Smart Voting

Rachael Colley, **Umberto Grandi** and **Arianna Novaro**
IRIT, University of Toulouse
{rachael.colley, umberto.grandi, arianna.novaro}@irit.fr

Abstract

We propose a generalisation of liquid democracy in which a voter can either vote directly on the issues at stake, delegate her vote to another voter, or express complex delegations to a set of trusted voters. By requiring a ranking of desirable delegations and a backup vote from each voter, we are able to put forward and compare four algorithms to solve delegation cycles and obtain a final collective decision.

1 Introduction

Advances in information technology are opening up avenues for the improvement of democratic practices and collective decision-making processes (see, e.g., Brill [2018]). One of the most popular ideas is to explore the space between direct democracy—often infeasible due to the limited amount of resources decision-makers can invest in each direct vote—and representative democracy, by allowing for transitive delegations to take place among agents. This method is named *liquid democracy*, in contrast to *proxy voting* in which delegations are not transitive [Miller, 1969]. Liquid democracy has been the subject of numerous recent investigations in AI, ranging from analysing its truth-tracking power [Cohensius *et al.*, 2017; Bloembergen *et al.*, 2019; Kahng *et al.*, 2018] to its adaptation for multiple issues and alternatives [Christoff and Grossi, 2017; Brill and Talmon, 2018]. In its simplest form, liquid democracy works as follows: a collective decision needs to be taken on a binary issue; agents can either vote directly or delegate their vote to a single other agent; direct votes are then weighted by the number of transitive delegations received, and a final decision is taken using a standard voting rule.

In this paper we propose to push forward the idea of liquid democracy under two aspects. First, while most implementations of liquid democracy [see, e.g., Behrens *et al.*, 2014] introduce a platform for the elicitation of voters' delegations, we aim at a decentralised voting system in which voters' ballots and delegations can remain private. Thus, we define a language for voters to express a direct vote, or a delegation to a single other agent, or a combination of the votes of multiple other agents. Second, to tackle the issue of delegation cycles we allow voters to express a number of prioritized delegations, with a final backup vote with the lowest priority. An example of what we call a *smart ballot* can be seen on the right-hand side of Figure 1.

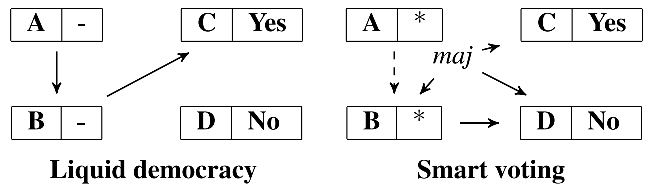

Figure 1: A profile of binary votes in liquid democracy on the left: voter A delegates her vote to B, who in turn delegates to C, who casts a direct vote in favour, unlike D who casts a direct vote against. On the right, a profile of smart ballots: voter A wants her vote to coincide with the majority of B, C, and D's votes, and in case this leads to a delegation cycle she gives a single delegation to B. Voter B delegates to D, who casts a direct vote against, while C votes in favour. Voters A and B abstain (*) as their final backup option.

Two possible practical implementations can be envisaged for smart voting. The first is a fully decentralised version that might make use of *smart contract* technology (hence the title of this paper), where a smart ballot would be a piece of self-executable and publicly available code included in a blockchain, with full transparency and accountability of the election procedure.[1]The second, in applications where vote secrecy is crucial, is a more standard method of votes being collected by a trusted central authority—raising interesting computational questions on providing a certificate to each voter that their ballot has been taken into account.

Our contribution. We define a script language for voters to express ranked, possibly complex, delegations in collective decisions with multiple issues (Section 2). The intended application is voting in a purely preferential setting (deciding on food is a perfect example [see, e.g., Hardt and Lopes, 2015]), and delegations are seen as a way to elicit trust or influence relations among voters. To transform profiles of smart ballots into direct votes for alternatives we propose four *unravelling procedures* (Section 2), and we show that they terminate in polynomial time (Section 3). The final objective of smart voting is that of obtaining a collective decision from the agents'

[1]Recent work by Dhillon *et al.* [2019] conducts a detailed analysis of smart contracts for electronic voting and liquid democracy. See also the reports of Kotsialou *et al.* [2018] and Riley *et al.* [2019].

ballots, and we assume here that a standard voting rule is applied on the outcome of the proposed unravelling procedures. We investigate further algorithmic properties of our setting in Section 3, and we conclude with a study of ranked delegations to single voters and participation axioms (Section 4). Some proofs are sketched or omitted due to space constraints.

Related work. Recent papers have analysed the efficacy of liquid democracy and proxy voting as a truth-tracking mechanism, mostly for a binary decision [Green-Armytage, 2015; Cohensius *et al.*, 2017; Kahng *et al.*, 2018; Caragiannis and Micha, 2019; Bloembergen *et al.*, 2019]. We do not study truth-tracking here, and rather propose a method to elicit mutual influence from voters and infer a collective decision.

This work takes inspiration from papers trying to solve the issue of delegation cycles. As in the work of Kotsialou and Riley [2020] and Escoffier *et al.* [2019; 2020], we let voters express a ranking of possible delegates, to break potential cycles of delegations.[2] We also draw inspiration from Degrave [2014] and Abramowitz and Mattei [2019] in allowing a delegation to be spread to multiple voters. However, we keep the transitivity of delegations and stick to the principle of "one person, one vote" in not splitting the voting power of any individual. We also borrow from Christoff and Grossi [2017] the requirement that voters must specify a *default value* (i.e., a backup vote) on each issue.

