S_d^{\mathbf{c}_d}}) \neq F_d^{\mathbf{c}_d}(X' \upharpoonright_{S_d^{\mathbf{c}_d}})$ and $X \upharpoonright_{S_d^{\mathbf{c}_d}} \neq X' \upharpoonright_{S_d^{\mathbf{c}_d}}$. Hence, there exists an entry that differs in the two vectors, which contradicts our inductive hypothesis. Then, $X_d = X'_d$. As $X_a = X'_a$ for all $a \in \mathcal{N}$, we have that $X = X'$. Hence, a consistent certificate \mathbf{c} gives a unique outcome $X_{\mathbf{c}}$.

Finally, we define the *rank* of a certificate \mathbf{c} as the sum of its preference levels used. Given profile \mathbf{B} , the rank of a certificate $\mathbf{c} \in \mathcal{C}(\mathbf{B})$ is $\text{rank}(\mathbf{c}) := \sum_{a \in \mathcal{N}} \mathbf{c}_a$.

The minimum possible value of rank for an unravelling is n , i.e., when all the agents get their first preference level. Thus, if a profile contains a delegation cycle at the first preference level, it cannot have a consistent certificate with rank equal to n .

2.3 Optimal unravellings

Our first procedure, MINSUM, is *optimal* with respect to the rank: i.e., it returns all outcome vectors which can be obtained by a consistent certificate minimising the sum of preference levels used for the agents.

Definition 6 (MinSum) *For a given profile \mathbf{B} , the MINSUM unravelling procedure is defined as:*

$$\text{MINSUM}(\mathbf{B}) := \{X_{\mathbf{c}} \mid \mathbf{c} \in \arg \min_{\mathbf{c} \in \mathcal{C}(\mathbf{B})} \text{rank}(\mathbf{c})\}.$$

Hence, MINSUM returns all vectors of direct votes $X_{\mathbf{c}}$ whose consistent certificate \mathbf{c} minimises the value of $\text{rank}(\mathbf{c})$. Intuitively, by minimising the agents' preference levels used, more trusted agents are being delegated to. Next, we give examples of consistent certificates and of the outcomes of the MINSUM procedure.

Example 2 Consider a binary issue with domain $\mathcal{D} = \{0, 1\}$ and five agents $\mathcal{N} = \{a, b, c, d, e\}$, whose ballots form the profile \mathbf{B} , shown schematically in Table 1.

	B_x^1	B_x^2	B_x^3
a	$(\{b, c\}, b \wedge c)$	$(\{d\}, d)$	1
b	1	-	-
c	$(\{d\}, d)$	0	-
d	$(\{e\}, e)$	1	-
e	$(\{a\}, a)$	$(\{b\}, b)$	0

Table 1: The smart profile \mathbf{B} of Example 2, where each row represents the ballot for each of the agents $\mathcal{N} = \{a, b, c, d, e\}$, while the columns separate the different preference levels of the agents' ballots.

Note that there is a delegation cycle at the first preference level B_x^1 : agent a needs the vote of c to compute their own vote, agent c delegates to d , agent d delegates to e , and agent e delegates to a . Hence, the certificate vector $\mathbf{c} = (1, 1, 1, 1, 1)$, which would be the minimal one for this profile, is not consistent: there is no ordering of the agents where their direct votes are computed using only their first preference levels. Thus, $\mathbf{c} \notin \mathcal{C}(\mathbf{B})$ and the value of rank for a consistent certificate will be at least 6.

Consider the certificate $\mathbf{c}' = (1, 1, 2, 1, 1)$, where only c has their second preference used: \mathbf{c}' is consistent, as it is shown by the ordering $\sigma = (b, c, a, e, d)$. As $\text{rank}(1, 1, 2, 1, 1) = 6$, the corresponding outcome $X_{\mathbf{c}'} = (0, 1, 0, 0, 0)$ is an outcome of $\text{MINSUM}(\mathbf{B})$. The consistent certificate $\mathbf{c}'' = (1, 1, 1, 2, 1)$ gives $X_{\mathbf{c}''} = (1, 1, 1, 1, 1)$ and as $\text{rank}(\mathbf{c}'') = 6$, it also is in $\text{MINSUM}(\mathbf{B})$. Since there can be multiple certificates minimising the total rank (yielding distinct vectors of direct votes X) the MINSUM unravelling procedure is not resolute.

While MINSUM maximises the *global* satisfaction of the agents, from an *individual* perspective there can be a large disparity in the selected preference levels.

Our second optimal procedure is motivated by an *egalitarian* approach, finding outcomes whose certificate minimises the maximum preference level used among the agents.

Definition 7 (MinMax) *Given profile B , the MINMAX unravelling procedure returns the following vectors of direct votes:*

$$\text{MINMAX}(B) := \{X_c \mid c \in \arg \min_{c \in \mathcal{C}(B)} \max(c)\}.$$

Example 3 *Consider a binary issue and 26 agents $\mathcal{N} = \{a, \dots, z\}$. Let the profile B be such that the smart ballot of agent a is $B_a = ((\mathcal{N} \setminus \{a\}, \bigvee_{x \in \mathcal{N} \setminus \{a\}} x) > (\mathcal{N} \setminus \{a, b\}, \bigvee_{x \in \mathcal{N} \setminus \{a, b\}} x) > (\mathcal{N} \setminus \{a, b, c\}, \bigvee_{x \in \mathcal{N} \setminus \{a, b, c\}} x) > 1)$, and for each agent $x \in \mathcal{N} \setminus \{a\}$ let $B_x = ((\{a\}, a) > 0)$ be their smart ballot.*

There are three outcomes of $\text{MINMAX}(B)$, with certificates $c = (1, 2, \dots, 2)$, $c' = (2, \dots, 2)$, $c'' = (2, 1, 2, \dots, 2)$, where $\max(c) = \max(c') = \max(c'') = 2$, even though $\text{rank}(c) = \text{rank}(c'') = 51$ and $\text{rank}(c') = 52$. The outcome of $\text{MINSUM}(B)$ has certificate $c''' = (4, 1, \dots, 1)$ and $\text{rank}(c''') = 29$; however, this is not an outcome of MINMAX , since $\max(c''') = 4$.

A disadvantage of MINMAX is that for some profiles B it may return a large number of tied outcomes, as we shall see in Example 5.

2.4 Greedy unravellings

In Section 3 we will prove that computing an outcome for MINSUM and MINMAX is not computationally tractable in general. This motivates us to introduce four unravelling procedures with a greedy approach, that break delegation cycles by using the lowest possible preference level of the ballots, while keeping the process tractable.

Algorithm 1 outlines our *general unravelling procedure* UNRAVEL. The input is a smart profile B , and the procedure initialises a vector X with placeholders Δ for each agent $a \in \mathcal{N}$. The outcome X is returned when each agent has a vote in \mathcal{D} , i.e., $X \in \mathcal{D}^n$. A counter `lev` is always reset to 1 to come back to the first preference level of the agents. An additional vector Y is used to help with intermediate computations.

In line 7 a subroutine using an *update procedure* is executed.³ Given a partial profile of direct votes and placeholders Δ , as well as a preference level `lev`, the

³In the following, we will simply write $\text{UNRAVEL}(\#)$, for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$, to indicate the UNRAVEL algorithm using UPDATE procedure $\#$.

Algorithm 1 General unravelling procedure UNRAVEL

```
1: Input:  $B$ 
2:  $X := (\Delta, \dots, \Delta)$   $\triangleright$  vector for direct votes initialised with placeholders  $\Delta$ 
3: while  $X \notin \mathcal{D}^n$  do
4:    $\text{lev} := 1$   $\triangleright$  reset preference level counter  $\text{lev}$  to 1
5:    $Y := X$   $\triangleright$  store a copy of  $X$  to compute changes
6:   while  $X = Y$  do
7:     procedure UPDATE( $\#$ ) with  $\# \in \{\text{U}, \text{RU}, \text{DU}, \text{DRU}\}$ 
8:      $\text{lev} := \text{lev} + 1$ 
9:   return  $X$   $\triangleright$  output a vector of direct votes
```

UPDATE procedure searches for a direct vote or a vote that can be computed via necessary winners (depending on which UPDATE is used) at the lev^{th} preference level in the profile; if this is not possible, UNRAVEL moves to level $\text{lev} + 1$ (line 8).

The four update procedures that could be called in Algorithm 1 are defined by the presence or absence of two properties. The first is *direct vote priority* (D): an update procedure prioritises *direct votes* over those that can be computed from the current vector Y of votes. The second is *random voter selection* (R): an update procedure *randomly* selects, when possible, a single agent whose direct or computable vote can be added to X . We thus get: basic update (U), update with direct vote priority (DU), update with random voter selection (RU), update with both direct vote priority and random voter selection (DRU).

Algorithm 2 UPDATE(U)

```
1: for  $a \in \mathcal{N}$  such that  $x_a = \Delta$  do
2:   if  $B_a^{\text{lev}} \in \mathcal{D}$  then  $\triangleright$  add  $a$ 's vote if  $a$  has a direct vote at  $\text{lev}$ 
3:      $x_a := B_a^{\text{lev}}$ 
4:   else if  $F_a^{\text{lev}}(Y_{\upharpoonright S_a^{\text{lev}}}) \in \mathcal{D}$  then
5:      $x_a := F_a^{\text{lev}}(Y_{\upharpoonright S_a^{\text{lev}}})$   $\triangleright$  add  $a$ 's vote if  $a$  has a computable vote at  $\text{lev}$ 
```

The UPDATE(U) procedure⁴ in Algorithm 2 updates the vector X with the direct votes for those agents who currently do not have one (line 1), if their pref-

⁴Unless otherwise specified, in case the condition in an **if** statement fails, our programs will skip to the next step. Recall also that $Y_{\upharpoonright S}$ denotes the restriction of vector Y to the elements in set S .

erence at lev is a direct vote (line 3) or it can be computed from the current votes in Y (line 5).

Algorithm 3 UPDATE(DU)

```

1: for  $a \in \mathcal{N}$  such that  $x_a = \Delta$  do
2:   if  $B_a^{\text{lev}} \in \mathcal{D}$  then  $\triangleright$  add all direct votes
3:      $x_a := B_a^{\text{lev}}$ 
4: if  $Y = X$  then  $\triangleright$  if no direct votes are added to  $X$ 
5:   for  $a \in \mathcal{N}$  such that  $x_a = \Delta$  do
6:     if  $F_a^{\text{lev}}(Y_{\upharpoonright S_a^{\text{lev}}}) \in \mathcal{D}$  then  $\triangleright$  find and add computable votes to  $X$ 
7:        $x_a := F_a^{\text{lev}}(Y_{\upharpoonright S_a^{\text{lev}}})$ 

```

In Algorithm 3, UPDATE(DU) first adds the direct votes from preference level lev to X for those agents without a vote in X (line 2). If there are no direct voters at lev (line 4), then the procedure tries to add computable votes (line 6).

Algorithm 4 UPDATE(RU)

```

1:  $P := \emptyset$   $\triangleright$  initialise an empty set
2: for  $a \in \mathcal{N}$  such that  $x_a = \Delta$  do
3:   if  $B_a^{\text{lev}} \in \mathcal{D}$  or  $F_a^{\text{lev}}(Y_{\upharpoonright S_a^{\text{lev}}}) \in \mathcal{D}$  then
4:      $P := P \cup \{a\}$   $\triangleright$  add voters to  $P$  if their vote can be determined
5: if  $P \neq \emptyset$  then  $\triangleright$  there are direct or computable votes in  $P$ 
6:   select  $b$  from  $P$  uniformly at random
7:   if  $B_b^{\text{lev}} \in \mathcal{D}$  then
8:      $x_b := B_b^{\text{lev}}$ 
9:   else if  $F_b^{\text{lev}}(Y_{\upharpoonright S_b^{\text{lev}}}) \in \mathcal{D}$  then
10:     $x_b := F_b^{\text{lev}}(Y_{\upharpoonright S_b^{\text{lev}}})$ 

```

The UPDATE(RU) procedure has the random voter selection property (Algorithm 4): at line 1 an empty set P is initialised to store agents with either a direct vote or a computable vote at lev (line 3). If P is non-empty, one agent will be randomly selected and their direct or computable vote will be added to X .

