Unravelling Expressive Delegations: Complexity and Normative Analysis

GIANNIS TYROVOLAS

ANDREI CONSTANTINESCU

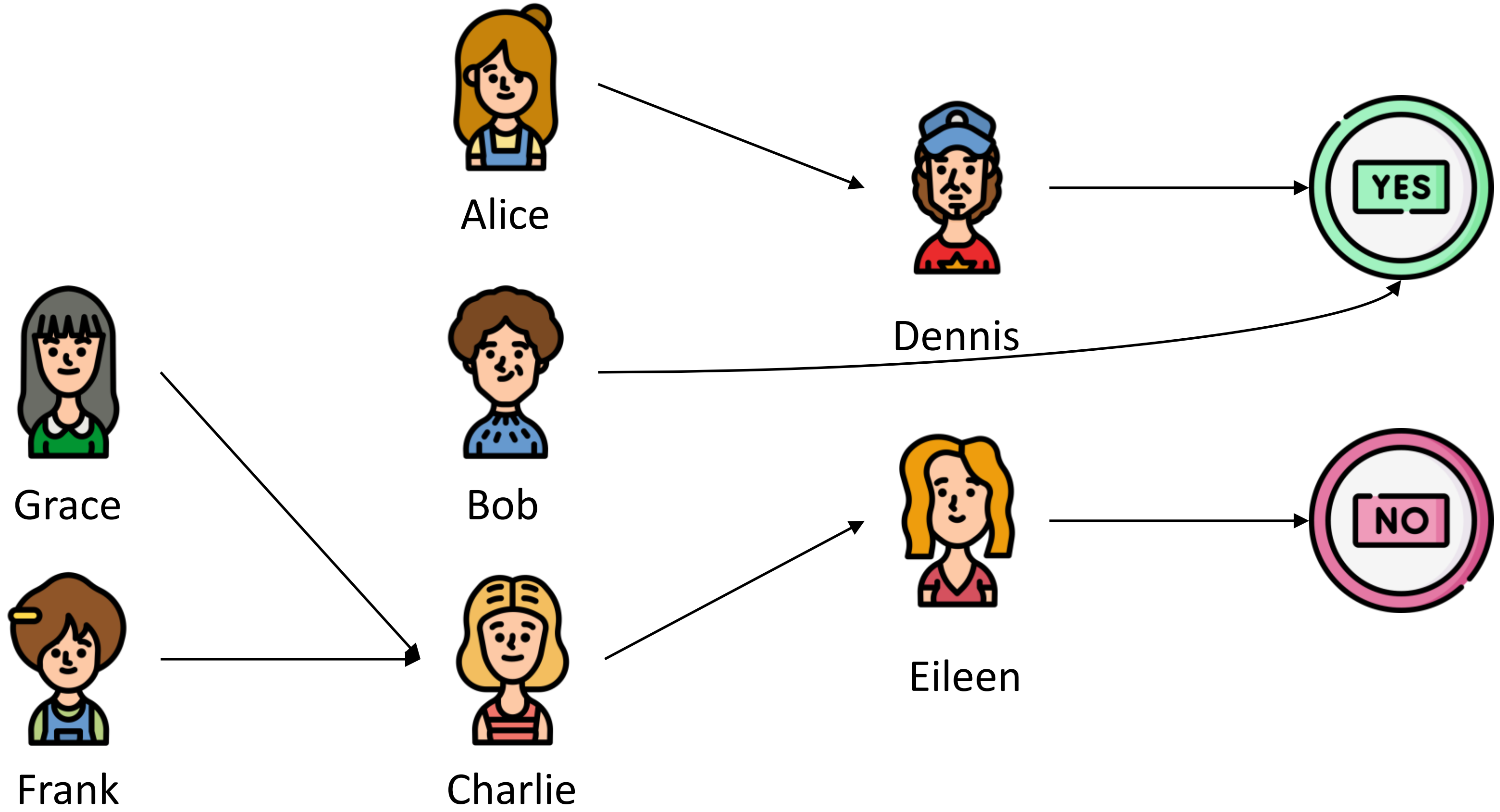
EDITH ELKIND

Summary

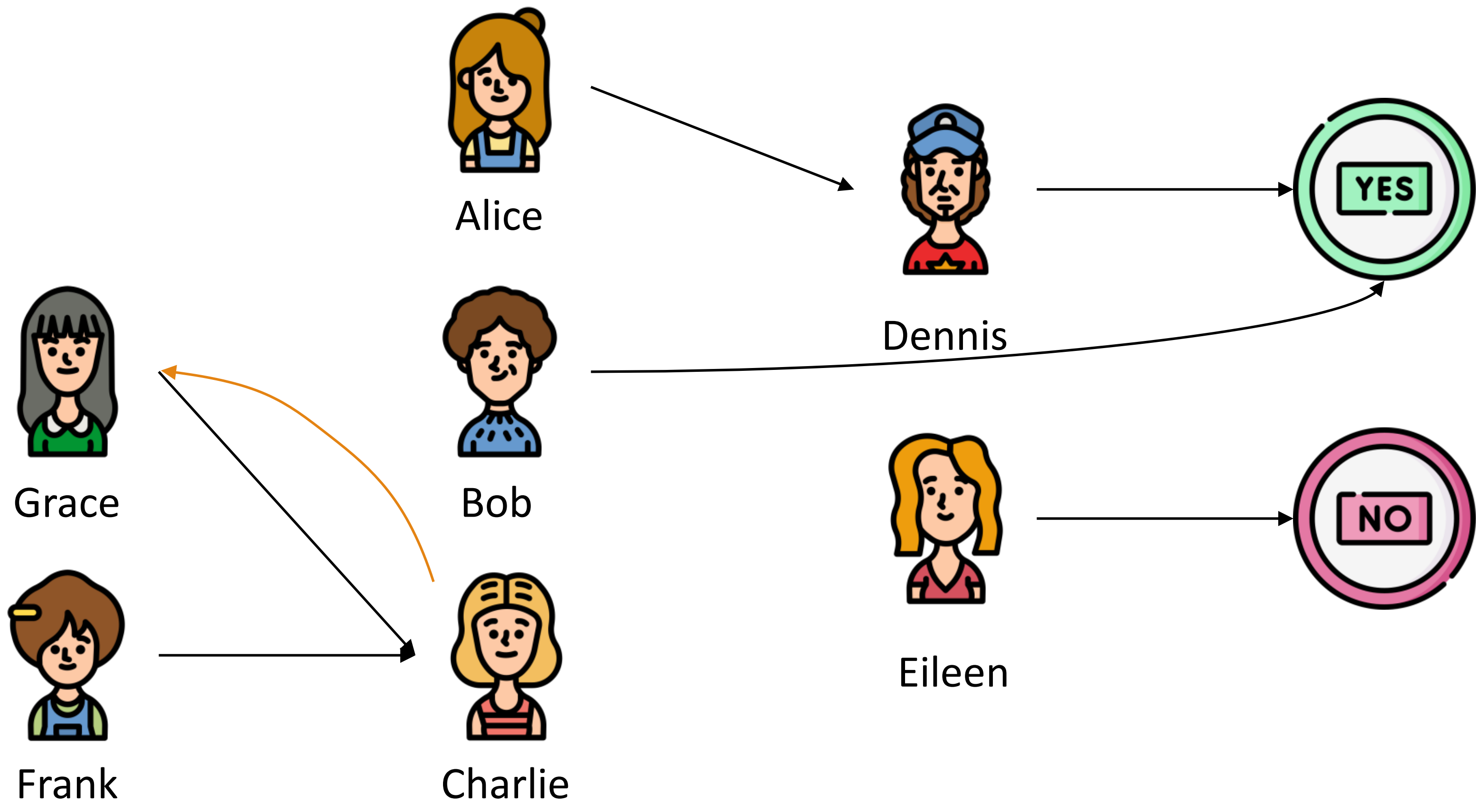
- We consider a **rich model** of Liquid Democracy.
- We prove computational **hardness** for many problems in the rich model.
- We focus on the **simpler model** and prove normative and computational results.