Generalisations of liquid democracy have been studied by Christoff and Grossi [2017] on multiple logically connected issues, and by Brill and Talmon [2018], who consider delegations on pairwise preferences on alternatives. We assume here votes on multiple independent issues, and we also do not consider aspects of strategic voting or analysis of voters power, as done by Escoffier *et al.* [2019] and Gölz *et al.* [2018].

In allowing voters to express complex delegations combining the votes of other agents, we follow the vast literature on social influence and opinion diffusion on networks [see, e.g., Easley and Kleinberg, 2010] and, most notably, research studying the use of aggregation functions such as majority in binary opinion diffusion [Grandi *et al.*, 2015; Bredereck and Elkind, 2017; Auletta *et al.*, 2018].

2 Formal Framework

We define here the components of our model for smart voting.

2.1 Smart Ballots

In smart voting we have a set $\mathcal{N}$ of n *agents* (or *voters*) who decide on a set $\mathcal{I}$ of m issues. The alternative values, or simply *alternatives*, for each issue $i \in \mathcal{I}$ range over a non-empty finite *domain* $D(i)$, which can also include abstention.

An agent a expresses her opinion over an issue $i \in \mathcal{I}$ by submitting a (valid) smart ballot B_{ai} defined as follows:

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

Thus, a smart ballot can be seen as a preference ordering over the agent's desired delegations, ending with a direct backup vote for an alternative in the issue's domain.

Definition 2 (Valid Smart Ballot)**.** *A* valid smart ballot *of agent a for an issue $i \in \mathcal{I}$ is a smart ballot such that, for all $1 \leq s \neq t \leq k$, (i) if $S^s \cap S^t \neq \emptyset$ then F^s is not equivalent to F^t, and (ii) $a \notin S^s$.*

The two validity requirements ensure that agents are not manipulating the election by submitting many equivalent versions of the same delegation function, and that they are not generating delegation loops by including themselves in the set of delegates. The following example shows the expressiveness of smart ballots:

Example 1. *Alice, Bob, Carl, and Diana, wonder whether to try a new restaurant (1) or stay in (0). Alice knows that her friends are real foodies, and she is too tired to check online reviews. She would like to state this complex delegation: "I vote to go if and only if Bob, and at least one of Carl or Diana, think it's worth it.*[3] *If that creates a delegation cycle, I will copy Bob's vote. If that also creates a cycle, I will vote to go". She submits the following smart ballot:*

- $((\{B, C, D\}, B \wedge (C \vee D)) > (\{B\}, B) > 1)$.

The next ballot represents Alice wanting to delegate to Bob, then Carl, then Diana, and as a last resort voting to go:

- $((\{B\}, B) > (\{C\}, C) > (\{D\}, D) > 1)$.

This last ballot expresses Alice wanting to vote with the majority opinion of her friends, and otherwise voting to go:

- $((\{B, C, D\}, \mathit{Maj}) > 1)$.

Let B^h_{ai} denote the h^{th} *preference level* given by agent a on issue i in her smart ballot B_{ai}—thus, $B^h_{ai} = (S^h_{ai}, F^h_{ai})$ or $B^h_{ai} = x$ with $x \in D(i)$. For instance, in Example 1 we have in the second case that $B^2_{Ai} = (S^2_{Ai}, F^2_{Ai}) = (\{C\}, C)$.

Given n smart ballots for an issue $i \in \mathcal{I}$ from each agent in $\mathcal{N}$, a *(smart) profile* for this issue i, is a vector $\boldsymbol{B}_i = (B_{1i}, \ldots, B_{ni})$. A *valid (smart) profile* is a smart profile where each smart ballot is valid (as per Definition 2).

2.2 Unravelling Procedures

In this section we propose four procedures that transform a valid smart profile into a profile of direct votes, i.e., votes supporting a single alternative in the issue's domain, $D(i)$.

Definition 3 (Unravelling Procedure)**.** *An* unravelling procedure *$\mathcal{U}$ for issue $i \in \mathcal{I}$ and agents in $\mathcal{N}$ is any function*

$$\mathcal{U} : (B_{1i} \times \cdots \times B_{ni}) \to D(i)^n.$$

Due to possible delegation cycles, it may be unclear how to assign direct votes to agents. We follow two principles in our definitions: use the highest *preference level* of voters when breaking delegation cycles (cf. Definition 4), and keep the unravelling process polynomial (cf. Section 3). Algorithm 1 outlines our *general unravelling procedure* UNRAVEL.

The input of UNRAVEL is a smart profile $\boldsymbol{B}_i$. A vector X is initialised with placeholders Δ at each position x_a for $a \in \mathcal{N}$

[2] We note in passing that ranked delegations for collective decisions were also experimented by Google [Hardt and Lopes, 2015].

[3] This can be expressed by propositional formula $B \wedge (C \vee D)$.

Algorithm 1 General unravelling procedure UNRAVEL

1: $X := (\Delta, \dots, \Delta)$ ▷ *empty vector for direct votes*
2: **while** $X \notin D(i)^n$ **do**
3: $\quad$ $\texttt{lev} := 1$ ▷ *reset preference level counter to 1*
4: $\quad$ $Y := X$ ▷ *a copy of X to compute changes*
5: $\quad$ **while** $X = Y$ **do**
6: $\quad\quad$ **procedure** UPDATE($\{\mathbf{U}, \mathbf{RU}, \mathbf{DU}, \mathbf{DRU}\}$)
7: $\quad\quad\quad$ $\texttt{lev} := \texttt{lev} + 1$
8: **return** X ▷ *output vector of direct votes*

and it is returned when a vote in $D(i)$ is found for all agents. Counter lev is set at the first preference level of the agents and an additional vector Y is added to help computation. In line 6 a subroutine with one *update procedure* is executed.[4]

We give four update procedures, defined by the presence or absence of two properties. Namely, *direct vote priority* (D), prioritising *direct votes* over those that can be computed from the current vector Y of votes; and *random voter selection* (R), *randomly* choosing an agent whose vote is added (or computed) next. The four procedures are thus: basic update ($\mathbf{U}$), update with direct vote priority ($\mathbf{DU}$), update with random voter selection ($\mathbf{RU}$), update with direct vote priority and random voter selection ($\mathbf{DRU}$).