Lastly, Algorithm 5 presents UPDATE(DRU), which has both properties. At lev , the procedure adds agents with direct votes to P (line 3) and agents with computable votes to P' . If P is not empty, an agent is selected from P and their direct vote is added to X (line 9). Otherwise, if P is empty and P' is not, an agent

Algorithm 5 UPDATE(DRU)

```
1:  $P, P' := \emptyset$  ▷ initialise two empty sets
2: for  $a \in \mathcal{N}$  such that  $x_a = \Delta$  do
3:   if  $B_a^{\text{lev}} \in \mathcal{D}$  then ▷ add agents with direct votes at lev to P
4:      $P := P \cup \{a\}$ 
5:   else if  $F_a^{\text{lev}}(Y \upharpoonright_{S_a^{\text{lev}}}) \in \mathcal{D}$  then ▷ add agents with computable votes at lev to P'
6:      $P' := P' \cup \{a\}$ 
7: if  $P \neq \emptyset$  then ▷ if there are agents with direct votes
8:   select  $b$  from  $P$  uniformly at random
9:    $x_b := B_b^{\text{lev}}$  ▷ add only the randomly selected voter's direct vote to X
10: else if  $P' \neq \emptyset$  then ▷ if there are some computable votes
11:   select  $b$  from  $P'$  uniformly at random
12:    $x_b := F_b^{\text{lev}}(Y \upharpoonright_{S_b^{\text{lev}}})$  ▷ add only the randomly selected voter's computable vote to X
```

is selected from P' and their computable vote is added to X (line 12). If both P and P' are empty, no votes are added to X and the procedure terminates.

We now give an example of the execution of these four unravelling procedures.

Example 4 For a binary issue with $\mathcal{D} = \{0, 1\}$ consider agents $\mathcal{N} = \{a, \dots, f\}$, whose ballots and delegation structure are represented schematically in Figure 1.⁵ First of all, \mathbf{B} is thus a valid profile. We now illustrate our four unravelling procedures for $\text{UNRAVEL}(\#)$ with $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$.

UNRAVEL(U)

At $\text{lev} = 1$ the procedure adds the direct votes of b and c to X . Thus, we have $X = (\Delta, 1, 0, \Delta, \Delta, \Delta)$. Then, the algorithm cannot find a direct or computable vote at $\text{lev} = 1$, so it moves to $\text{lev} = 2$ where it uses Y to add the direct votes of d and e , as well as f 's vote that is computable from X by copying b , giving $X = (\Delta, 1, 0, 0, 1, 1)$. As no other update is possible, the algorithm sets $\text{lev} = 1$ and it computes a 's vote, yielding $X = (0, 1, 0, 0, 1, 1)$, with $\mathbf{c} = (1, 1, 1, 2, 2, 2)$.

UNRAVEL(DU)

As with $\text{UNRAVEL}(\mathbf{U})$, the direct votes of b and c are added initially, which

⁵Observe that a formula of propositional logic is a Boolean function.

	B_x^1	B_x^2	B_x^3	
a	$(\{b, c, d\}, (b \wedge c) \vee (b \wedge d))$	$(\{e\}, e)$	1	$f \begin{array}{ c } \hline 1 \\ \hline \end{array} \rightarrow a \begin{array}{ c } \hline 1 \\ \hline \end{array} \dashrightarrow e \begin{array}{ c } \hline 1 \\ \hline \end{array}$
b	1	-	-	\downarrow
c	0	-	-	$b \begin{array}{ c } \hline 1 \\ \hline \end{array} \leftarrow (b \wedge c) \vee (b \wedge d) \rightarrow d \begin{array}{ c } \hline 0 \\ \hline \end{array}$
d	$(\{e\}, e)$	0	-	\downarrow
e	$(\{f\}, f)$	1	-	$c \begin{array}{ c } \hline 0 \\ \hline \end{array}$
f	$(\{a\}, a)$	$(\{b\}, b)$	1	

Figure 1: Representation of the ballots (left) and the delegation structure (right) of the agents in \mathbf{B} from Example 4. In the graph on the right, a solid line indicates the first preference for delegation, a dashed line represents the second, and the final preference (a direct vote in $\{0, 1\}$) is written next to the agents' names.

yields to $X = (\Delta, 1, 0, \Delta, \Delta, \Delta)$, and then the algorithm moves to $lev = 2$. Unlike $\text{UNRAVEL}(\mathbf{U})$, the procedure $\text{UNRAVEL}(\mathbf{DU})$ adds only the direct votes of d and e , giving $X = (\Delta, 1, 0, 0, 1, \Delta)$. Returning to $lev = 1$, a 's vote can be computed from the votes of b, c and d , giving $X = (0, 1, 0, 0, 1, \Delta)$. Finally, at $lev = 1$, f 's computable vote (a delegation to a) can be added, thus giving $X = (0, 1, 0, 0, 1, 0)$, with certificate $\mathbf{c} = (1, 1, 1, 2, 2, 1)$.

$\text{UNRAVEL}(\mathbf{RU})$

First, the direct votes of b and c are added, each in a separate iteration, giving $X = (\Delta, 1, 0, \Delta, \Delta, \Delta)$. Then, the algorithm moves to $lev = 2$, where it chooses a single vote at random to add to X from the agents d, e and f . If, for example, the vote of f was added, then $X = (\Delta, 1, 0, \Delta, \Delta, 1)$. At $lev = 1$, e 's vote can be computed from f 's, and then d 's from e 's, giving $X = (\Delta, 1, 0, 1, 1, 1)$. Then, at $lev = 1$, a 's vote can be computed from b, c and d 's, yielding $X = (1, 1, 0, 1, 1, 1)$, whose certificate is $\mathbf{c} = (1, 1, 1, 1, 1, 2)$.

$\text{UNRAVEL}(\mathbf{DRU})$

This procedure moves as $\text{UNRAVEL}(\mathbf{RU})$, except that it chooses randomly only between the direct votes of d and e at the iteration where $\text{UNRAVEL}(\mathbf{RU})$ can also choose to select the vote of agent f .

Note that in this example, MINMAX would return outcomes corresponding to all certificates \mathbf{c} where $\max(\mathbf{c}) = 2$. This would include, e.g., $\mathbf{c} = (1, 1, 1, 2, 1, 1)$,

which is also returned by MINSUM, but also $\mathbf{c}' = (2, 1, 1, 2, 2, 2)$ and many more.

2.5 Language restrictions for smart ballots

Starting from our general concept of a valid smart ballot in Definition 2, we now focus on some restrictions on the language of delegations in order to study our procedures.

We start by focusing on Boolean functions expressed as propositional formulas on a binary domain, with a few additional requirements. Firstly, we impose that the formulas are *contingent*—i.e., neither a tautology, nor a contradiction—in order to avoid a direct vote in disguise for (in the case of a tautology) or against (in the case of a contradiction) the issue, as they would always evaluate to true (respectively, to false). Secondly, the formulas must be expressed in *disjunctive normal form* (DNF): i.e., they are written as a disjunction of *cubes*, where a cube is a conjunction of *literals* (and a literal is a variable or its negation). Finally, call a cube C an *implicant* of formula φ if $C \models \varphi$, and call C a *prime implicant* of φ if C is an implicant of φ and for all other $C' \models \varphi$ we have that $C' \not\models C$. Intuitively, prime implicants are the minimal partial assignments to make a formula true. A *complete* DNF is the unique representation of a DNF listing all of its prime implicants.

This representation may seem restrictive, but for an implementation of our framework we could envisage a pre-processing step where the agents are aided by a computer platform when creating their ballots—which would use techniques such as the *consensus method* or *variable depletion* (see the textbook by Crama and Hammer [21] for further details) to find the corresponding complete DNF of a formula.

We call this restricted language **BOOL**:⁶

Definition 8 (BOOL) A smart ballot B_a for agent a and a binary issue is in language **BOOL** if every F_a^h in B_a is a contingent propositional formula in complete DNF.

Observe that a propositional atom is a Boolean function corresponding to the identity function: i.e., it is equivalent to copying another agent’s vote. In Example 1, ballot (v) does not belong to language **BOOL** as $B_a^1 = \text{Maj}$ is not a

⁶Note that in previous work [19], the language **BOOL** was initially defined simply as the language of contingent propositional formulas in DNF, for which however the necessary winners cannot be computed in polynomial time. We are grateful to an anonymous reviewer for pointing this out.

Boolean formula; however, ballot (iv) belongs to **BOOL**, but note that the formula $(b \wedge c) \vee (b \wedge \neg c) \vee f$, which is equivalent to the formula used at the first preference level, would not be in **BOOL** as it is not complete. For the language **BOOL**, we often write φ_a^{lev} instead of F_a^{lev} .

The following proposition shows that the necessary winner for **BOOL** ballots can be computed in polynomial time.

Proposition 2 *Deciding if a formula in a **BOOL** ballot has a necessary winner can be done in polynomial time.*

Proof 2 *Observe that the necessary winner for a formula being 1 (resp., 0) means that the formula is true (resp., false). We first need to prove the following two claims:*

1. *The necessary winner of a complete DNF formula is 1 if and only if every literal of at least one cube of the formula is true.*
2. *The necessary winner of a complete DNF formula is 0 if and only if every cube of the formula is made false by at least one literal.*

These two claims can be computed by reading the formula and the partial truth assignment; thus, if they are true, a necessary winner can be found in polynomial time.

For the right-to-left direction of claim (1), assume that one cube of the formula is true. As the formula is a complete DNF, each cube represents one of its prime implicants. By definition, if a prime implicant is made true, so is the formula.

For the left-to-right direction of claim (1), assume that the complete DNF formula φ is made true by some partial truth assignment X . We create a cube C from the partial assignment, where if a variable x is true (resp., false) in X then x (resp., $\neg x$) is a literal in C . As C is built from a partial truth assignment making φ true, we have that $C \models \varphi$ and thus C is an implicant of φ . Then, either C is a prime implicant of φ or there exists a prime implicant C' of φ , such that $C' \models C$, where C' contains a subset of literals in C . As φ is a complete DNF, in either case there will be a cube of φ made true by X (i.e., either C or C').

For the right-to-left direction of claim (2), if all cubes in the formula are made false, then the formula is also necessarily false (i.e., the necessary winner is 0).

For the left-to-right direction of claim (2), assume that a complete DNF φ evaluates to false under a partial truth assignment X . Yet, assume for a contradiction that there exists a cube C of φ that does not evaluate to false under X .

As C is not false, then either C is true under X (yielding a contradiction, as φ would be true), or there are some variables $v \in \text{Var}(C)$ without a truth value in X and the remaining literals are made true. We can then extend X for each such $v \in \text{Var}(C)$ such that the literal of v in C is made true. As φ is a complete DNF, the cube C would be true—as no cube can contain contradictions (e.g., x and $\neg x$). Thus, the formula φ would be true and we would reach a contradiction.

Finally, checking that each literal of at least one cube is true (or that every cube is made false by at least one literal) can be done by simply inspecting the formula together with the partial truth assignment, and thus in polynomial time.

A further advantage of having delegations expressed in complete DNF is that we can check whether a ballot is valid in polynomial time:⁷ a tautology in complete DNF is \top , a contradiction is \perp , and to check if two complete DNF formulas are equivalent it suffices to see if the lists of their prime implicants are the same.

The next language restriction that we introduce is to ranked liquid democracy ballots. The language LIQUID restricts the delegations to single other agents, where the delegation function is the identity function id .

Definition 9 (LIQUID) *Smart ballot B_a for agent a belongs to LIQUID if every delegating B_a^b is of the form $(\{b\}, id)$ for $b \in \mathcal{N} \setminus \{a\}$ and id the identity function.*

In some models of ranked liquid democracy—e.g., in Liquid Feedback [4]—the final backup vote must be an abstention (*): we denote this language as LIQUID_{*}.