Liquid democracy

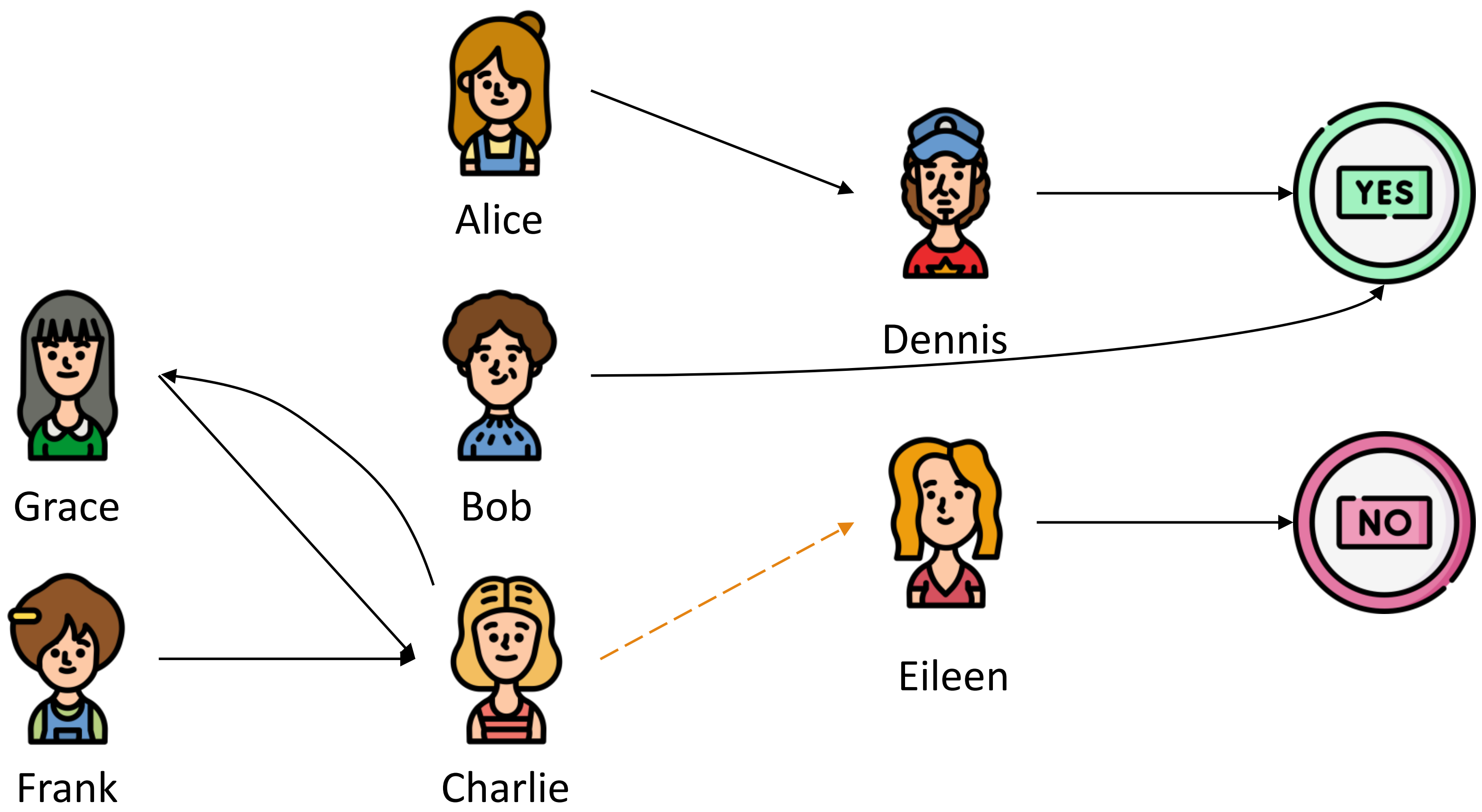
Liquid democracy allows delegations to be **transitive.**



How to deal with cycles?