Unless otherwise specified, if the condition in an **if** statement fails, the program skips to the next step. Moreover, $Y_{\restriction S}$ denotes the restriction of vector Y to the elements in set S.

Algorithm 2 UPDATE($\mathbf{U}$)

1: **for** $a \in \mathcal{N}$ such that $x_a = \Delta$ **do**
2: $\quad$ **if** $B_{ai}^{\texttt{lev}} \in D(i)$ **then** ▷ *a has a direct vote at* lev
3: $\quad\quad$ $x_a := B_{ai}^{\texttt{lev}}$
4: $\quad$ **else if** $F_{ai}^{\texttt{lev}}(Y_{\restriction S_{ai}^{\texttt{lev}}}) \in D(i)$ **then**
5: $\quad\quad$ $x_a := F_{ai}^{\texttt{lev}}(Y_{\restriction S_{ai}^{\texttt{lev}}})$ ▷ *add a's computable vote*

UPDATE($\mathbf{U}$) checks for each agent without a direct vote at lev (line 1): if their preference at lev is either a direct vote (line 3) or can be computed from the current values in Y (line 5), then vector X is updated with the new direct votes.

Algorithm 3 UPDATE($\mathbf{DU}$)

1: **for** $a \in \mathcal{N}$ such that $x_a = \Delta$ **do**
2: $\quad$ **if** $B_{ai}^{\texttt{lev}} \in D(i)$ **then** ▷ *add direct votes*
3: $\quad\quad$ $x_a := B_{ai}^{\texttt{lev}}$
4: **if** $Y = X$ **then** ▷ *if no direct vote added to X*
5: $\quad$ **for** $a \in \mathcal{N}$ such that $x_a = \Delta$ **do**
6: $\quad\quad$ **if** $F_{ai}^{\texttt{lev}}(Y_{\restriction S_{ai}^{\texttt{lev}}}) \in D(i)$ **then** ▷ *find computables*
7: $\quad\quad\quad$ $x_a := F_{ai}^{\texttt{lev}}(Y_{\restriction S_{ai}^{\texttt{lev}}})$

UPDATE($\mathbf{DU}$) first tries to add directs votes to X (line 2); if there are none (line 4) it tries with votes computable at lev.

At line 1 of UPDATE($\mathbf{RU}$) an empty set P is initialised to hold agents with either a direct vote or a computable vote at lev (line 3); if P is non-empty, one agent will be randomly chosen and her direct/computable vote is added to X.

[4]In what follows, we simply write UNRAVEL(#) to indicate the UNRAVEL algorithm using UPDATE procedure #.

Algorithm 4 UPDATE($\mathbf{RU}$)

1: $P := \emptyset$ ▷ *initialise an empty set*
2: **for** $a \in \mathcal{N}$ such that $x_a = \Delta$ **do**
3: $\quad$ **if** $B_{ai}^{\texttt{lev}} \in D(i)$ or $F_{ai}^{\texttt{lev}}(Y_{\restriction S_{ai}^{\texttt{lev}}}) \in D(i)$ **then**
4: $\quad\quad$ $P := P \cup \{a\}$
5: **if** $P \neq \emptyset$ **then** ▷ *there are direct/computable votes*
6: $\quad$ **select** b from P uniformly at random
7: $\quad$ **if** $B_{bi}^{\texttt{lev}} \in D(i)$ **then**
8: $\quad\quad$ $x_b := B_{bi}^{\texttt{lev}}$
9: $\quad$ **else if** $F_{bi}^{\texttt{lev}}(Y_{\restriction S_{bi}^{\texttt{lev}}}) \in D(i)$ **then**
10: $\quad\quad$ $x_b := F_{bi}^{\texttt{lev}}(Y_{\restriction S_{bi}^{\texttt{lev}}})$

Algorithm 5 UPDATE($\mathbf{DRU}$)

1: $P, P' := \emptyset$ ▷ *initialise an empty set*
2: **for** $a \in \mathcal{N}$ such that $x_a = \Delta$ **do**
3: $\quad$ **if** $B_{ai}^{\texttt{lev}} \in D(i)$ **then** ▷ *add agents with direct vote to P*
4: $\quad\quad$ $P := P \cup \{a\}$
5: $\quad$ **else if** $F_{ai}^{\texttt{lev}}(Y \restriction_{S_{ai}^{\texttt{lev}}}) \in D(i)$ **then**
6: $\quad\quad$ $P' := P' \cup \{a\}$
7: **if** $P \neq \emptyset$ **then** ▷ *there are agents with direct votes*
8: $\quad$ **select** b from P uniformly at random
9: $\quad$ $x_b := B_{bi}^{\texttt{lev}}$
10: **else if** $P' \neq \emptyset$ **then** ▷ *there are computable votes*
11: $\quad$ **select** b from P' uniformly at random
12: $\quad$ $x_b := F_{bi}^{\texttt{lev}}(Y \restriction_{S_{bi}^{\texttt{lev}}})$

UPDATE($\mathbf{DRU}$) first selects agents with a direct vote at level lev (line 3) and chooses one randomly to update X (line 9). If not, UPDATE($\mathbf{DRU}$) will repeat this process, but it will now look for computable votes at level lev.