Finally, for a given language \mathcal{L} we write $\mathcal{L}[k]$ to indicate the smart ballots in \mathcal{L} having at most k delegations in their ordering. For instance, in Example 1 the smart ballots (i) and (iv) belong to the language $\text{BOOL}[2]$, while ballot (iii) belongs to the language LIQUID_{*} as well as LIQUID[2]. Note that checking if a ballot is valid for LIQUID is a tractable problem as it suffices to check that all delegation functions use id and that no one delegates to themselves or to the same agent multiple times.

3 Computational complexity of unravellings

In this section we study the complexity of computational problems for each of our unravelling procedures. First, we study how hard it is to unravel a smart profile

⁷We previously showed [19] that checking if a ballot of contingent DNF formulas is valid is an NP-complete problem. Restricting formulas to contingent *complete* DNFs makes this problem tractable.

under a given procedure. We begin with MINSUM and MINMAX, showing that an associated decision problem, BOUNDEDMINSUM and BOUNDEDMINMAX, respectively, are NP-complete. However, when smart ballots are restricted to LIQUID, finding a solution becomes tractable. Unlike MINSUM and MINMAX, we show that our greedy procedures, UNRAVEL($\#$) with $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$, always terminate in a polynomial number of time steps.

3.1 Computational complexity of MINSUM

In this section we study the computational complexity of finding MINSUM outcomes, when ballots are restricted to either the BOOL or LIQUID language, finding the problem to be NP-complete in the former case and tractable in the latter.

We begin by studying the decision problem BOUNDEDMINSUM, whose input is a smart profile \mathbf{B} , such that every ballot is restricted to BOOL, and a constant $M \in \mathbb{N}$. The problem then asks if there is a consistent certificate \mathbf{c} that unravels \mathbf{B} such that $\text{rank}(\mathbf{c}) \leq M$. Repeatedly using BOUNDEDMINSUM for different values of M gives us the minimum bound, and a modified version of BOUNDEDMINSUM using partial certificates would allow us to compute an outcome of MINSUM. Both problems are harder than BOUNDEDMINSUM, which we now show being NP-complete.

Lemma 1 BOUNDEDMINSUM is in NP.

Proof 3 Recall that BOUNDEDMINSUM is defined on BOOL profiles. We prove membership in NP by showing that a witness can be checked in polynomial time. Our witness will be the certificate vector $\mathbf{c} \in \mathbb{N}^n$, such that \mathbf{c}_i represents the preference level of agent $i \in \mathcal{N}$ when unravelling the profile.

First, we check that \mathbf{c} abides by Definition 4: that is, for each $i \in \mathcal{N}$, \mathbf{c}_i corresponds to a preference level in B_i . To do this, we need to read the certificate and the profile, taking a polynomial number of time steps. Next, we check that \mathbf{c} is consistent: we first find the direct voters $B_i^{\mathbf{c}_i} \in \{0, 1\}$ from the certificate and the profile, and we add them to a set D . We construct the outcome vector $X \in \{\Delta\}^n$ and append the entry $X_i = B_i^{\mathbf{c}_i}$ for these direct voters, which can be done in polynomial time. Then, we check if any necessary winners can be computed from D : for each agent $i \in \mathcal{N} \setminus D$ such that there exists a $j \in D$ such that $j \in S_i^{\mathbf{c}_i}$, we check if we can compute a necessary winner of $F_i^{\mathbf{c}_i}$ given $X \upharpoonright_{S_i^{\mathbf{c}_i \cap D}}$. If so, we add $i \in D$ and let $X_i = F_i^{\mathbf{c}_i}(X \upharpoonright_{S_i^{\mathbf{c}_i \cap D}})$. Since all functions in the ballots are in complete DNF, by Proposition 2 we can check for a necessary winner in

polynomial time. Since at least one agent gives a direct vote, we have to check at most $n - 1$ agents' functions for a necessary winner in the first 'round'. If the certificate \mathbf{c} is consistent, at least one agent is added in each round. Therefore, we have to do at most $\sum_{k=1}^{n-1} k = \frac{(n-1)n}{2}$ polynomial checks, if a single agent is found in each round. Finally, we check in polynomial time that $\sum_{i \in \mathcal{N}} \mathbf{c}_i \leq M$. All steps can be done in polynomial time, showing that **BOUNDEDMINSUM** is in NP.

We now show that **BOUNDEDMINSUM** is NP-hard by giving a reduction from Feedback Vertex Set (FVS), a problem shown by Karp [35] to be NP-complete. The input of FVS is a directed irreflexive graph $G = (V, E)$ and a positive integer k ,⁸ and it asks if there is a subset $X \subseteq V$ with $|X| \leq k$ such that, when all vertices of X and their adjacent edges are deleted from G , the remaining graph is cycle-free.

Lemma 2 **BOUNDEDMINSUM** is NP-hard.

Proof 4 Recall that **BOUNDEDMINSUM** is defined on the language of complete DNFs. We prove the claim by reducing from Feedback Vertex Set (FVS). Given an instance (G, k) of FVS, let an instance of **BOUNDEDMINSUM** be such that $\mathcal{N} = V$, $M = |V| + k$, and for each vertex-agent $v \in V$ their ballot B_v is constructed as follows, for $O_v = \{u \in V \mid (v, u) \in E\}$ the set of outgoing edges of vertex v in G :

$$B_v = (O_v, \bigwedge_{u \in O_v} u) > 1.$$

The first delegation of each agent v is a conjunction of positive literals (hence, a complete DNF), each representing one of the outgoing edges from v in graph G . Then, the backup vote for 1 represents the removal of the vertex v in the FVS problem. For the agents $v \in V$ without any outgoing edges ($O_v = \emptyset$), their ballot is $B_v = 1$.

To show the correctness of our reduction, we first prove the following claim: a graph G is acyclic if and only if $\mathbf{c} = \{1\}^n$ is a consistent certificate for the profile \mathbf{B} given by the translation above.

For the left-to-right direction, we prove the contrapositive: assume that the certificate $\mathbf{c} = \{1\}^n$ is not consistent for \mathbf{B} . Therefore, there exists no ordering

⁸The formulation by Karp [35] is on directed graphs G which allow for reflexive edges. However, our sub-problem is also NP-complete, since a reduction can be given where the constructed graph G' adds a dummy node a' for each node a that had a reflexive edge in G , as well as the edges (a, a') and (a', a) .

of the agents such that all their votes can be added by using the votes added previously following the ordering. This means that there exists a delegation cycle between the formulas at the first preference level of some agents in \mathbf{B} , as at least two agents require each others' votes to determine their own. Since by construction of \mathbf{B} , the literals in the formulas represent the outgoing edges in G , the graph G is not acyclic.

For the right-to-left direction, let $\mathbf{c} = \{1\}^n$ be a consistent certificate for \mathbf{B} . First, note that all nodes in G representing non-delegating agents in \mathbf{B} have no outgoing edges. Second, for each delegating agent $v \in V$, since they can only possibly be assigned a vote for 1 (their backup direct vote), the truth assignment to the formula $\bigwedge_{u \in O_v} u$ can only be determined when all agents in the delegation have been assigned a vote. Hence, there can be no delegation cycles within the first preferences, as this would entail that a second preference must be used. Thus, every maximal path in G starting from a node v ends in a node without any outgoing edges (a node representing a direct voter). Therefore, G is acyclic.

We now prove the reduction using the previous claim. First, assume that there exists a subset X such that $|X| \leq k$ and the resulting graph has no cycles: we want to show that $\text{rank}(\mathbf{B}) \leq M = |V| + k$. If all of the agents in X receive their second preference, then all of the agents in $\mathcal{N} \setminus X$ get their first preference. Since this subset is acyclic, it is also consistent (given our claim above), and the addition of direct voters does not impact the consistency of a certificate. The rank of this unravelling is $|V| + |X| \leq |V| + k$ and therefore, $\text{rank}(\mathbf{B}) \leq M = |V| + k$.

Next, we show that if $\text{rank}(\mathbf{B}) \leq M = |V| + k$, then there exists a subset X such that $|X| \leq k$ and the remainder of G without X is cycle-free. We let \mathbf{c} be the certificate of unravelling \mathbf{B} such that the rank is less than or equal to $|V| + k$. From \mathbf{c} , we build $X = \{u \mid \text{rank}(\mathbf{c}_u) = 2\}$. We remove the agents in X from the profile, both their ballots and any mention of them in delegations. Note that since $\text{rank}(\mathbf{B}) \leq M = |V| + k$, it must be the case that $|X| \leq k$. Thus, the restriction of the certificate to $v \in \mathcal{N} \setminus X$ must be such that $\mathbf{c}_v = 1$. We can now use the claim above to state that the resulting graph with nodes $V \setminus X$ is acyclic.

Therefore, BOUNDEDMINSUM is NP-hard.

Lemmas 1 and 2 together give us the following theorem.

Theorem 1 BOUNDEDMINSUM is NP-complete.

Remark 1 The reduction in the proof of Lemma 2 does not use negated literals in the ballots: thus, BOUNDEDMINSUM would still be NP-complete if we were

to further restrict **BOOL** to contingent complete DNF formulas with only positive literals.

The proof of our next result uses *Edmonds' algorithm* [25].⁹ This algorithm finds, for a given weighted directed graph, a minimum *arborescence tree*, i.e., a directed rooted tree minimising the sum of the weights of the edges in the tree.¹⁰

Edmond's algorithm takes as input a (pre-processed) weighted directed graph $D = (V, E, w)$ and a root $r \in V$, where V is a set of vertices (or nodes), E is a set of edges, and w is a vector of the edges' weights. At each step the algorithm picks a vertex $v \in V \setminus \{r\}$ that does not have yet an incoming edge in the arborescence tree and it adds an incoming edge of this vertex having minimum weight. After each edge has been added, the algorithm checks if a cycle has formed: if that is the case, the nodes involved in the cycle are *contracted* to a single node v_C creating a new directed graph D' . The algorithm continues until the contracted graph is a directed spanning tree, and then all of the contractions are expanded.

The contraction of cycles is performed as follows. Given a set of nodes C involved in a cycle, we let $V' = (V \setminus C) \cup \{v_C\}$, for v_C a new node. In case $e_{uv} \in E$ for $u \notin C$ and $v \in C$, we let $e_{uv_C} \in E'$ such that $w(e_{uv_C}) = w(e_{uv}) - w(e_{wv})$ where e_{wv} is the lowest weighted incoming edge of v (the weight of e_{uv_C} corresponds to the incoming weight to the cycle, minus the lowest weighted incoming weight to node v in the cycle). In case $e_{vu} \in E$ for $v \in C$ and $u \notin C$, we let $e_{v_C u} \in E'$ with $w(e_{v_C u}) = w(e_{vu})$. All edges whose nodes are not in the cycle C remain unchanged.

Theorem 2 *An outcome in $\text{MINSUM}(\mathbf{B})$ on a profile \mathbf{B} in LIQUID can be found in $\mathcal{O}(n(d + n))$ time, where d represents the number of delegations in \mathbf{B} .*

Proof 5 *The proof idea is to create a graph on which to apply Edmonds' algorithm [25]. For a profile \mathbf{B} of LIQUID ballots, we construct a directed graph $D = (V, E)$, where $V = \mathcal{N} \cup \{r\}$ for a fresh node r . For the edges in E , we let $e_{ji} \in E$ if $B_i^k = (\{j\}, j)$ for some k : i.e., we add an edge from j to i if i was delegating to j at i 's k^{th} preference level. Furthermore, we add an edge $e_{ri} \in E$ for all $i \in \mathcal{N}$, representing the final direct vote of each voter. The weight of each edge is always given by the preference level of the delegation: if $B_i^k \in \mathcal{D}$ then $w(e_{ri}) = k$, and if $B_i^k = (\{j\}, j)$ then $w(e_{ji}) = k$.*

⁹Also independently suggested by Chu [16] and Bock [7].

¹⁰For undirected graphs, the corresponding problem is that of finding a minimum spanning tree.

Edmonds' algorithm returns the arborescence tree $A = (V, E')$ rooted at r in time $\mathcal{O}(|V| \times |E|)$, minimising its weight $w(A) = \sum_{e_{ij} \in E'} w(e_{ij})$. Note that by applying Edmonds' algorithm to the graph D above, we find an unravelling of \mathbf{B} , whose certificate vector \mathbf{c} minimises $\text{rank}(\mathbf{c})$. Moreover, since it returns a tree which includes every node, there are no delegation cycles and every agent has one of their preference levels used: hence, the unravelling is consistent.