Solution: ranked delegations



Summary of Liquid Democracy

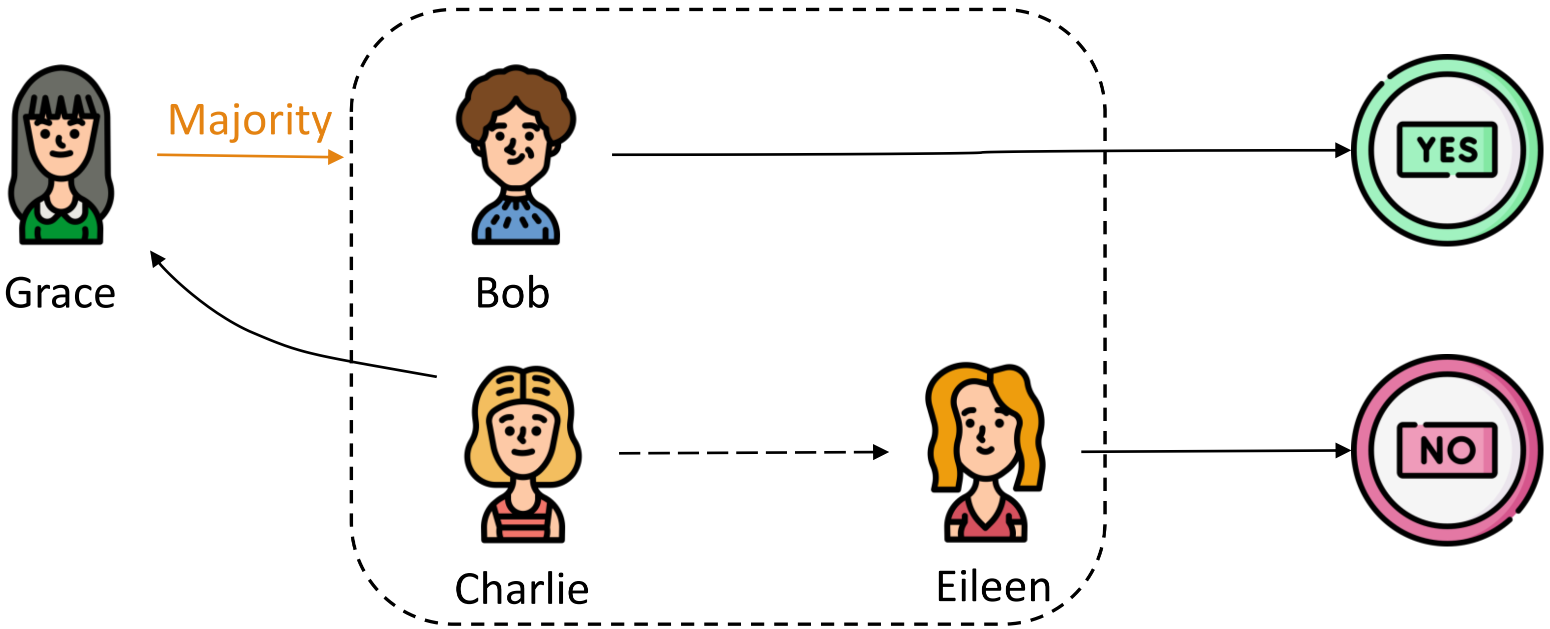
In this model of **Liquid Democracy**:

1. All voters can vote **directly** on issues.
2. Voters can delegate their votes to each other with **transitive delegations**.

Voters submit **a ranked preference order** of delegations. The **final preference** of each delegate must be for either YES or NO, to guarantee that cycles can be resolved.

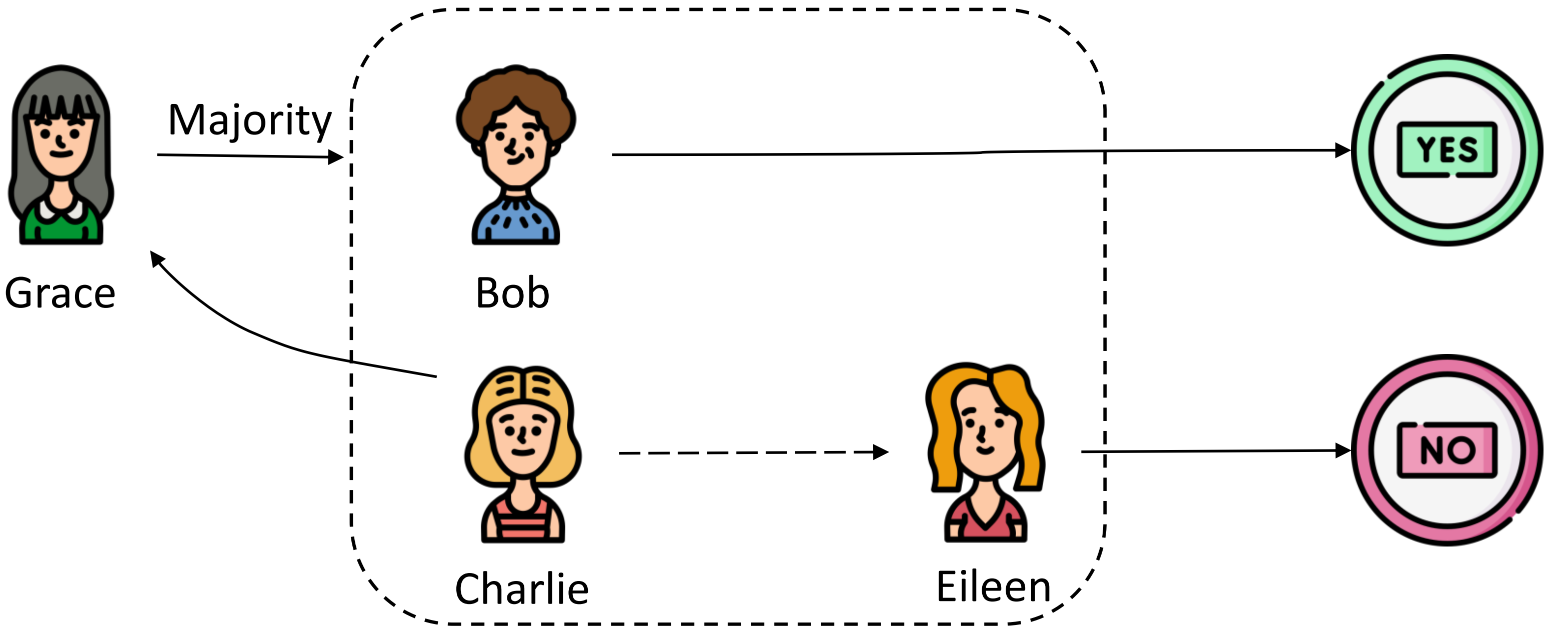
Smart voting by Colley et al. adds more expressive delegations: voters can delegate to functions of other voters.

