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 Talmor, 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.

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. 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’

\[ \text{Smart voting} \]

\[ \text{Liquid democracy} \]

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.

[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; Bloemergen 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 Escofier et al. [2019; 2020], we let voters express a ranking of possible delegates, to break potential cycles of delegations. 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 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 Escofier 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 \( N \) of \( n \) agents (or voters) who decide on a set \( I \) of \( m \) issues. The alternative values, or simply alternatives, for each issue \( i \in I \) range over a non-empty finite domain \( D(i) \), which can also include abstention.

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

**Definition 1 (Smart Ballot).** A smart ballot of agent \( a \) for an issue \( i \in I \) is an ordering \(((S^h, F^h) > \cdots > (S^1, F^1) > x)\) where \( k \geq 0 \) and for \( 1 \leq h \leq k \) we have that \( S^h \subseteq N \) is a set of agents, \( F^h: D(i|S^h) \rightarrow D(i) \) 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 \( i \in I \) 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.” 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 \land (C \lor 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:

\[ (((B, C, D), Maj) > 1). \]

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

Given \( n \) smart ballots for an issue \( i \in I \) from each agent in \( N \), a (smart) profile for this issue \( i \), is a vector \( B_i = (B_{1i}, \ldots, B_{ni}) \). 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 \( U \) for issue \( i \in I \) and agents in \( N \) is any function \( U: (B_{1i} \times \cdots \times B_{ni}) \rightarrow D(i)^n \).

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 \( B_i \). A vector \( X \) is initialised with placeholders \( \Delta \) at each position \( x_a \) for \( a \in N \)

\[ \text{This can be expressed by propositional formula } B \land (C \lor D). \]

\[ \text{Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)} \]
and it is returned when a vote in \(D(i)\) is found for all agents. Counter \(\text{lev}\) 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, \textit{direct vote priority} (D), prioritising direct votes over those that can be computed from the current vector \(Y\) of votes; and \textit{random voter selection} (R), randomly choosing an agent whose vote is added (or computed) next. The four procedures are thus: basic update (U), update with direct vote priority (DU), update with random voter selection (RU), update with direct vote priority and random voter selection (DRU).

Unless otherwise specified, if the condition in an if statement fails, the program skips to the next step. Moreover, \(Y|S\) denotes the restriction of vector \(Y\) to the elements in set \(S\).

\textbf{Algorithm 2 \textsc{Update}(U)}

\begin{algorithmic}
  \footnotesize
  \STATE for \(a \in N\) such that \(x_a = \Delta\)
  \IF {\(B_{ai}^{lev} \in D(i)\)} \Comment{a has a direct vote at \text{lev}}
  \STATE \(x_a := B_{ai}^{lev}\)
  \ELSE if \(F_{ai}^{lev}(Y|S_{ai}^{lev}) \in D(i)\) \Comment{add a’s computable vote}
  \STATE \(x_a := F_{ai}^{lev}(Y|S_{ai}^{lev})\)
  \ENDIF
  \ENDIF
\end{algorithmic}

\textbf{Algorithm 3 \textsc{Update}(DU)}

\begin{algorithmic}
  \footnotesize
  \STATE for \(a \in N\) such that \(x_a = \Delta\)
  \IF {\(B_{ai}^{lev} \in D(i)\)} \Comment{add direct votes}
  \STATE \(x_a := B_{ai}^{lev}\)
  \ENDIF
  \IF {\(Y = X\)} \Comment{if no direct vote added to X}
  \FOR {\(a \in N\) such that \(x_a = \Delta\)}
  \IF {\(F_{ai}^{lev}(Y|S_{ai}^{lev}) \in D(i)\)} \Comment{find computables}
  \STATE \(x_a := F_{ai}^{lev}(Y|S_{ai}^{lev})\)
  \ENDIF
  \ENDFOR
  \ENDIF
\end{algorithmic}