Thus, we can find a solution of MINSUM in $\mathcal{O}(|V| \times |E|) = \mathcal{O}((n+1) \times (d+n))$ time steps, since $|V| = n+1$ (all the agents plus the vertex r), and $|E| = d+n$, where d represents the number of delegations in \mathbf{B} . To simplify the bound, this can be done in $\mathcal{O}(n(d+n))$ time steps.

We can thus find an optimal unravelling of a LIQUID smart profile in a polynomial number of time steps. Furthermore, as the Edmonds' algorithm is recursive, we are guaranteed that it will terminate giving an optimal unravelling, provided that there is some tie-breaking rule when there are many optimal unravellings. We now illustrate in an example the application of Edmonds' algorithm in the proof of Theorem 2.

Example 5 We show the application of Edmonds' algorithm in the proof of Theorem 2 to get a MINSUM outcome associated to profile \mathbf{B} in Table 2. The directed graph $D = (V, E, w)$ has nodes $V = \{a, b, c, d, e, r\}$, edges $E = \{(ra), (cb), (ab), (rb), (dc), (ec), (rc), (bd), (ed), (rd), (re)\}$, and weights $w(ra) = w(cb) = w(dc) = w(bd) = w(re) = 1$, $w(ab) = w(ec) = w(ed) = 2$, and $w(rb) = w(rc) = w(rd) = 3$. The graph D is shown on the left of Figure 2, with solid, dashed, and dotted lines representing first, second, and third preference levels, respectively.

	B_x^1	B_x^2	B_x^3
a	1	-	-
b	$(\{c\}, c)$	$(\{a\}, a)$	*
c	$(\{d\}, d)$	$(\{e\}, e)$	*
d	$(\{b\}, b)$	$(\{e\}, e)$	*
e	0	-	-

Table 2: The LIQUID profile that in Example 5 is unravelled via Edmonds' algorithm, and in Example 6 by the algorithm from Theorem 4.

In Figure 2, we see at the bottom of D that there is a cycle among the nodes b, c and d , among the edges representing the first preference levels. Edmonds'

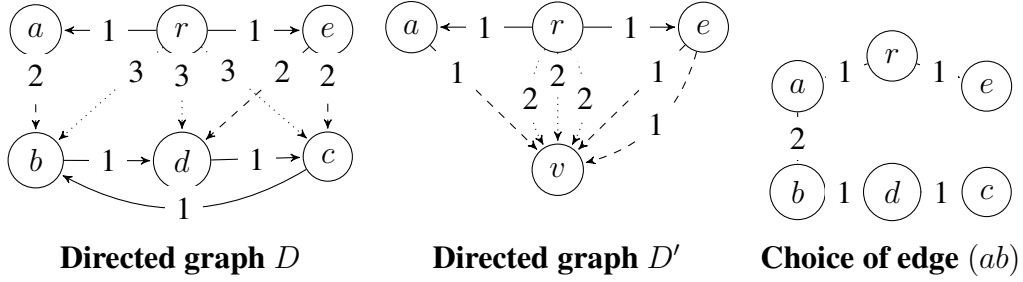


Figure 2: Three stages of unravelling the LIQUID profile B from Table 2 by using Edmonds' algorithm. The directed graph D (left) represents the initial profile. In D' (centre), the nodes b, c and d are contracted into v , as they were in a cycle in D . The arborescence tree (right) is the output where the edge (ab) was chosen to break the tie, and it corresponds to an outcome of MINSUM on B .

algorithm contracts this cycle to a single vertex v , creating a second directed graph $D' = (V', E', w)$, in the centre of Figure 2. The nodes of D' are $V' = \{a, v, e, r\}$; while for the edges E' , we keep (ra) and (re) but we alter the edges coming into and out of the cycle. However, note that there are only incoming edges to the cycle: $(ab), (rb), (rd), (rc), (ed), (ec)$ and thus we add to E' only edges coming into v , taking into account the lowest weighted incoming edge to each node in the cycle.

For the edge $(rb) \in E$, we thus have an edge $(rv) \in E'$ whose weight is computed as $w(rv) = w(rb) - w(xb)$ where $w(xb)$ is the weight of the lowest incoming edge of b , e.g., (cb) , which has weight $w(cb) = 1$. Thus, $w(rv) = 3 - 1 = 2$, and analogously for (rc) and (rd) . For the edge $(ed) \in E$, we have an edge $(ev) \in E'$ whose weight is $w(ev) = w(ed) - w(xd) = 2 - 1 = 1$, and similarly for (ec) . Finally, for $(ab) \in E$, we have an edge $(av) \in E'$ with weight $w(av) = w(ab) - w(xb) = 1$.

Since there are no cycles in D' , we can find an arborescence tree of D' rooted at r with edges $(ra), (re)$ and then either (av) or (ev) , as they both have the lowest weight of 1. Suppose that (av) is chosen: this represents the delegation from b to a with weight 2 and in the arborescence tree a will be followed by b —this unravelling is shown on the right hand-side of Figure 2. From here the unravelling continues, until all of the vertices of the cycle have been chosen; giving the edges (bd) and (dc) . Alternatively, the algorithm could have chosen the edges (ec) or (ed) instead of (ab) : all of these unravellings are optimal, with a total weight of 6.

3.2 Computational complexity of MINMAX

We study here the computational complexity of the MINMAX rule, showing that like MINSUM: it is NP-hard for the language BOOL, and tractable for LIQUID. We begin by studying the problem BOUNDEDMINMAX, which takes as input a valid smart profile \mathbf{B} restricted to BOOL as well as a constant M , and it asks whether there is an unravelling given by a certificate \mathbf{c} such that $\max_{a \in \mathcal{N}}(\mathbf{c}) \leq M$. We first show membership in NP and then NP-hardness.

Lemma 3 BOUNDEDMINMAX is in NP.

Proof 6 Recall that BOUNDEDMINMAX is defined on BOOL profiles. To prove membership in NP we can check in polynomial time that a certificate vector \mathbf{c} abides by Definition 4 and is consistent, as we did for Lemma 1. Then, we need to check that all entries in the certificate are less than or equal to the constant M , which can be done in polynomial time.

Lemma 4 BOUNDEDMINMAX is NP-hard.

Proof 7 Recall that BOUNDEDMINMAX is defined on the language of BOOL where delegations are expressed as complete DNFs. We reduce from the NP-complete problem CNF-SAT which has as input a formula φ in CNF—i.e., a conjunction of clauses (disjunctions of literals)—and it asks if there exists a satisfying assignment for φ .

For a given formula φ in CNF, let $C = \{c \mid c \text{ is a clause of } \varphi\}$ be a set of variables c , each one representing one of the clauses of φ . Construct now an instance of BOUNDEDMINSUM where $M = 2$ and the set of agents is $\mathcal{N} = \{x, y\} \cup C \cup \text{Var}(\varphi)$, for x and y fresh variables. The ballots are defined as follows:

- $B_x = (1)$,
- $B_v = ((\{x\}, x) > 0)$ for all $v \in \text{Var}(\varphi)$,
- $B_y = ((\{x\} \cup C, x \wedge \bigwedge_{c \in C} c) > (C, \bigwedge_{c \in C} c) > 1)$,
- $B_c = ((\{y\}, y) > (\{y\} \cup \text{Var}(c), y \vee \bigvee_{l \in c} l) > 1)$ for all $c \in C$,
where $l \in c$ represents the literal l of clause c , and if c contains a variable and its negation (i.e., c is a tautology), we remove the second delegation.

Note that each delegation is a complete DNF since it is either a cube or a clause.

Assume that φ is satisfiable. Then, each agent $v \in \text{Var}(\varphi)$ gets their first preference if in the satisfying truth assignment of φ the variable v is true ($\mathbf{c}_v = 1$), and their second preference if v is false ($\mathbf{c}_v = 2$). The satisfying truth assignment of φ makes every clause $c \in C$ true, and therefore one literal in $y \vee \bigvee_{l \in c} l$ is made true, making the whole formula true. Thus, agents $c \in C$ cannot receive higher than their second preference ($\mathbf{c}_c \leq 2$). Agent y can receive their first preference ($\mathbf{c}_y = 1$), given that we can determine the vote of each $c \in C$. Finally, agent x receives their first preference ($\mathbf{c}_x = 1$). Therefore, if φ is satisfiable, there is an unravelling such that every agent receives at most their second preference level in the ballot.

Next, assume that φ is not satisfiable. For a contradiction, assume that there exists an unravelling with certificate \mathbf{c} such that $\max(\mathbf{c}) \leq 2$. Since $\mathbf{c}_x = 1$ and for all $v \in \text{Var}(\varphi)$ we have $\mathbf{c}_v \leq 2$, it must be the case that either $\mathbf{c}_y = 1$ or $\mathbf{c}_y = 2$.

If $\mathbf{c}_y = 1$ then either all clauses $c \in C$ evaluate to 1 or there exists a $c \in C$ whose vote is 0. In the latter case, this has to come from c 's first or second preference. It cannot be c 's first delegation, as in this scenario y 's vote is determined by c 's and thus the unravelling would not be consistent. Their second preference can only be 0 if all of the literals of c and y are false, which without the vote of y cannot be determined: thus, we have reached a contradiction. However, if the votes of all of $c \in C$ are 1 this either means that each $c \in C$ can be made true (using the second preference delegation) and therefore φ is satisfiable, or the third preference of $c \in C$ have been used and therefore, $\max(\mathbf{c}) > 2$. In both cases we reach a contradiction.

The same reasoning holds for $\mathbf{c}_y = 2$, and thus this concludes the proof.

Lemmas 3 and 4 together give us the following:

Theorem 3 BOUNDEDMINMAX is NP-complete.

Next we study the complexity of finding a MINMAX solution on LIQUID ballots.

Theorem 4 An outcome of MINMAX on a profile \mathbf{B} in LIQUID can be found in time $\mathcal{O}(n^2 \ell^2)$, where ℓ is the highest preference level of any agent in the profile.

Proof 8 We provide an algorithm to find a MINMAX outcome, by transforming the profile \mathbf{B} into a directed graph and then finding an arborescence tree.

Construct a directed graph $G = (V, E, w)$ where $V = \mathcal{N} \cup \{r\}$, G is rooted at r , and the set of edges E with weights w is constructed iteratively from $E = \emptyset$. Starting from $\text{lev} = 1$ until $\text{lev} = \ell$, where ℓ is the maximum preference level given by any agent in \mathbf{B} , the following procedure is executed:

1. Add to the current set E an edge e_{ij} if $B_i^{\text{lev}} = (\{j\}, j)$, and an edge e_{ri} if $B_i^{\text{lev}} \in \mathcal{D}$. Namely, $E := E \cup \{e_{ji} \mid B_i^{\text{lev}} = (\{j\}, j)\} \cup \{e_{ri} \mid B_i^{\text{lev}} \in \mathcal{D}\}$. In both cases, let $w(e_{ij}) = w(e_{ri}) = \text{lev}$;
2. Check in $\mathcal{O}(|V| + |E|)$ time (see, e.g., [20], pg. 606) if there is a path from r to a via E , for $a \in \mathcal{N}$ (hence, this step has to be repeated n times, for each $a \in \mathcal{N}$). Then, if r is connected to all $a \in \mathcal{N}$, exit the loop; otherwise, if there is some $a \in \mathcal{N}$ not connected to r , let $\text{lev} := \text{lev} + 1$.

After the execution of this iterative procedure, we thus obtain a graph G where all nodes are connected to r . Then, we can find any arborescence tree from E rooted at r in $\mathcal{O}(|V| + |E|)$ time [39, pg. 19]. The certificate \mathbf{c} for an outcome of MINMAX is then given by \mathbf{c}_a being the incoming weight of a in the arborescence tree, for all $a \in \mathcal{N}$. Intuitively, we obtain a MINMAX outcome since the root r represents direct votes, if there are paths from r to any agent we can determine their votes, and since the edges are added iteratively we know that a path does not exist for a lower preference level.