The following example shows how UPDATE($\mathbf{U}$) works:

Example 2. *Consider a binary issue i with $D(i) = \{0, 1\}$ and agents $\mathcal{N} = \{A, \dots, E\}$. Their valid ballots in $\boldsymbol{B}_i$, stating their preferences for delegations or direct votes, are shown schematically in the table below and in Figure 2.*

	1st	2nd	3rd
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

We spell out here the unravelling UNRAVEL*(*$\mathbf{U}$*) on* $\boldsymbol{B}_i$*:*

- *Starting at $\texttt{lev} = 1$, the direct vote of B is stored in $X = (\Delta, 1, \Delta, \Delta, \Delta)$.*
- *As there are no direct or computable votes at $\texttt{lev} = 1$ (using Y), the level counter is increased to $\texttt{lev} = 2$.*
- *The procedure adds the direct votes of C and D, and computes the vote of E from the current Y by copying the vote of B, obtaining $X = (\Delta, 1, 0, 1, 1)$.*

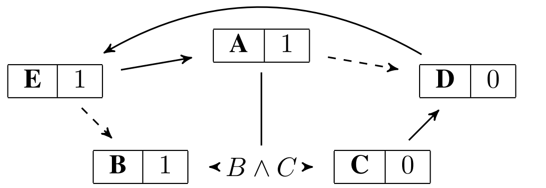

Figure 2: A network representation of $\boldsymbol{B}_i$ from Example 2. Solid lines represent first preferences and dashed lines second preferences. The final preference for a direct vote is next to the agent's name.

- *The preference level is set back to* $\texttt{lev} = 1$.
- *A's vote can be computed from her first preference level, and vector* $X = (0, 1, 0, 1, 1)$ *is thus returned.*

An important clarification is that in all our procedures, when checking if an agent's vote can be computed from a partial profile of other agents' votes, we test the existence of a *necessary winner* [Konczak and Lang, 2005]. Formally, $F^{\texttt{lev}}_{ai}(X_{\restriction S^{\texttt{lev}}_{ai}})$ has a necessary winner $y \in D(i)$ if for all $a \in S^{\texttt{lev}}_{ai}$ such that $x_a = \Delta$, for all $z \in D(i)$ such that $x_a = z$ we have that $F^{\texttt{lev}}_{ai}(X_{\restriction S^{\texttt{lev}}_{ai}}) = y$.[5]

We conclude with two further definitions. The notion of vote *certificate* can explain to an agent which part of their ballot was used to compute the unravelled profile, without revealing the full profile:

Definition 4 (Certificates)**.** *An agent a's* certificate *of an unravelled profile $\mathcal{U}(\boldsymbol{B}_i)$ is $Cert^{\mathcal{U}}(\boldsymbol{B}_i, a) = (k, X_{\restriction S^k_{ai}})$, where k is a preference level and $X_{\restriction S^k_{ai}}$ is the partial profile of direct votes used by $\mathcal{U}$ to compute a's vote.*

We can then define the set of voters influenced by a voter a:

Definition 5 (Influence)**.** *The set of voters* influenced *by voter a in profile $\boldsymbol{B}_i$ using unravelling procedure $\mathcal{U}$ is $I^{\mathcal{U}}(\boldsymbol{B}_i, a) = \{b \mid a \in S^k_{bi}$ for $Cert^{\mathcal{U}}(\boldsymbol{B}_i, b) = (k, X_{\restriction S^k_{bi}})\}$.*

Let $I^{\mathcal{U}}_*(\boldsymbol{B}_i, a) = I^{\mathcal{U}}(\boldsymbol{B}_i, a) \cup \{c | c \in I^{\mathcal{U}}(\boldsymbol{B}_i, b) \wedge b \in I^{\mathcal{U}}(\boldsymbol{B}_i, a)\} \cup \ldots$ be the voters b *indirectly* influenced by a. In Example 2, we have that $\text{Cert}^{\mathbf{U}}(\boldsymbol{B}_i, C) = (2, \emptyset)$ and $I^{\mathbf{U}}(\boldsymbol{B}_i, C) = \{A\}$.

2.3 Restricting the Language of Smart Ballots

In this section, we define useful restrictions to the language of smart ballots. Firstly, we focus on binary Boolean functions compactly expressed as *contingent* (i.e., neither a contradiction, nor a tautology) propositional formulas in DNF:

Definition 6 (BOOL Ballots)**.** *A smart ballot B_{ai} for agent a and binary issue i belongs to language* BOOL *if every F^h_{ai} in B_{ai} is a contingent propositional formula in DNF.*

Therefore, BOOL ballots can only contain Boolean functions—as, for instance, all the delegating ballots in Example 2. Atomic boolean functions are equivalent to the identity function, i.e., copying another agent's vote. The requirement for a contingent formula is to prevent agents from overriding the delegation structure by giving a direct vote for 1 or 0 in disguise at multiple preference levels. When using language BOOL, we will write $\varphi^{\texttt{lev}}_{ai}$ instead of $F^{\texttt{lev}}_{ai}$.