\textbf{Algorithm 4 \textsc{Update}(RU)}

\begin{algorithmic}
  \footnotesize
  \STATE \(P := \emptyset\) \Comment{initialise an empty set}
  \FOR {\(a \in N\) such that \(x_a = \Delta\)} \Comment{for a in N such that x_a = Δ do}
  \IF {\(B_{ai}^{lev} \in D(i)\)} \Comment{add agents with direct vote to P}
  \STATE \(P := P \cup \{a\}\)
  \ELSE if \(F_{ai}^{lev}(Y|S_{ai}^{lev}) \in D(i)\) \Comment{are there direct/computable votes}
  \STATE \(x_a := F_{ai}^{lev}(Y|S_{ai}^{lev})\)
  \ENDIF
  \ENDIF
\end{algorithmic}

\textbf{Algorithm 5 \textsc{Update}(DRU)}

\begin{algorithmic}
  \footnotesize
  \STATE \(P, P' := \emptyset\) \Comment{initialise an empty set}
  \FOR {\(a \in N\) such that \(x_a = \Delta\)} \Comment{for a in N such that x_a = Δ do}
  \IF {\(B_{ai}^{lev} \in D(i)\)} \Comment{add agents with direct vote to P}
  \STATE \(P := P \cup \{a\}\)
  \ELSE if \(F_{ai}^{lev}(Y|S_{ai}^{lev}) \in D(i)\) \Comment{are there agents with direct votes}
  \STATE \(P := P \cup \{a\}\)
  \ENDIF
  \ENDIF
\end{algorithmic}

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

The following example shows how \textsc{update(U)} works:

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

<table>
<thead>
<tr>
<th>(1^{st})</th>
<th>(2^{nd})</th>
<th>(3^{rd})</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B, C}</td>
<td>{B}</td>
<td>{D}</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>{B}</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>{C}</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

We spell out here the unravelling \textsc{Unravel(U)} on \(B_i\):

\begin{itemize}
  \item Starting at \text{lev} = 1, the direct vote of \(B\) is stored in \(X = (\Delta, 1, \Delta, \Delta, \Delta)\).
  \item As there are no direct or computable votes at \text{lev} = 1 (using \(Y\)), the level counter is increased to \text{lev} = 2.
  \item 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 \(X = (\Delta, 1, 0, 1, 1)\).
\end{itemize}
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, \( F^{lev}_{ai}(X|S^{lev}_{ai}) \) has a necessary winner \( y \in D(i) \) if for all \( a \in S^{lev}_{ai} \) such that \( x_a = \Delta \), for all \( z \in D(i) \) such that \( x_a = z \) we have that \( F^{lev}_{ai}(X|S^{lev}_{ai}) = y \).

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

**Definition 4** (Certificates). An agent \( a \)’s certificate of an unravelled profile \( U(B_{ai}) \) is \( Cert^U(B_{ai}, a) = (k, X|S^k_{ai}) \), where \( k \) is a preference level and \( X|S^k_{ai} \) is the partial profile of direct votes used by \( U \) 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 \( B_{ai} \) using unravelling procedure \( U \) is \( I^U(B_{ai},a) = \{ b \mid a \in S^k_{ai} \text{ for } Cert^U(B_{ai},b) = (k, X|S^k_{ai}) \} \).

Let \( I^U(B_{ai},a) = \bigcup \{ c \mid c \in I^U(B_{ai},b) \text{ and } b \in I^U(B_{ai},a) \} \). The voters \( b \) indirectly influenced by \( a \) in Example 2, we have that \( Cert^U(B_{ai},C) = (2, \emptyset) \) and \( I^U(B_{ai},C) = \{ A \} \).

### 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 \( F^{lev}_{ai} \) 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 \( \varphi^{lev}_{ai} \) instead of \( F^{lev}_{ai} \).