As all agents give a backup vote, there will eventually be $e_{ri} \in E$ to add for $i \in \mathcal{N}$ and thus the algorithm always terminates. Since the loop iterates at most ℓ times, and each time it makes n checks, each bounded by $\mathcal{O}(|V| + |E|)$, it overall takes at most $\mathcal{O}(n\ell(|V| + |E|))$. Since $|V| = n + 1$ and $|E| \leq n\ell$, the time bound is $\mathcal{O}(n\ell(n + 1 + n\ell))$. In $\mathcal{O}(|V| + |E|) = \mathcal{O}(n + 1 + n\ell)$ an arborescence tree is found. Thus, a solution can be found in $\mathcal{O}((n\ell + 1)(n + 1 + n\ell))$ time steps, which can be simplified to $\mathcal{O}(n^2\ell^2)$.

We now show in an example the application of the algorithm from the proof of Theorem 4 to find a MINMAX outcome on a LIQUID profile.

Example 6 Consider the LIQUID profile in Table 2. We construct the directed graph $D_1 = (V, E_1, w)$, shown in Figure 3 (left), where $V = \{a, b, c, d, e, r\}$ and E_1 are the edges added when considering $\text{lev} = 1$. Since the nodes b, d and c are not connected to the root r in D_1 , we set $\text{lev} = 2$ and we create the

graph $D_2 = (V, E_2, w)$, shown in Figure 3 (right). The set E_2 thus contains edges representing all of the first and second preference levels. Since in D_2 there is a path from r to every other node, we search for an arborescence tree that will represent a MINMAX outcome, e.g., via a depth-first algorithm. One such tree has edges $\{(ra), (re), (ab), (bd), (dc)\}$.

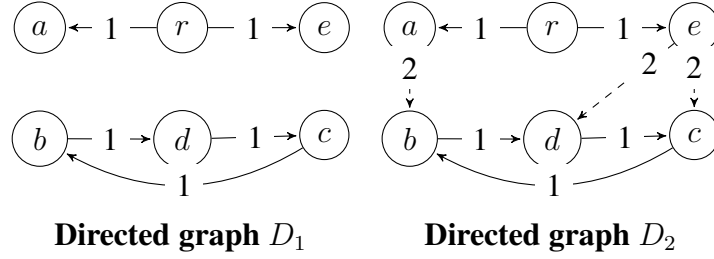


Figure 3: Application of the algorithm in the proof of Theorem 4 to the LIQUID profile B from Table 2. The directed graph D_1 (left) shows the first iteration of the algorithm for the first preference levels of the agents. As in D_1 there is not a path from r to every other node, the algorithm moves to the second iteration, constructing D_2 (right), which shows the agents' first and second preference levels. Since D_2 is connected, the algorithm terminates.

3.3 Computational complexity of greedy unravellings

We show here that $\text{UNRAVEL}(\#)$ always terminates when paired with any update procedure $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$, given a valid profile. Next, we show that they are all tractable algorithms, terminating in a polynomial number of time steps.

Proposition 3 *Algorithms $\text{UNRAVEL}(\#)$ with $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ always terminate on a valid smart profile B .*

Proof 9 *Let B be a valid smart profile for n agents. For the sake of a contradiction, assume that $\text{UNRAVEL}(\#)$ by Algorithm 1 does not terminate on B . Hence, UNRAVEL cannot exit the **while** loop from either line 6, due to no direct votes being computable at any preference level, or from line 3, due to $X \notin \mathcal{D}^n$.*

*Consider UNRAVEL being unable to terminate due to a cycle involving the **while** loop from line 6. Let $A = \{a \in \mathcal{N} \mid x_a = \Delta\}$ be the set of agents whose votes have not been computed due to a cycle. As B is a valid smart profile, we*

know that for all $a \in A$, B_a has a finite number of preference levels¹¹ and the final preference is a direct vote. In each of the update procedures (**U**, **DU**, **RU** and **DRU**), after a finite number of loops, we will reach a direct vote of an agent in A . Each of the update procedures will add at least one direct vote to X at this point, breaking this cycle. Moreover, no procedure replaces a vote in X with Δ or with any value not in \mathcal{D} .

Therefore, if the algorithm does not terminate, it must be due to the **while** loop at line 3. This can only happen while $X \notin \mathcal{D}^n$. However, as we can exit the cycle from line 6, the algorithm always changes some $x_a = \Delta$ to a vote in \mathcal{D} . Thus, after a finite number of iterations we will have that $X \in \mathcal{D}^n$ and UNRAVEL terminates.

Next, we show that our unravelling procedures terminate in polynomial time on **BOOL** ballots. Recall that the delegations in a **BOOL** ballot are Boolean functions φ expressed in complete DNF. The size of the input for UNRAVEL($\#$) for smart ballots in **BOOL** is in $\mathcal{O}(\max_p(\mathbf{B}) \cdot n \cdot \max_\varphi(\mathbf{B}))$, where $\max_p(\mathbf{B})$ is the highest preference level of any ballot in \mathbf{B} and $\max_\varphi(\mathbf{B})$ is the maximum length of any formula in \mathbf{B} .

Proposition 4 UNRAVEL($\#$) for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ terminates in at most $\mathcal{O}(n^2 \cdot \max_p(\mathbf{B}) \cdot \max_\varphi(\mathbf{B}))$ time steps, on a valid smart profile \mathbf{B} in **BOOL**.

Proof 10 The **while** loop from line 3 in UNRAVEL (see Algorithm 1) can be repeated at most n times (when a single vote is added to X at each iteration). Moreover, the **while** loop from line 6 can be repeated at most $\max_p(\mathbf{B})$ times, when all smart ballots are of the same length and no vote is computable in the first $\max_p(\mathbf{B}) - 1$ iterations.

The following is executed at most $n \cdot \max_p(\mathbf{B})$ times. UNRAVEL($\#$) checks that for each agent a such that $x_a = \Delta$ (at most $n - 1$) either $B_a^{\text{lev}} \in \mathcal{D}$ or φ_a^{lev} has a necessary winner (depending on the update procedure used). As each φ_a^{lev} is a complete DNF, to verify if it has a necessary winner we check if either (i) all literals of a cube of φ_a^{lev} are made true by $X \upharpoonright_{S_a^{\text{lev}}}$, or (ii) one literal in each cube is made false by $X \upharpoonright_{S_a^{\text{lev}}}$, returning a direct vote of 1 or 0, respectively, as described in Proposition 2.

The use of UNRAVEL($\#$) takes at most $\mathcal{O}(n \cdot 2 \max_\varphi(\mathbf{B}))$ steps, which is equivalent to $\mathcal{O}(n \cdot \max_\varphi(\mathbf{B}))$ steps. Thus, UNRAVEL($\#$) with $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ yields a vector X of direct votes in $\mathcal{O}(n^2 \cdot \max_p(\mathbf{B}) \cdot \max_\varphi(\mathbf{B}))$ time steps.

¹¹Recall that since both \mathcal{D} and the possible sets of delegates are finite, and since all functions given in an agent's valid ballot must differ, the possible number of functions must also be finite.

4 Comparing the unravelling procedures

In this section we complement the results of Section 3, which analysed the computational complexity of our unravelling procedures, with the aim to further distinguish our defined unravellings, to understand when a procedure would be preferable.

4.1 Restrictions yielding distinct or identical outcomes

We study here under which restrictions on the language of the ballots, the outcomes of our unravelling procedures coincide or differ. First, we show that all unravelling procedures defined in Subsection 2.2 can give different outcomes, even when the ballots are restricted to LIQUID.

Proposition 5 *The unravellings $\text{UNRAVEL}(\#)$, for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$, MINSUM and MINMAX can give different certificates and outcomes on the same smart profile \mathbf{B} of LIQUID ballots.*

Proof 11 *Consider the LIQUID profile \mathbf{B} for the domain $\mathcal{D} = \{1, 0\}$ and the set of agents $\mathcal{N} = \{a, b, c, d\}$ presented in Table 3 (left). The outcomes of the unravelling procedures and their certificates are also shown in Table 3 (right). However, we do not show the outcomes of MINMAX , since it returns all consistent certificates such that no entry is greater than 3; e.g., it will also include the certificate $\mathbf{c} = (3, 3, 2, 2)$ giving the outcome $(1, 0, 0, 1)$. Since the latter is not an outcome of any of the other procedures, MINMAX differs from those. Moreover, while procedures $\text{UNRAVEL}(\mathbf{RU})$ and $\text{UNRAVEL}(\mathbf{DRU})$ give the same outcomes $(1, 1, 1, 1)$ and $(0, 0, 0, 1)$, they are returned at different rates.*

We now show that when we restrict the ballots to $\text{LIQUID}[1]_*$ the outcome is the the same for all our unravelling procedures, except for MINMAX .

Proposition 6 *If $\mathbf{B} \in \text{LIQUID}[1]_*$, the procedures MINSUM and $\text{UNRAVEL}(\#)$ for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ give the same outcome X , but the certificate may differ.*

Proof 12 $\text{UNRAVEL}(\mathbf{U})$ and $\text{UNRAVEL}(\mathbf{DU})$ act in an identical manner for $\text{LIQUID}[1]_*$ ballots. They first add all non-delegating agents' votes to X . Then, they iteratively unravel the first preference delegations of all agents who are not in a cycle. Once no more votes can be added from the first preference level, i.e., there are agents in

	B_x^1	B_x^2	B_x^3	B_x^4
a	$(\{b\}, b)$	$(\{c\}, c)$	$(\{d\}, d)$	1
b	$(\{a\}, a)$	$(\{c\}, c)$	0	-
c	$(\{a\}, a)$	$(\{b\}, b)$	1	-
d	$(\{a\}, a)$	1	-	-

Procedure	Outcome	Certificate
U	(1, 0, 1, 1)	(3, 3, 3, 2)
DU	(0, 0, 1, 1)	(1, 3, 3, 2)
RU	(1, 1, 1, 1)	(3, 1, 1, 2)
	(0, 0, 0, 1)	(1, 3, 1, 2)
	(1, 1, 1, 1)	(2, 1, 3, 2)
	(1, 1, 1, 1)	(1, 2, 3, 2)
DRU	(0, 0, 0, 1)	(1, 3, 1, 2)
	(1, 1, 1, 1)	(2, 1, 3, 2)
	(1, 1, 1, 1)	(1, 2, 3, 2)
MINSUM	(0, 0, 0, 0)	(1, 3, 1, 1)

Table 3: On the left, we show the profile B used in the proof of Proposition 5. On the right, the table shows the outcomes and certificates of unravelling profile B with the procedures $\text{UNRAVEL}(\mathbf{U})$, $\text{UNRAVEL}(\mathbf{DU})$, $\text{UNRAVEL}(\mathbf{RU})$, $\text{UNRAVEL}(\mathbf{DRU})$, and MINSUM .

a delegation cycle, these agents are assigned their second choice, i.e., the abstention $$.*

$\text{UNRAVEL}(\mathbf{RU})$ picks one agent at a time from the first preference level who either gives a direct vote, or their delegate has a vote in X . When no more agents are available at the first preference level, the remaining agents are in delegation cycles. Moving to the second preference level, one of these agents will be added with an abstention. Consequently, everyone caught in this delegation cycle will also receive abstentions from the agent who was picked at random. This is repeated until all cycles have been resolved and all agents have a vote in X .

$\text{UNRAVEL}(\mathbf{DRU})$ first adds one by one the direct votes of the agents who do not delegate. Then, it does the same for delegating agents whose delegate already has a vote in X . Once there are no more agents to add from their top

preference, the procedure adds a single random agent with an abstention (from their second preference) and then it continues as for $\text{UNRAVEL}(\mathbf{RU})$, until all delegation cycles are resolved.

Finally, MINSUM returns all outcomes that minimise the total rank. Therefore, all agents will receive their first preference, except for a single agent from each delegation cycle, as in the previous unravellings. Observe that on any profile \mathbf{B} in $\text{LIQUID}[1]_{*,*}$ we have $\mathcal{C}_{\text{MINSUM}}(\mathbf{B}) = \mathcal{C}_{\text{UNRAVEL}(\mathbf{RU})}(\mathbf{B}) = \mathcal{C}_{\text{UNRAVEL}(\mathbf{DRU})}(\mathbf{B})$.