Next, we define a restriction for smart ballots where all of the (possibly many) delegations must be to single agents:

Definition 7 (LIQUID Ballots)**.** *Smart ballot B_{ai} for agent a and issue i belongs to* LIQUID *if every delegating B^h_{ai} in B_{ai} is of the form $(\{b\}, id)$ for $b \in \mathcal{N} \setminus \{a\}$ and id the identity function, and if $h > 1$ the final vote of a is abstention $(*)$.*

Finally, for a given language $\mathcal{L}$ we write $\mathcal{L}^k$ to indicate smart ballots in $\mathcal{L}$ having at most k delegations in their ordering. For instance, in Example 2 the smart ballots of all agents belong to the language BOOL^2.

2.4 Voting Rules

The final step of smart voting aggregates the vector of direct votes into an outcome. For the remainder of the paper, we will use the following rules (for a domain with abstentions $*$): 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 among $D(i) \setminus \{*\}$, and if there is a tie it returns $*$.

3 Algorithmic Analysis

We here analyse the unravelling procedures introduced in Section 2.2. For simplicity, in the rest of the paper we assume that $\mathcal{I}$ contains a single issue (thus, we drop the subscript i).

3.1 Properties of Unravelling

Given the four update procedures, our first result shows that for any valid smart profile the general unravelling procedure UNRAVEL always terminates:

Proposition 1. *Algorithms* UNRAVEL($\mathbf{U}$), UNRAVEL($\mathbf{DU}$), UNRAVEL($\mathbf{RU}$) *and* UNRAVEL($\mathbf{DRU}$) *always terminate on a valid smart profile* $\boldsymbol{B}$.

Proof (sketch). The proof (omitted in the interest of space) rests on the fact that the two **while** loops in UNRAVEL cannot be in an infinite cycle due to each B_a having a finite number of preference levels,[6] the final preference being a direct vote, and each procedure updating at least one agent's direct vote in X at each iteration (and removing none). $\square$

In the next proposition we prove that the four update procedures actually give different results.

Proposition 2. *There exists a valid smart profile* $\boldsymbol{B}$ *for which* UNRAVEL($\mathbf{U}$), UNRAVEL($\mathbf{DU}$), UNRAVEL($\mathbf{RU}$), *and* UNRAVEL($\mathbf{DRU}$) *give different outputs.*

Proof. Consider the smart profile of Example 2: the outcomes of the four unravelling procedures are shown in Table 1. It is clear that procedures $\mathbf{U}$ and $\mathbf{DU}$ lead to different outcomes than the others. Although procedures $\mathbf{RU}$ and $\mathbf{DRU}$ give the same profiles of direct votes, these outcomes occur at different rates. $\square$

[5] In Example 2, in case $x_B = 0$ then necessarily $B \wedge C$ will be false, and thus $x_A = 0$ (even if $x_C = \Delta$).

[6] Recall that since $D(i)$ 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.

UNRAVEL	Output $X \in D(i)^n$	Random voter
U	$(0, 1, 0, 1, 1)$	-
DU	$(0, 1, 0, 1, 0)$	-
RU	$(0, 1, 0, 0, 0)$	C
	$(1, 1, 1, 1, 1)$	D
	$(1, 1, 1, 1, 1)$	E
DRU	$(0, 1, 0, 0, 0)$	C
	$(1, 1, 1, 1, 1)$	D

Table 1: The last column indicates, for procedures $\mathbf{RU}$ and $\mathbf{DRU}$, which voters have been picked (only one random choice here).

Observe that the proof above implies that the collective decision can be different depending on the unravelling used, and on the randomly chosen voter. The majority outcome would be 1 in two thirds of the cases with procedure $\mathbf{RU}$, whereas only in half of the cases when using procedure $\mathbf{DRU}$.

3.2 Computational Complexity

The first problem we study is how hard it is to verify that a smart ballot is valid with respect to language $\mathcal{L}$. The VALIDB($\mathcal{L}$) problem takes as input a smart ballot B_a in $\mathcal{L}$ and agents $\mathcal{N}$, and it asks if B_a is valid for $\mathcal{L}$.

Theorem 1. *VALIDB(BOOL^k) is NP-complete, for $k \geq 1$.*

Proof. For membership, observe that for B_a to be a valid BOOL^k ballot, it needs to verify the following properties (for $1 \leq h, \ell \leq k$), that can either be checked in polynomial time by reading B_a, or require a (polynomial) certificate. The properties are: there are $\leq k$ top preferences of the form (S_a^h, φ_a^h); there is one direct vote in $\{0, 1\}$ as final preference; each φ_a^h is in DNF; each S_a^h is such that $a \notin S_a^h$; and each φ_a^h is such that $Var(\varphi_a^h) \subseteq S_a^h \subseteq \mathcal{N}$. For the final three properties, we have to guess certificates: at most k to check that each φ_a^h is not a tautology ($\neg\varphi_a^h$ is satisfiable); at most k to check that each φ_a^h is not a contradiction (φ_a^h is satisfiable); at most $\frac{k}{2}(k-1)$ to check that for all φ_a^h and φ_a^ℓ such that $h \neq \ell$, φ_a^h and φ_a^ℓ are not logically equivalent (i.e., $\neg(\varphi_a^h \leftrightarrow \varphi_a^\ell)$ is satisfiable). All this requires at most $\frac{k}{2}(k+3)$ certificates for constant k. Thus, VALIDB(BOOL^k) is in NP.

For hardness, we reduce from the NP-complete problem DNF-FALSIFIABLE,[7] whose input is a formula φ in DNF. We create an instance of VALIDB(BOOL^k) where $\mathcal{N} = Var(\varphi) \cup \{a, b\}$, for fresh variables a and b, $D(i) = \{0, 1\}$ and $B_a = ((\mathcal{N}\backslash\{a\}, \varphi \vee b) > 1)$. We now show that DNF-FALSIFIABLE has a positive answer if and only if the answer to VALIDB(BOOL^k) on our instance is also positive.