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 \( B^j_{ai} \) in \( B_{ai} \) is of the form \( \{ b \mid id \} \) for \( b \in N \setminus \{ a \} \) and \( id \) the identity function, and if \( h > 1 \) the final vote of \( a \) is abstention (*).

Finally, for a given language \( L \) we write \( L^k \) to indicate smart ballots in \( L \) 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 \( I \) 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 \( UNRAVEL(U), UNRAVEL(DU), UNRAVEL(RU) \) and \( UNRAVEL(DRU) \) always terminate on a valid smart profile \( B \).

**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 \( B_{ai} \) having a finite number of preference levels,¹ 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).

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

**Proposition 2.** There exists a valid smart profile \( B \) for which \( UNRAVEL(U), UNRAVEL(DU), UNRAVEL(RU), \) and \( UNRAVEL(DRU) \) 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 \( U \) and \( DU \) lead to different outcomes than the others. Although procedures \( RU \) and \( DRU \) give the same profiles of direct votes, these outcomes occur at different rates.

¹Recall 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.
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 RU, whereas only in half of the cases when using procedure 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 \( \mathcal{L} \). Assume that a smart ballot is valid with respect to language \( \mathcal{L} \), 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 \( k \) top preferences of the form \( (S^h_a, \varphi^h_a) \); each \( \varphi^h_a \) is in DNF; each \( S^h_a \) is such that \( a \not\in S^h_a \); and each \( \varphi^h_a \) is such that \( \text{Var}(\varphi^h_a) \subseteq S^h_a \subseteq \mathcal{N} \). For the final three properties, we have to guess certificates: at most \( k \) to check that each \( \varphi^h_a \) is not a tautology (\( \neg \varphi^h_a \) is satisfiable); at most \( k \) to check that each \( \varphi^h_a \) is not a contradiction (\( \varphi^h_a \) is satisfiable); at most \( \frac{k}{2} (k-1) \) to check that for all \( \varphi^h_a \) and \( \varphi^h_b \), such that \( h \neq \ell \), \( \varphi^h_a \) and \( \varphi^h_b \) are not logically equivalent (i.e., \( \neg (\varphi^h_a \leftrightarrow \varphi^h_b) \) is satisfiable). All this requires at most \( \frac{k}{2} (k+3) \) certificates for constant \( k \). Thus, \( \text{ValidB}(\text{Bool}^k) \) is in \( NP \).

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

Assume that \( \varphi \) is falsifiable. By construction of \( B_a \), we only need to check that \( \varphi \lor b \) is neither a contradiction nor a tautology: this follows from \( \varphi \) being falsifiable and \( b \) being a fresh variable. Assume that \( \varphi \) is not falsifiable, i.e., \( \varphi \) is a tautology. Thus, agent \( a \) provides a function \( \varphi \lor b \) which is a tautology, and hence \( B_a \) is not valid. Therefore, \( \text{ValidB}(\text{Bool}^k) \) 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 \( \text{Bool}^k \) 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 \( \text{Bool}^k \) ballots. We refer to this problem as \( \text{Unravel}(\#)^{\text{Bool}} \) for \( \# \in \{U, DU, RU, DRU\} \). Observe that the size of the input for \( \text{Unravel}(\#)^{\text{Bool}} \) is in \( O(\max_p(B) \cdot n \cdot \max_p(B)) \), where \( \max_p(B) \) is the maximum preference level of any ballot in \( B \) and \( \max_p(B) \) is the maximum length of any formula from an agent in \( B \).

Proposition 3. \( \text{Unravel}(\#)^{\text{Bool}} \) terminates in at most \( \max_p(B) \cdot 2 \max_p(B) \) times, for \( \# \in \{U, DU, RU, DRU\} \).

Proof. The while loop from line 2 in \( \text{Unravel} \) 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 \( \max_p(B) \) times, in case all smart ballots are of the same length and no vote is computable in the first \( \max_p(B) - 1 \) positions for any agent.