Remark 2 All our six unravelling procedures will have the certificate $\mathbf{c} = \{1\}^n$ on $\text{LIQUID}[1]_{*,*}$ profiles with no delegation cycles. However, if there are cycles, MINMAX will return many outcomes—including the one whose certificate gives to all delegating agents their second preference (*), regardless of if they are in a delegation cycle or not. Furthermore, Proposition 6 does not hold for $\text{LIQUID}[1]$, where backup votes are not restricted to *, as the tie-breaking affects the outcome X .

Remark 3 The breadth-first and depth-first rules by Kotsialou and Riley [38] differ from all six of our unravellings. Consider $\mathcal{N} = \{a, b, c\}$, an issue with domain $\mathcal{D} = \{1, 0, *\}$, and agents' ballots as follows: $B_a = ((\{b\}, b) > (\{c\}, c) > *)$, $B_b = (*)$, and $B_c = (1)$. Our six unravelling procedures would return the outcome $(*, *, 1)$ with certificate $\mathbf{c} = (1, 1, 1)$, whereas the breadth-first and depth-first procedures would return the outcome $(1, *, 1)$ with certificate $\mathbf{c} = (2, 1, 1)$.

Next, we show that all possible outcomes of $\text{UNRAVEL}(\mathbf{DRU})$ are also possible outcomes of $\text{UNRAVEL}(\mathbf{RU})$, as the set of certificates of the former is a subset of the set of certificates of the latter.

Proposition 7 $\mathcal{C}_{\text{UNRAVEL}(\mathbf{DRU})}(\mathbf{B}) \subseteq \mathcal{C}_{\text{UNRAVEL}(\mathbf{RU})}(\mathbf{B})$ for valid smart profiles \mathbf{B} .

Proof 13 At any iteration of $\text{UNRAVEL}(\mathbf{RU})$, the random choice can either include direct voters or not. If there are direct voters, $\text{UNRAVEL}(\mathbf{DRU})$ will thus have a subset of the choices of $\text{UNRAVEL}(\mathbf{RU})$ (and hence potential outcomes). If there are no direct voters at an iteration, the potential outcomes from this step are the same for $\text{UNRAVEL}(\mathbf{RU})$ and $\text{UNRAVEL}(\mathbf{DRU})$. Thus, all certificates of $\text{UNRAVEL}(\mathbf{DRU})$ will also be certificates of $\text{UNRAVEL}(\mathbf{RU})$.

4.2 Participation axioms

In this subsection we study two properties of resolute unravelling procedures, focusing on a binary domain (with abstentions) $\mathcal{D} = \{0, 1, *\}$. Both properties were

proposed by Kotsialou and Riley [38] and they focus on a voter's incentive to participate in the election, either by voting directly or by delegating, in line with the classical participation axiom from social choice (see, e.g., [45]).

We assume that an agent a expressing a direct vote for $x \in \{0, 1\}$ prefers x over $1 - x$ and over an abstention, denoted by $x >_a 1 - x$ and $x >_a *$, respectively. Furthermore, we focus on resolute rules to directly compare the breadth-first and depth-first procedures to our own, as the participation axioms were originally constructed to study these procedures.

First, we make a distinction between the unravelling procedures being *resolute*, as our greedy procedures, or *irresolute*, as our optimal procedures. By the former, a unique outcome is returned, and by the latter, possibly many tied outcomes are returned. Although throughout the paper we present *all* outcomes of the greedy procedures with random voter selection **RU** and **DRU** (see, e.g., the outcomes displayed in Table 3), these procedures are resolute as defined in Algorithms 4 and 5.

Definition 10 (Cast-Participation) *A resolute voting rule r and a resolute unravelling procedure \mathcal{U} satisfy cast-participation if for all valid smart profiles \mathbf{B} and agents $a \in \mathcal{N}$ such that $B_a \in \mathcal{D} \setminus \{*\}$ we have for all $B'_a \neq B_a$*

$$r(\mathcal{U}(\mathbf{B})) \geq_a r(\mathcal{U}(\mathbf{B}_{-a}, B'_a))$$

where \mathbf{B}_{-a} is equal to \mathbf{B} without a 's ballot. For randomised procedures we require the inequality to hold for any possible outcome of \mathcal{U} .

Cast-participation implies that agents who vote directly have an incentive to do so, rather than to express any other ballot. In order to prove if a pair of an unravelling procedure and an aggregation rule satisfies such a participation axiom, we need some further notation. Let the set of voters *influenced* by a voter a in a profile \mathbf{B} using a resolute deterministic unravelling procedure \mathcal{U} be $I^\mathcal{U}(\mathbf{B}, a) = \{b \mid a \in S_b^k \text{ for } \mathcal{U}(\mathbf{B}) = X_{\mathbf{c}} \text{ with } \mathbf{c} \in \mathcal{C}(\mathbf{B}) \text{ and } \mathbf{c}_b = k\}$. Further, let $I_*^\mathcal{U}(\mathbf{B}, a) = I^\mathcal{U}(\mathbf{B}, a) \cup \{c \mid c \in I^\mathcal{U}(\mathbf{B}, b) \wedge b \in I^\mathcal{U}(\mathbf{B}, a)\} \cup \dots$ be the voters who are influenced by a both *directly* and *indirectly*.

Given our domain $\mathcal{D} = \{0, 1, *\}$, we consider the following rules: the *majority rule* (Maj) returns the alternative in the domain having more than $n/2$ votes, and $*$ otherwise; the *relative majority rule* (RMaj) returns the plurality outcome in $\mathcal{D} \setminus \{*\}$, and if there is a tie it returns $*$. A voting rule r on the domain $\{0, 1, *\}^n$ satisfies *monotonicity* if for any profile X , if $r(X) = x$ with $x \in \{0, 1\}$ then $r(X_{+x}) = x$, where X_{+x} is obtained from X by having one voter switch from

either an initial vote of $1 - x$ to x or $*$, or from an initial vote of $*$ to x . Observe that both Maj and RMaj satisfy monotonicity. Due to this definition we can now show the following:¹²

Theorem 5 *Any monotonic rule r with $\text{UNRAVEL}(\#)$ for $\# \in \{\mathbf{U}, \mathbf{DU}\}$ satisfies cast-participation for LIQUID_* with domain $\mathcal{D} = \{0, 1, *\}$.*

Proof 14 *Without loss of generality, assume that for agent $a \in \mathcal{N}$ we have $B_a = (1)$. To falsify cast-participation, we need to find a profile \mathbf{B} with $r(\text{UNRAVEL}(\#)(\mathbf{B})) = 0$ or $*$, and a ballot B'_a such that $r(\text{UNRAVEL}(\#)(\mathbf{B}_{-a}, B'_a)) = 1$, for $\# \in \{\mathbf{U}, \mathbf{DU}\}$.*

First, observe that all voters $c \in I_^\#(a, \mathbf{B})$ vote for 1 in \mathbf{B} , since the language is restricted to single-agent delegations. Now, if $B'_a = 0$ or $*$ (or they delegate to some agent who is assigned these votes), then by monotonicity the result of \mathbf{B}' will keep being 0 or $*$. Moreover, all $c \notin I_*^\#(a, \mathbf{B})$ do not change their vote from \mathbf{B} to \mathbf{B}' , no matter if B'_a is a direct vote or a possibly ranked delegation. Therefore, the final votes of \mathbf{B}' can be obtained from those of \mathbf{B} by switching 1s to 0s or $*$ s. Thus, this contradicts the monotonicity assumption of rule r .*

Remark 4 *Theorem 5 does not hold for BOOL ballots. Consider the counterexample with agents $\mathcal{N} = \{a, b, c\}$ having ballots $B_a = (1)$, $B_b = ((\{a\}, \neg a) > 0)$ and $B_c = ((\{a\}, \neg a) > 0)$. Each of our greedy unravellings would return the outcome $\text{Maj}(1, 0, 0) = 0$. However, if $B'_a = 0$ then $\text{Maj}(0, 1, 1) = 1$. Thus, agent a strictly prefers to submit a ballot that is not a direct vote for their preferred alternative.*

We now focus on the incentive that a voter has to receive and accept delegations; namely, what has been introduced as the guru-participation property.

Definition 11 (Guru-participation) *A voting rule r and a resolute unravelling procedure \mathcal{U} satisfy guru-participation if and only if for all profiles \mathbf{B} and all agents $a \in \mathcal{N}$ such that $B_a = (x)$ with $x \in \mathcal{D} \setminus \{*\}$ we have that for any $b \in I_*^\#(\mathbf{B}, a)$*

$$r(\mathcal{U}(\mathbf{B})) \geq_a r(\mathcal{U}(\mathbf{B}_{-b}, (*)))$$

where \mathbf{B}_{-b} is \mathbf{B} without b 's ballot. For randomised procedures we require the inequality to hold for any possible outcome of \mathcal{U} .

¹²Note that Definition 10 slightly differs from the one given in previous work [19], and thus Theorem 5 does not hold for RU or DRU: a counterexample can be constructed exploiting the fact that an agent may prefer the outcome of one random iteration of the procedure to another.

All four greedy unravellings do not satisfy this property for the rule RMaj:

Theorem 6 *RMaj and UNRAVEL($\#$) for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ do not satisfy guru-participation for LIQUID $_*$ with domain $\mathcal{D} = \{0, 1, *\}$.*

Proof 15 *Consider a smart profile \mathbf{B} , as shown on the left hand-side of Table 4, and profile $\mathbf{B}' = (\mathbf{B}_{-b}, (*))$ obtained from \mathbf{B} by switching b 's vote to $B'_b = (*)$. The outcomes of the four procedures are shown in the right hand-side of Table 4.*

	B_x^1	B_x^2	B_x^3		$\#$	\mathbf{B}	\mathbf{B}'
a	1	-	-				
b	$(\{c\}, id)$	$(\{a\}, id)$	$*$		$\mathbf{U}/$	$X^1 = (1, 1, 0, 0, 1, 0)$	$X^2 = (1, *, *, *, 1, 0)$
c	$(\{d\}, id)$	$(\{f\}, id)$	$*$		\mathbf{DU}		
d	$(\{b\}, id)$	$(\{f\}, id)$	$*$		$\mathbf{RU}/$	$X^3 = (1, 1, 1, 1, 1, 0)$	$X^2 = (1, *, *, *, 1, 0)$
e	1	-	-		\mathbf{DRU}	$X^4 = (1, 0, 0, 0, 1, 0)$	
f	0	-	-			$X^5 = (1, 0, 0, 0, 1, 0)$	

Table 4: A profile \mathbf{B} (on the left) and the outcomes of UNRAVEL(\mathbf{U}), UNRAVEL(\mathbf{DU}), UNRAVEL(\mathbf{RU}) and UNRAVEL(\mathbf{DRU}) on the profiles \mathbf{B} and \mathbf{B}' (on the right), where $\mathbf{B}' = (\mathbf{B}_{-b}, (*))$ is obtained from \mathbf{B} by switching b 's vote to $B'_b = (*)$.

By applying UNRAVEL(\mathbf{U}) and UNRAVEL(\mathbf{DU}), agent a prefers the outcome of \mathbf{B}' to that of \mathbf{B} , since $\text{RMaj}(X^1) = *$ and $\text{RMaj}(X^2) = 1$. For UNRAVEL(\mathbf{RU}) and UNRAVEL(\mathbf{DRU}), the outcome on \mathbf{B}' is $\text{RMaj}(X^2) = 1$. However, the outcome on \mathbf{B} can be either $\text{RMaj}(X^4) = \text{RMaj}(X^5) = 0$ or $\text{RMaj}(X^3) = 1$. Hence, when the random choice of \mathbf{RU} or \mathbf{DRU} leads to X^4 or X^5 , agent a strictly prefers the outcome $\text{RMaj}(X^2)$ to the outcome $\text{RMaj}(X^4)$ and $\text{RMaj}(X^5)$. Therefore, the inequality does not hold for any outcome of the randomised procedures.

4.3 Pareto dominance and optimality

We now focus on comparing the outcomes of our unravelling procedures in terms of *Pareto dominance* and *Pareto optimality*. We show that none of our procedures always Pareto dominates another. However, we prove that all outcomes of MINSUM are Pareto optimal with respect to all outcomes with consistent certificates.