Expressive delegation



Example ballot

$$B_B = (\text{Yes})$$
$$B_E = (\text{No})$$
$$B_C = (G > E > \text{Yes})$$
$$B_G = (\text{Maj}(B, C, E) > \text{Yes})$$

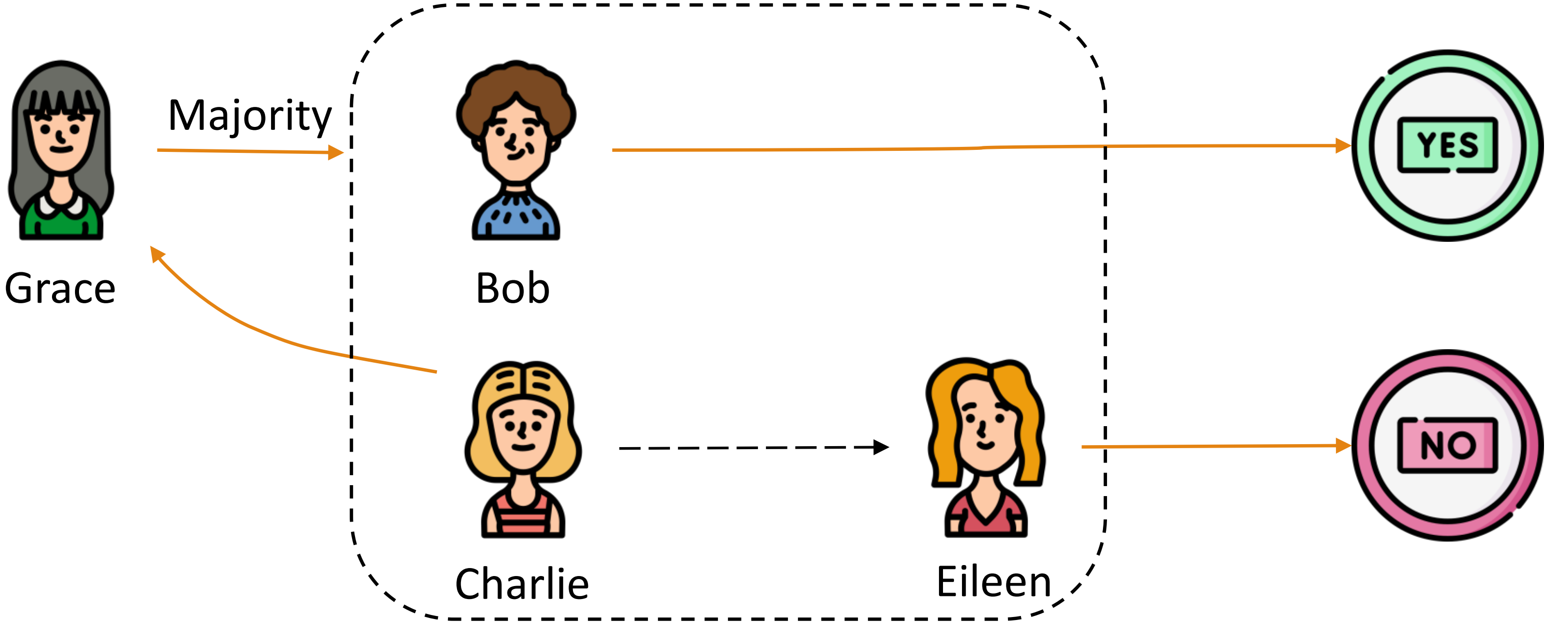


Converting ballots to votes

Given a ballot for the Smart Voting model, how can we convert it to votes for each agent?