The following is executed at most \( n \cdot \max_p(B) \) times: Update(\#) checks that for each agent \( a \) such that \( x_a = \Delta \) (at most \( n \) times) either \( B_a = D(i) \) or \( \varphi^a_{1^\text{st}} \) is a winner (depending on the \( \# \) used). As each \( \varphi^a_{1^\text{st}} \) is in DNF, i.e., a disjunction of literals (called cubes), to verify if it has a necessary winner we check if either:

1. all literals of a cube of \( \varphi^a_{1^\text{st}} \) are made true by \( X_{1^\text{st}} \).
2. one literal in each cube is made false by \( X_{1^\text{st}} \), returning a direct vote of 1 or 0, respectively.

The use of Update(\#) takes at most \( O(n \cdot 2 \max_p(B)) \) steps. Hence, in \( O(n^2 \cdot \max_p(B) \cdot 2 \max_p(B)) \) time steps a profile \( X \) of direct votes is output by \( \text{Unravel}(\#)^{\text{Bool}} \).

4 Ranked Singleton Delegations

In this section we focus on the language \( \text{Liquid}^k \) 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 \( \text{Liquid}^1 \), corresponding to the simplest setting of liquid democracy:

Proposition 4. If \( B \in \text{Liquid}^1 \) then \( \text{Unravel}(\#) \) outputs the same result for \( \# \in \{U, DU, RU, DRU\} \).

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 \( b \in \mathcal{N} \setminus \{a\} \) 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 \{U, DU, RU, 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, \(x >_a \ast\).

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

\[r(U(B)) \geq_a r(U(B_{-a}, (B'_a)))\]

for all \(B'_a \neq B_a\), and \(B_{-a}\) is equal to \(B\) 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 \(\{0, 1, \ast\}^N\) 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 an initial vote of \(1 - x\) to \(x\) or \(\ast\), or from an initial vote of \(\ast\) 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 \{U, DU, RU, DRU\} satisfies cast-participation for LIQUID.

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

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

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 \(I_r^a(B, a)\) is the set of agents that are directly or indirectly influenced by \(a\)'s vote.

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

\[r(U(B)) \geq_a r(U(B_{-a}, (\ast)))\]

for any \(b \in I_r^a(B, a)\), and \(B_{-b}\) is \(B\) 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 \{U, DU, RU, DRU\} for LIQUID.

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

The outcomes of the four procedures are shown here:

<table>
<thead>
<tr>
<th>#</th>
<th>B</th>
<th>B'</th>
</tr>
</thead>
<tbody>
<tr>
<td>U/DU</td>
<td>(X_1 = (1, 1, 0, 1, 0))</td>
<td>(X_2 = (1, 1, 1, 0, 1))</td>
</tr>
<tr>
<td>RU/</td>
<td>(X_3 = (1, 1, 1, 0, 0))</td>
<td>(X_2 = (1, 1, 1, 0, 0))</td>
</tr>
<tr>
<td>DRU/</td>
<td>(X_4 = (1, 0, 0, 0, 1))</td>
<td>(X_5 = (1, 0, 0, 0, 1))</td>
</tr>
</tbody>
</table>

By applying unravelling procedures \(U\) and \(DU\), agent \(a\) prefers the outcome from \(B'\) to \(B\), since \(RMaj(X_1) = \ast\) and \(RMaj(X_2) = 1\). For procedures \(RU\) and \(DRU\), the outcome on \(B'\) is \(RMaj(X_2) = 1\). However, their outcome on \(B\) can be either \(RMaj(X_4) = RMaj(X_5) = 0\) or \(RMaj(X_3) = 1\). Agent \(a\) strictly prefers the outcome from \(B'\), which is certainly 1, over profile \(B\) 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 \(B\) would output \((1, 0, 1, 0, 1, 0)\), while their breadth-first procedure on \(B'\) would give \((1, \ast, 0, 0, 1, 0)\). 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 voting 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 JCIC 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