A certificate \mathbf{c} *Pareto dominates* another certificate \mathbf{c}' if for every $i \in \mathcal{N}$, we have that $\mathbf{c}_i \leq \mathbf{c}'_i$, and there exists a $j \in \mathcal{N}$ such that $\mathbf{c}_j < \mathbf{c}'_j$. We say that the unravelling procedure \mathcal{U} Pareto dominates another unravelling procedure \mathcal{U}' when for any valid profile \mathbf{B} , all (possible) certificates \mathbf{c} obtained from $\mathcal{U}(\mathbf{B})$ Pareto dominate all the (possible) certificates \mathbf{c} obtained from $\mathcal{U}'(\mathbf{B})$. Note that the possibility of multiple certificates arises not only for irresolute procedures but also from different executions of the random procedures $\text{UNRAVEL}(\mathbf{RU})$ and $\text{UNRAVEL}(\mathbf{DRU})$.

Example 7 Consider the example given in Table 3. The certificate of the outcome of $\text{UNRAVEL}(\mathbf{U})$ is $\mathbf{c}^{\mathbf{U}} = (3, 3, 3, 2)$, and that of the outcome of $\text{UNRAVEL}(\mathbf{DU})$ is $\mathbf{c}^{\mathbf{DU}} = (1, 3, 3, 2)$. Thus, $\text{UNRAVEL}(\mathbf{DU})$ Pareto dominates $\text{UNRAVEL}(\mathbf{U})$, since $\mathbf{c}^{\mathbf{DU}}$ Pareto dominates $\mathbf{c}^{\mathbf{U}}$, given that each entry of $\mathbf{c}^{\mathbf{DU}}$ is less than or equal to the corresponding entry in $\mathbf{c}^{\mathbf{U}}$. Moreover, since there is an outcome of $\text{UNRAVEL}(\mathbf{RU})$ with certificate $\mathbf{c}^{\mathbf{RU}} = (3, 1, 1, 2)$, neither $\mathbf{c}^{\mathbf{DU}}$ Pareto dominates $\mathbf{c}^{\mathbf{RU}}$ (as $\mathbf{c}_a^{\mathbf{DU}} < \mathbf{c}_a^{\mathbf{RU}}$ for the first agent a) nor vice-versa (as $\mathbf{c}_b^{\mathbf{DU}} > \mathbf{c}_b^{\mathbf{RU}}$ for the second agent b).

When comparing our greedy procedures, one might think that $\text{UNRAVEL}(\mathbf{DRU})$ should always be chosen, given that it has both direct vote priority and random voter selection. However, the following example shows a profile where $\text{UNRAVEL}(\mathbf{DU})$, $\text{UNRAVEL}(\mathbf{RU})$ and $\text{UNRAVEL}(\mathbf{DRU})$ do not Pareto dominate $\text{UNRAVEL}(\mathbf{U})$, and thus, they do not Pareto dominate $\text{UNRAVEL}(\mathbf{U})$ in general.

Example 8 Take agents $\mathcal{N} = \{a, b, c, d, e, f\}$, whose ballots are shown in Table 5. On this profile, $\text{UNRAVEL}(\mathbf{U})$ gives the outcome $X_{\mathbf{c}} = (1, 1, 1, 1, 1, 0)$, where $\mathbf{c} = (1, 1, 1, 1, 2, 2)$ and $\text{UNRAVEL}(\mathbf{DU})$ gives $X_{\mathbf{c}'} = (0, 0, 0, 1, 0, 0)$, with certificate $\mathbf{c}' = (3, 3, 3, 1, 1, 2)$. Thus, \mathbf{c} does not Pareto dominate \mathbf{c}' as agent e has that $\mathbf{c}_e < \mathbf{c}'_e$. It is also not the case that \mathbf{c}' Pareto dominates \mathbf{c} as for some agents, i.e., agent a , we have that $\mathbf{c}_a > \mathbf{c}'_a$. Thus $\text{UNRAVEL}(\mathbf{DU})$ does not Pareto dominate $\text{UNRAVEL}(\mathbf{U})$ or vice-versa.

Furthermore, a possible outcome of $\text{UNRAVEL}(\mathbf{RU})$ is $X_{\mathbf{c}''} = (0, 0, 0, 1, 0, 0)$ where $\mathbf{c}'' = (3, 1, 1, 1, 1, 2)$ —the random choices picking first f and then a . Again, \mathbf{c}'' does not Pareto dominate \mathbf{c} , as $\mathbf{c}_a > \mathbf{c}''_a$. Therefore, $\text{UNRAVEL}(\mathbf{RU})$ does not Pareto dominate $\text{UNRAVEL}(\mathbf{U})$, and as $X_{\mathbf{c}''}$ is also an outcome of $\text{UNRAVEL}(\mathbf{DRU})$, $\text{UNRAVEL}(\mathbf{DRU})$ does not Pareto dominate $\text{UNRAVEL}(\mathbf{U})$ as well.

Proposition 8 None of the four greedy unravelling procedures $\text{UNRAVEL}(\#)$ for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ Pareto dominates another greedy procedure.

	B_x^1	B_x^2	B_x^3
a	$(\{b, e\}, b \vee e)$	$(\{c, e\}, c \vee e)$	0
b	$(\{c, e\}, c \vee e)$	$(\{a, e\}, a \vee e)$	0
c	$(\{a, e\}, a \vee e)$	$(\{b, e\}, b \vee e)$	0
d	1	-	-
e	$(\{f\}, f)$	$(\{d\}, d)$	0
f	$(\{e\}, e)$	0	-

Table 5: A profile \mathbf{B} showing that $\text{UNRAVEL}(\mathbf{U})$ is not dominated in general by $\text{UNRAVEL}(\mathbf{DU})$, $\text{UNRAVEL}(\mathbf{RU})$ or $\text{UNRAVEL}(\mathbf{DRU})$.

Proof 16 *Example 8 shows that $\text{UNRAVEL}(\mathbf{U})$ is not Pareto dominated by $\text{UNRAVEL}(\#)$ for $\# \in \{\mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$. Then, in Table 3 the outcome of $\text{UNRAVEL}(\mathbf{U})$ is Pareto dominated by the outcomes of $\text{UNRAVEL}(\#)$ for $\# \in \{\mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ and therefore, $\text{UNRAVEL}(\mathbf{U})$ does not Pareto dominate the other greedy procedures.*

From Table 3, we can also conclude that $\text{UNRAVEL}(\mathbf{DRU})$ does not Pareto dominate $\text{UNRAVEL}(\mathbf{DU})$, and vice-versa. The outcome of $\text{UNRAVEL}(\mathbf{DRU})$ with certificate $\mathbf{c} = (2, 1, 3, 2)$ does not Pareto dominate the outcome of $\text{UNRAVEL}(\mathbf{DU})$, having certificate $\mathbf{c}' = (1, 3, 3, 2)$, as $\mathbf{c}_a > \mathbf{c}'_a$. For the other direction, as $\mathbf{c}'_b > \mathbf{c}_b$ $\text{UNRAVEL}(\mathbf{DU})$ does not always Pareto dominate $\text{UNRAVEL}(\mathbf{DRU})$. Since the outcome with $\mathbf{c} = (2, 1, 3, 2)$ is also possible for $\text{UNRAVEL}(\mathbf{RU})$, $\text{UNRAVEL}(\mathbf{RU})$ is not guaranteed to Pareto dominate $\text{UNRAVEL}(\mathbf{DU})$ and vice-versa.

Finally, $\text{UNRAVEL}(\mathbf{DRU})$ and $\text{UNRAVEL}(\mathbf{RU})$ do not Pareto dominate one another as the certificates of the former are a subset of the latter (Proposition 7).

For irresolute procedures, by checking whether for any profile an unravelling procedure always has an outcome whose certificate Pareto dominates the certificates of all the outcomes of another procedure, we find the following negative results:

- The certificates $\mathbf{c} = (4, 1, \dots, 1)$ of MINSUM and $\mathbf{c}' = (1, 2, \dots, 2)$ of MINMAX from Example 3 show that neither \mathbf{c} Pareto dominates \mathbf{c}' nor vice-versa. Therefore, neither MINMAX nor MINSUM dominates the other.
- From Table 3, we see that $\text{UNRAVEL}(\mathbf{U})$ does not Pareto dominate MINSUM or MINMAX in general. From Example 3 we see that MINSUM does not

Pareto dominate UNRAVEL(\mathbf{U}) in general.

Finally, we introduce the notion of *Pareto optimality*, which defines all those consistent certificates that are not Pareto dominated by any other consistent certificate.

Definition 12 *A certificate \mathbf{c} for \mathbf{B} is Pareto optimal for the consistent certificates $\mathcal{C}(\mathbf{B})$ if there exists no $\mathbf{c}' \in \mathcal{C}(\mathbf{B})$ with $\mathbf{c}' \neq \mathbf{c}$, such that \mathbf{c}' Pareto dominates \mathbf{c} .*

The following proposition corresponds to the well-known fact that maximising the average of a vector leads to a Pareto optimal vector, but not vice-versa.

Proposition 9 *The certificate \mathbf{c} for any outcome $X_{\mathbf{c}} \in \text{MINSUM}(\mathbf{B})$ is Pareto optimal for $\mathcal{C}(\mathbf{B})$, for any profile \mathbf{B} .*

Proof 17 *Take an arbitrary valid smart profile \mathbf{B} , and arbitrary $X_{\mathbf{c}} \in \text{MINSUM}(\mathbf{B})$. For the sake of a contradiction, assume that \mathbf{c} is not Pareto optimal for $\mathcal{C}(\mathbf{B})$. Hence, there exists a $\mathbf{c}' \in \mathcal{C}(\mathbf{B}) \setminus \{\mathbf{c}\}$ such that \mathbf{c}' Pareto dominates \mathbf{c} . Therefore, for all $i \in \mathcal{N}$, we get $\mathbf{c}'_i \leq \mathbf{c}_i$. We thus have that $\sum_{i \in \mathcal{N}} \mathbf{c}'_i \leq \sum_{i \in \mathcal{N}} \mathbf{c}_i$. Furthermore, since $\mathbf{c} \neq \mathbf{c}'$ and \mathbf{c}' Pareto dominates \mathbf{c} , there exists an agent $j \in \mathcal{N}$ such that $\mathbf{c}'_j < \mathbf{c}_j$, and thus $\sum_{i \in \mathcal{N}} \mathbf{c}'_i < \sum_{i \in \mathcal{N}} \mathbf{c}_i$. Since $\sum_{i \in \mathcal{N}} \mathbf{c}_i$ is not minimal, we have $X_{\mathbf{c}} \notin \text{MINSUM}(\mathbf{B})$, and thus we have reached a contradiction.*

Note that the opposite direction does not hold: in Example 3, the MINSUM procedure does not return the Pareto optimal certificate $\mathbf{c} = (1, 2, \dots, 2)$. Moreover, Proposition 9 does not hold for the other unravelling procedures, as there exist outcomes of each of them whose certificates are Pareto dominated by some other consistent certificate, as seen in previous examples.

4.4 Discussion on unravelling procedures

In this paper we have provided six unravelling procedures and we have given results that should guide a user of this model as to which procedure to choose. Here we provide a summary and a discussion of these results.

The main distinction between the optimal and greedy procedures is that finding an outcome with a greedy procedure is a tractable problem, whereas even checking if an outcome of an optimal procedure exists under a given bound on the optimised score is an NP-complete problem for the general language BOOL (where

the delegations are contingent formulas expressed in complete DNF). Although we acknowledge that the improvements in performance of SAT-solvers make the intractability of the problems BOUNDEDMINMAX and BOUNDEDMINSUM less concerning, the associated search problem of computing the outcomes of the unravelling remains in principle even harder. Hence, the greedy procedures are desirable when tractability is key.

Proposition 9 shows that the certificates of the outcomes of MINSUM are Pareto optimal and thus are never dominated by outcomes found by a consistent certificate. In contrast, MINMAX cannot make this guarantee. Although MINMAX may return outcomes that are not Pareto optimal, it provides more egalitarian outcomes. In Example 3, the outcome with the lowest `rank` relies on the fourth preference of agent a being chosen: while it is still a trusted delegate, the agent may be less confident in them than in the three previous delegations.