Assume that φ is falsifiable. By construction of B_a, we only need to check that $\varphi \vee b$ is neither a contradiction nor a tautology: this follows from φ being falsifiable and b being a fresh variable. Assume that φ is not falsifiable, i.e., φ is a tautology. Thus, agent a provides a function $\varphi \vee b$ which is a tautology, and hence B_a is not valid. Therefore, VALIDB(BOOL^k) is NP-complete. □

[7] Since SAT-CNF is NP-hard, by the duality principle DNF-FALSIFIABLE is also NP-hard; for membership in NP it suffices to verify a falsifying truth assignment in polynomial time.

Although checking if a BOOL^k smart ballot is valid is not a tractable problem, it is as hard as the SAT problem for propositional formulas, for which efficient solvers exist.

Next, we show that our unravelling procedures terminate in polynomial time given BOOL ballots. We refer to this problem as UNRAVEL(#)$^{\mathcal{L}}$ for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$. Observe that the size of the input for UNRAVEL(#)$^{\text{BOOL}}$ is in $\mathcal{O}(\max_p(\boldsymbol{B}) \cdot n \cdot \max_\varphi(\boldsymbol{B}))$, where $\max_p(\boldsymbol{B})$ is the highest preference level of any ballot in $\boldsymbol{B}$ and $\max_\varphi(\boldsymbol{B})$ is the maximum length of any formula from an agent in $\boldsymbol{B}$.

Proposition 3. *UNRAVEL(#)$^{\text{BOOL}}$ terminates in at most $\mathcal{O}(n^2 \cdot \max_p(\boldsymbol{B}) \cdot 2\max_\varphi(\boldsymbol{B}))$ time steps, for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$.*

Proof. The **while** loop from line 2 in UNRAVEL can be repeated at most n times (in case just one direct vote is added to X at each iteration). Moreover, the **while** loop from line 5 can be repeated at most $\max_p(\boldsymbol{B})$ times, in case all smart ballots are of the same length and no vote is computable in the first $\max_p(\boldsymbol{B}) - 1$ positions for any agent.

The following is executed at most $n \cdot \max_p(\boldsymbol{B})$ times: UPDATE(#) checks that for each agent a such that $x_a = \Delta$ (at most n) either $B_a^{\mathtt{lev}} \in D(i)$ or $\varphi_a^{\mathtt{lev}}$ has a necessary winner (depending on the # used). As each $\varphi_a^{\mathtt{lev}}$ is in DNF, i.e., a disjunction of conjunctions of literals (called *cubes*), to verify if it has a necessary winner we check if either:

1. all literals of a cube of $\varphi_a^{\mathtt{lev}}$ are made true by $X_{\restriction S_a^{\mathtt{lev}}}$,
2. one literal in each cube is made false by $X_{\restriction S_a^{\mathtt{lev}}}$,

returning a direct vote of 1 or 0, respectively. The use of UPDATE(#) takes at most $\mathcal{O}(n \cdot 2\max_\varphi(\boldsymbol{B}))$ steps. Hence, in $\mathcal{O}(n^2 \cdot \max_p(\boldsymbol{B}) \cdot 2\max_\varphi(\boldsymbol{B}))$ time steps a profile X of direct votes is output by UNRAVEL(#)$^{\text{BOOL}}$. □

4 Ranked Singleton Delegations

In this section we focus on the language LIQUID^k from Definition 7. Agents are restricted to express either a direct vote or a (partial) ranking of single-agent delegations.

4.1 Liquid Democracy

We begin by showing that our unravelling procedures yield the same result on the language LIQUID^1, corresponding to the simplest setting of liquid democracy:

Proposition 4. *If $\boldsymbol{B} \in$ LIQUID^1 then UNRAVEL(#) outputs the same result for $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$.*

Proof (sketch). At the first iteration step, all direct votes are added to the vector X. Each unravelling procedure then computes the votes of those agents who are delegating but are not in a delegation cycle (possibly not in the same order, but with the same result). Cycles are broken by looking at backup votes of possibly different agents, which are all abstentions by the definition of LIQUID^1. □

Now, consider an example of liquid democracy. By creating a profile of smart ballots with $B_a = ((\{b\}, id) > *)$ if agent a was delegating her decision to agent $b \in \mathcal{N}\backslash\{a\}$ and $B_a = (x)$ with $x \neq *$ if agent a was voting directly, we obtain the following result:

Proposition 5. *Liquid Democracy can be translated into a smart voting election with* LIQUID^1 *ballots and* UNRAVEL*(#) for* $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$.

The proof of Proposition 5 is omitted for lack of space.

4.2 Participation Axioms

In this section we study two properties of unravelling procedures, focussing on a binary domain (with abstentions). The first, proposed by Kotsialou and Riley [2020], was inspired by the classical participation axiom from social choice [Moulin, 1988]. Both properties focus on a voter's incentive to participate in the election, either by voting directly or by delegating. Thus, we assume that an agent a expressing a direct vote for alternative $x \in \{0,1\}$ prefers x over $1-x$, denoted by $x >_a 1-x$, and that a prefers x over abstention, $x >_a *$.