Furthermore, as MINSUM and MINMAX are irresolute, they would have to be paired with a tie-breaking mechanism to select a single outcome from the possibly many that they produce. In contrast, the greedy procedures are not only, in general, quicker than the optimal procedures, but they are also resolute.

With profiles of LIQUID[1]_{*} ballots, MINSUM and the greedy procedures return the same outcome vector, so they can be used interchangeably. For profiles of LIQUID ballots, there should be a preference for MINSUM or MINMAX, since an outcome can be found in polynomial time (Theorems 2 and 4). The choice between these two procedures should then be determined by whether the situation would benefit more from Pareto optimality or egalitarian properties. However, these procedures do rely on tie-breaking, which could bring up issues of fairness in the certificates.

As the participation axioms do not differentiate the greedy procedures, the properties that they are defined on (i.e., direct vote priority and random voter selection) are the clearest way to compare them. Random voter selection should be used when a lottery is acceptable and it should be avoided when it would be unfair to give a worse preference level to just some agents. Direct vote priority should be used when a direct vote from an agent is preferred to a delegation, perhaps in situations that could benefit from a level of expertise on the issue, and to ensure shorter delegation chains.

Given the above discussion, one may think that UNRAVEL(DRU) gives the best outcomes overall. However, we have proved that no greedy procedure is guaranteed to Pareto dominate another (Proposition 8). Thus, the notion of Pareto dominance does not distinguish between greedy procedures.

In summary, greedy procedures should be preferred when finding outcomes

needs to be tractable, except in the special case of LIQUID ballots for which this problem is polynomial for all proposed rules. The MINSUM procedure should be used when outcomes need to be Pareto optimal and MINMAX should be used when an egalitarian approach is required. When using the greedy procedures, the choice between them should be determined by whether the situation asks for either of the random voter selection and direct vote priority properties.

5 Conclusion

We proposed a model of multi-agent ranked delegations in voting, which generalises liquid democracy in two aspects. The first is that delegations are more expressive, as they can involve many agents instead of a single one, who in turn determine their vote. We introduced a general language named BOOL, in which delegations are expressed as contingent propositional formulas in complete DNF. We emphasise that although agents may not want to use the full expressivity of the language, many natural delegations types are captured by it: for example, both liquid democracy delegations and delegations using threshold rules can be expressed in BOOL. Our second generalisation is the possibility of ranked delegations: as transitive delegations can lead to cycles among the agents' most preferred delegates, the linear order of trusted delegations given by the agents can be used to break these cycles.

Our main contribution is the definition and study of six unravelling procedures (two optimal and four greedy ones), which take a profile of smart ballots and return a standard voting profile. We show that all of the procedures can give different certificates and outcomes (Proposition 5), and that they differ from the breadth-first and depth-first procedures of Kotsialou and Riley [38] (Remark 3). Moreover, we show that all of the procedures, except MINMAX, coincide on LIQUID[1] ballots, i.e., liquid democracy ballots with a single delegation per agent. The certificates of the outcomes of MINSUM are Pareto optimal with respect to the outcomes found from any consistent certificate (Proposition 9), while greedy procedures do not Pareto dominate one another (Proposition 8). Our main results are that deciding if there exists an outcome of the unravellings MINSUM and MINMAX bounded by some constant are NP-complete problems (Theorems 1 and 3) over the general language BOOL, but they become tractable when ballots are restricted to LIQUID, the language of ranked liquid democracy (Theorems 2 and 4). Finally, we prove that our four greedy unravelling procedures always terminate (Proposition 3) and do so in a polynomial number of time steps for general BOOL ballots

(Proposition 4).

Future work. This paper provides a first analysis of six unravelling procedures in terms of their computational (and some axiomatic) properties. A game-theoretic analysis of our procedures is yet to be undertaken, by focusing, e.g., on various notions of manipulative actions. Moreover, we have shown that finding a solution for our optimal unravellings is a tractable problem when ballots are restricted to ranked single agent delegations. Other tractable cases could be found, e.g., by restricting delegation functions or by limiting the number of delegations—leading to a study of the parameterised complexity of optimal unravellings. Finally, we focused only on independent issues: extending our model to account for the agents’ rationality with respect to interconnected issues (in line with the work of Christoff and Grossi [15] and, to a lesser extent, Brill and Talmon [11]) would be a natural avenue of future research.

Acknowledgments

The authors acknowledge the support of the ANR JCJC project SCONE (ANR 18-CE23-0009-01).

References

- [1] Ben Abramowitz and Nicholas Mattei. “Flexible Representative Democracy: An Introduction with Binary Issues”. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. 2019.
- [2] Dan Alger. “Voting by proxy”. In: *Public Choice* 126.1-2 (2006), pp. 1–26.
- [3] Jan Behrens and Björn Swierczek. “Preferential Delegation and the Problem of Negative Voting Weight”. In: *The Liquid Democracy Journal* 3 (2015).
- [4] Jan Behrens et al. *Principles of Liquid Feedback*. Interaktive Demokratie, 2014.
- [5] Daan Bloembergen, Davide Grossi, and Martin Lackner. “On Rational Delegations in Liquid Democracy”. In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. 2019.

- [6] Christian Blum and Christina Isabel Zuber. “Liquid democracy: Potentials, problems, and perspectives”. In: *Journal of Political Philosophy* 24.2 (2016), pp. 162–182.
- [7] F. C. Bock. “An algorithm to construct a minimum directed spanning tree in a directed network”. In: *Developments in operations research* (1971), pp. 29–44.
- [8] Paolo Boldi et al. “Viscous democracy for social networks”. In: *Communications of the ACM* 54.6 (2011), pp. 129–137.
- [9] Robert Brederick and Edith Elkind. “Manipulating Opinion Diffusion in Social Networks”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. 2017.
- [10] Markus Brill. “Interactive democracy”. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018.
- [11] Markus Brill and Nimrod Talmon. “Pairwise Liquid Democracy”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2018.
- [12] Markus Brill et al. *Liquid Democracy with Ranked Delegations*. Presentation at the Workshop on Liquid Democracy at the University of Toulouse. 2021. URL: www.irit.fr/~Umberto.Grandi/scone/WK_Ulrike.pdf.
- [13] Markus Brill et al. “Pairwise Diffusion of Preference Rankings in Social Networks”. In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. 2016.
- [14] Ioannis Caragiannis and Evi Micha. “A contribution to the critique of liquid democracy”. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. 2019.
- [15] Zoé Christoff and Davide Grossi. “Binary Voting with Delegable Proxy: An Analysis of Liquid Democracy”. In: *Proceedings of the 16th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*. 2017.
- [16] Yoeng-Jin Chu. “On the shortest arborescence of a directed graph”. In: *Scientia Sinica* 14 (1965), pp. 1396–1400.
- [17] Gal Cohensius and Reshef Meir. “Proxy Voting for Revealing Ground Truth”. In: *Proceedings of the 4th Workshop on Exploring Beyond the Worst Case in Computational Social Choice (EXPLORE)*. 2017.

- [18] Gal Cohensius et al. “Proxy Voting for Better Outcomes”. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2017.
- [19] Rachael Colley, Umberto Grandi, and Arianna Novaro. “Smart Voting”. In: *Proceeding of the the 29th International Joint Conference on Artificial Intelligence (IJCAI)*. 2020.
- [20] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.
- [21] Yves Crama and Peter L Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.
- [22] Jonas Degraeve. “Resolving multi-proxy transitive vote delegation”. In: *arXiv preprint arXiv:1412.4039* (2014).
- [23] Amrita Dhillon et al. *Introduction to Voting and the Blockchain: some open questions for economists*. Tech. rep. Competitive Advantage in the Global Economy (CAGE), 2019.
- [24] Charles Lutwidge Dodgson. *The Principles of Parliamentary Representation*. Harrison and Sons, 1884.
- [25] Jack Edmonds. “Optimum branchings”. In: *Journal of Research of the national Bureau of Standards B* 71.4 (1967), pp. 233–240.
- [26] Bruno Escoffier, Hugo Gilbert, and Adèle Pass-Lanneau. “Iterative delegations in liquid democracy with restricted preferences”. In: *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*. 2020.
- [27] Bruno Escoffier, Hugo Gilbert, and Adèle Pass-Lanneau. “The Convergence of Iterative Delegations in Liquid Democracy in a Social Network”. In: *Proceedings of the 12th International Symposium on Algorithmic Game Theory (SAGT)*. 2019.
- [28] Bryan Alexander Ford. *Delegative democracy*. Tech. rep. 2002.
- [29] Paul Gözl et al. “The fluid mechanics of liquid democracy”. In: *International Conference on Web and Internet Economics (ICWIE)*. 2018.
- [30] Umberto Grandi. “Social choice and social networks”. In: *Trends in Computational Social Choice* (2017), pp. 169–184.
- [31] Mark Granovetter. “Threshold models of collective behavior”. In: *American journal of sociology* 83.6 (1978), pp. 1420–1443.

- [32] James Green-Armytage. “Direct voting and proxy voting”. In: *Constitutional Political Economy* 26.2 (2015), pp. 190–220.
- [33] Steve Hardt and Lia C. R. Lopes. *Google Votes: A Liquid Democracy Experiment on a Corporate Social Network*. Tech. rep. Technical Disclosure Commons, Google, 2015.
- [34] Anson Kahng, Simon Mackenzie, and Ariel D Procaccia. “Liquid democracy: An algorithmic perspective”. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*. 2018.
- [35] Richard M Karp. “Reducibility among combinatorial problems”. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.
- [36] Christoph Kling et al. “Voting behaviour and power in online democracy: A study of LiquidFeedback in Germany’s Pirate Party”. In: *Proceedings of the International AAAI Conference on Web and Social Media*. 2015.
- [37] Kathrin Konczak and Jérôme Lang. “Voting procedures with incomplete preferences”. In: *Proceedings of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling (MPREF)*. 2005.
- [38] Grammateia Kotsialou and Luke Riley. “Incentivising Participation in Liquid Democracy with Breadth-First Delegation”. In: *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2020.
- [39] Dexter C Kozen. *The design and analysis of algorithms*. Springer Science & Business Media, 2012.
- [40] Rob Lanphier. “A model for electronic democracy”. In: *Manuscript*. <http://robla.net/1996/steward> (1995).
- [41] Anna Litvinenko. “Social media and perspectives of liquid democracy on the example of political communication of Pirate Party in Germany”. In: *Proceedings of the 12th European Conference on e-Government in Barcelona*. 2012.
- [42] Pia Mancini. “Why it is time to redesign our political system”. In: *European View* 14.1 (2015), pp. 69–75.
- [43] Reshef Meir et al. “Sybil-Resilient Social Choice with Partial Participation”. In: *arXiv preprint arXiv:2001.05271* (2020).
- [44] James C Miller. “A program for direct and proxy voting in the legislative process”. In: *Public choice* 7.1 (1969), pp. 107–113.

- [45] Hervé Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1988.
- [46] Dennis C Mueller, Robert D Tollison, and Thomas D Willett. “Representative democracy via random selection”. In: *Public Choice* 12.1 (1972), pp. 57–68.
- [47] Ehud Shapiro. “Point: foundations of e-democracy”. In: *Communications of the ACM* 61.8 (2018), pp. 31–34.
- [48] Björn Swierczek. “Five years of Liquid Democracy in Germany”. In: *The Liquid Democracy Journal* 1.1 (2014), pp. 8–19.
- [49] Gordon Tullock. “Computerizing politics”. In: *Mathematical and Computer Modelling* 16.8-9 (1992), pp. 59–65.
- [50] Gordon Tullock. *Toward a mathematics of politics*. Ann Arbor: University of Michigan Press, 1967.
- [51] Bingsheng Zhang and Hong-Sheng Zhou. “Statement voting”. In: *International Conference on Financial Cryptography and Data Security*. 2019.
- [52] Yuzhe Zhang and Davide Grossi. “Power in Liquid Democracy”. In: *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*. 2021.