Definition 8 (Cast-Participation)**.** *A voting rule* r *and unravelling procedure* $\mathcal{U}$ *satisfy* cast-participation *if for all valid smart profiles* $\boldsymbol{B}$ *and agents* $a \in \mathcal{N}$ *such that* $B_a \in D(i) \backslash \{*\}$

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

for all $B'_a \neq B_a$, *and* $\boldsymbol{B}_{-a}$ *is equal to* $\boldsymbol{B}$ *without* a*'s ballot.*

Cast-participation implies that agents who vote directly have an incentive to do so, rather than to express any other ballot (recall our restriction to ranked singleton delegations).

A voting rule r on the domain $\{0,1,*\}^{\mathcal{N}}$ satisfies *monotonicity* if for any profile X, if $r(X) = x$ with $x \in \{0,1\}$ then $r(X_{+x}) = x$, where profile X_{+x} is obtained from X by having one voter switch from an initial vote of $1-x$ to x or $*$, or from an initial vote of $*$ to x. Observe that all rules introduced in Section 2.4 satisfy this property. Due to this definition we can now show the following:

Theorem 2. *Any monotonic rule* r *with unravelling procedure* UNRAVEL$(\#)$ *for* $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ *satisfies cast-participation for* LIQUID.

Proof (sketch). Without loss of generality, assume that for agent $a \in \mathcal{N}$ we have $B_a = (1)$; thus for a it is the case that $1 >_a 0$. To falsify cast-participation, we need to construct a profile $\boldsymbol{B}$ such that $r(\text{UNRAVEL}(\#)(\boldsymbol{B}))$=0 or $*$, and a smart ballot B'_a such that $r(\text{UNRAVEL}(\#)(\boldsymbol{B}_{-a}, B'_a))$=1.

If a now delegates to an agent with a direct vote for 1, the outcome does not change. Therefore, all voters $c \in I^{\#}_{*}(a, \boldsymbol{B})$ vote for 1 in $\boldsymbol{B}$, but vote for either 0 or $*$ in $\boldsymbol{B}'$ (i.e., the final votes of $\boldsymbol{B}'$ can be obtained from those of $\boldsymbol{B}$ by switching 1s to 0s or $*$s). Moreover, all $c \notin I^{\#}_{*}(a, \boldsymbol{B})$ do not change their vote from $\boldsymbol{B}$ to $\boldsymbol{B}'$. Thus, this contradicts the monotonicity assumption of voting rule r. ☐

The theorem above does not hold for non-singleton delegations—we omit the proof in the interest of space.

We now focus on the incentive that a voter has to receive and accept delegations. Recall that $I^{\#}_{*}(\boldsymbol{B}, a)$ is the set of agents who are directly or indirectly influenced by a's vote.

Definition 9 (Guru-participation)**.** *A voting rule* r *and unravelling procedure* $\mathcal{U}$ *satisfy the* guru-participation property *if and only if for all profiles* $\boldsymbol{B}$ *and all agents* $a \in \mathcal{N}$ *such that* $B_a = (x)$ *with* $x \in D(i) \setminus \{*\}$ *we have that*

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

for any $b \in I^{\#}_{*}(\boldsymbol{B}, a)$, *and* $\boldsymbol{B}_{-b}$ *is* $\boldsymbol{B}$ *without* b*'s ballot.*

We now show that all four unravelling procedures we propose do not satisfy this property for a specific rule r:

Theorem 3. *RMaj and* UNRAVEL$(\#)$ *do not satisfy guru-participation for* $\# \in \{\mathbf{U}, \mathbf{DU}, \mathbf{RU}, \mathbf{DRU}\}$ *for* LIQUID.

Proof. Consider a smart profile $\boldsymbol{B}$ with $B_a = (1)$, $B_b = ((\{c\}, id) > (\{a\}, id) > *)$, $B_c = ((\{d\}, id) > (\{f\}, id) > *)$, $B_d = ((\{b\}, id) > (\{f\}, id) > *)$, $B_e = (1)$ and $B_f = (0)$ and profile $\boldsymbol{B}' = (\boldsymbol{B}_{-b}, (*))$ obtained from $\boldsymbol{B}$ by switching b's vote to $B'_b = (*)$.

The outcomes of the four procedures are shown here:

#	$\boldsymbol{B}$	$\boldsymbol{B}'$
U/ DU	$X_1 = (1,1,0,0,1,0)$	$X_2 = (1,*,*,*,1,0)$
RU/ DRU	$X_3 = (1,1,1,1,1,0)$ $X_4 = (1,0,0,0,1,0)$ $X_5 = (1,0,0,0,1,0)$	

By applying unravelling procedures $\mathbf{U}$ and $\mathbf{DU}$, agent a prefers the outcome from $\boldsymbol{B}'$ to $\boldsymbol{B}$, since RMaj$(X_1) = *$ and RMaj$(X_2) = 1$. For procedures $\mathbf{RU}$ and $\mathbf{DRU}$, the outcome on $\boldsymbol{B}'$ is RMaj$(X_2) = 1$. However, their outcome on $\boldsymbol{B}$ can be either RMaj(X_4) = RMaj$(X_5) = 0$ or RMaj$(X_3) = 1$. Agent a strictly prefers the outcome from $\boldsymbol{B}'$, which is certainly 1, over profile $\boldsymbol{B}$ which leads to an outcome of 0 for two thirds of the cases. ☐

Observe that the profile in the above proof shows that our unravelling procedures differ from those of Kotsialou and Riley [2020], as their depth-first procedure on $\boldsymbol{B}$ would output $(1,0,1,0,1,0)$, while their breadth-first procedure on $\boldsymbol{B}'$ would give $(1,*,0,0,1,0)$. The breadth-first procedure does satisfy guru-participation, but at the price of using delegations that are quite low in the voters' rankings.

5 Conclusion

In this paper we propose and study an extension of liquid democracy that accounts for ranked and multi-voter delegations. We introduce four unravelling procedures to transform voters' ballots into profiles of direct votes, on which a collective decision is taken using a standard voting rule. Our procedures are polynomial, and aim at making use of the highest-ranked delegations when breaking delegation cycles.

With our proposal we want to put forward a general framework to study delegative voting, with notable examples being the classical settings of liquid democracy. Future work will include the investigation of further axiomatic properties for unravelling procedures and delegative voting, in line with the participation axioms, and a more fine-grained analysis of restricted languages for smart ballots.

Acknowledgments

The authors acknowledge the support of the ANR JCJC project SCONE (ANR 18-CE23-0009-01). This paper developed from ideas discussed at the Dagstuhl Seminar 19381 (Application-Oriented Computational Social Choice), 2019.

References

[Abramowitz and Mattei, 2019] Ben Abramowitz and Nicholas Mattei. Flexible representative democracy: An introduction with binary issues. In *Proc. of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[Auletta *et al.*, 2018] Vincenzo Auletta, Diodato Ferraioli, and Gianluigi Greco. Reasoning about consensus when opinions diffuse through majority dynamics. In *Proc. of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[Behrens *et al.*, 2014] J. Behrens, A. Kistner, A. Nitsche, and B. Swierczek. *Principles of Liquid Feedback*. Interacktive Demokratie, 2014.

[Bloembergen *et al.*, 2019] Daan Bloembergen, Davide Grossi, and Martin Lackner. On rational delegations in liquid democracy. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[Bredereck and Elkind, 2017] Robert Bredereck and Edith Elkind. Manipulating opinion diffusion in social networks. In *Proc. of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[Brill and Talmon, 2018] Markus Brill and Nimrod Talmon. Pairwise liquid democracy. In *Proc. of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[Brill, 2018] Markus Brill. Interactive democracy. In *Proc. of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2018.

[Caragiannis and Micha, 2019] Ioannis Caragiannis and Evi Micha. A contribution to the critique of liquid democracy. In *Proc. of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[Christoff and Grossi, 2017] Zoé Christoff and Davide Grossi. Binary voting with delegable proxy: An analysis of liquid democracy. In *Proc. of the 16th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, 2017.

[Cohensius *et al.*, 2017] Gal Cohensius, Shie Mannor, Reshef Meir, Eli A. Meirom, and Ariel Orda. Proxy voting for better outcomes. In *Proc. of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2017.

[Degrave, 2014] Jonas Degrave. Resolving multi-proxy transitive vote delegation. *arXiv preprint arXiv:1412.4039*, 2014.

[Dhillon *et al.*, 2019] Amrita Dhillon, Grammateia Kotsialou, Peter McBurney, Luke Riley, et al. Introduction to voting and the blockchain: some open questions for economists. Technical report, Competitive Advantage in the Global Economy (CAGE), 2019.

[Easley and Kleinberg, 2010] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[Escoffier *et al.*, 2019] Bruno Escoffier, Hugo Gilbert, and Adèle Pass-Lanneau. The convergence of iterative delegations in liquid democracy in a social network. In *Proc. of the 12th International Symposium on Algorithmic Game Theory (SAGT)*, 2019.

[Escoffier *et al.*, 2020] Bruno Escoffier, Hugo Gilbert, and Adèle Pass-Lanneau. Iterative delegations in liquid democracy with restricted preferences. In *Proc. of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[Gölz *et al.*, 2018] Paul Gölz, Anson Kahng, Simon Mackenzie, and Ariel D Procaccia. The fluid mechanics of liquid democracy. In *International Conference on Web and Internet Economics (ICWIE)*, 2018.

[Grandi *et al.*, 2015] Umberto Grandi, Emiliano Lorini, and Laurent Perrussel. Propositional opinion diffusion. In *Proc. of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2015.

[Green-Armytage, 2015] James Green-Armytage. Direct voting and proxy voting. *Constitutional Political Economy*, 26(2):190–220, 2015.

[Hardt and Lopes, 2015] Steve Hardt and Lia C. R. Lopes. Google votes: A liquid democracy experiment on a corporate social network. Technical report, Technical Disclosure Commons, Google, 2015.

[Kahng *et al.*, 2018] Anson Kahng, Simon Mackenzie, and Ariel D Procaccia. Liquid democracy: An algorithmic perspective. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[Konczak and Lang, 2005] Kathrin Konczak and Jérôme Lang. Voting procedures with incomplete preferences. In *Proc. of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling*, volume 20, 2005.

[Kotsialou and Riley, 2020] Grammateia Kotsialou and Luke Riley. Incentivising participation in liquid democracy with breadth-first delegation. In *Proc. of the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2020.

[Kotsialou *et al.*, 2018] Grammateia Kotsialou, Luke Riley, Amrita Dhillon, Toktam Mahmoodi, Peter McBurney, Paul Massey, and Richard Pearce. Using distributed ledger technology for shareholder rights management. In *Proc. of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2018.

[Miller, 1969] James C Miller. A program for direct and proxy voting in the legislative process. *Public choice*, 7(1):107–113, 1969.

[Moulin, 1988] Hervi Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1988.

[Riley *et al.*, 2019] Luke Riley, Grammateia Kotsialou, Amrita Dhillon, Toktam Mahmoodi, Peter McBurney, and Richard Pearce. Deploying a shareholder rights management system onto a distributed ledger. In *Proc. of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2019.