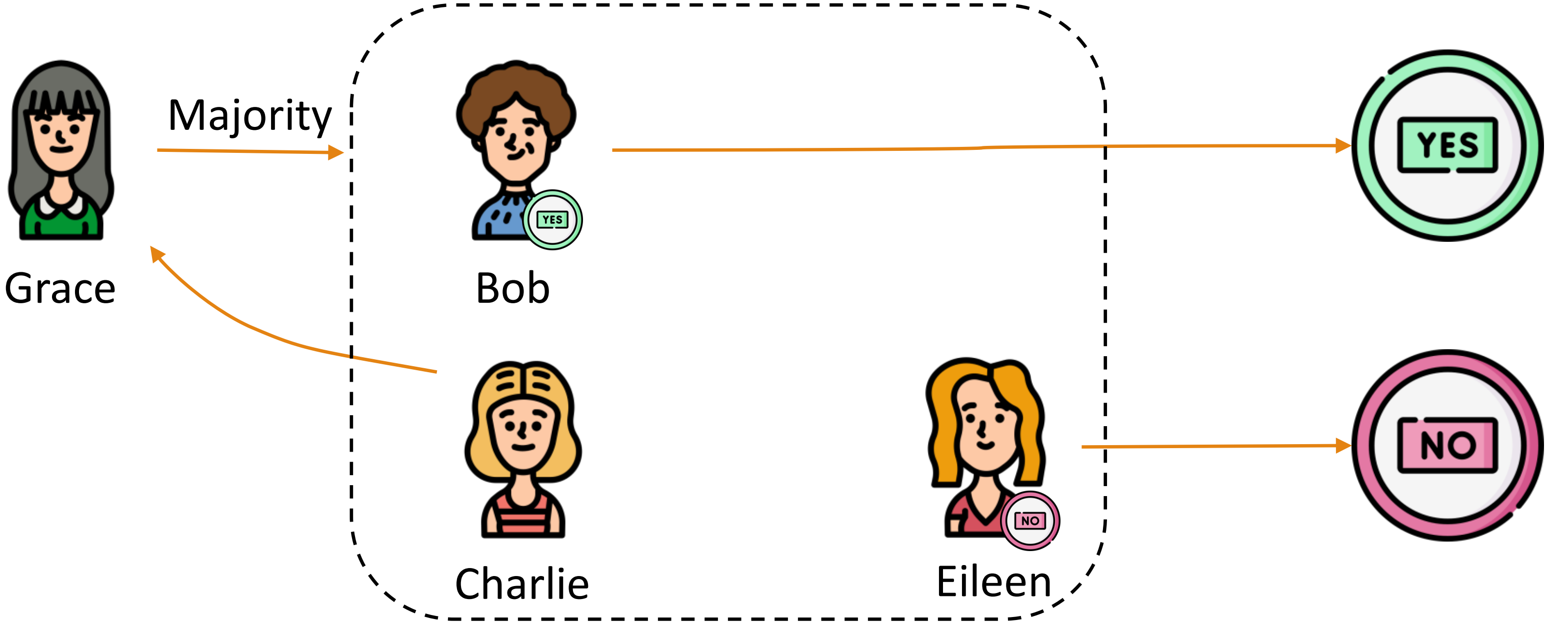
First preferences lead to cycles

$$B_B = (\textbf{Yes})$$
$$B_E = (\textbf{No})$$
$$B_C = (\boldsymbol{G} > E > \text{Yes})$$
$$B_G = (\textbf{Maj}(\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{E}) > \text{Yes})$$



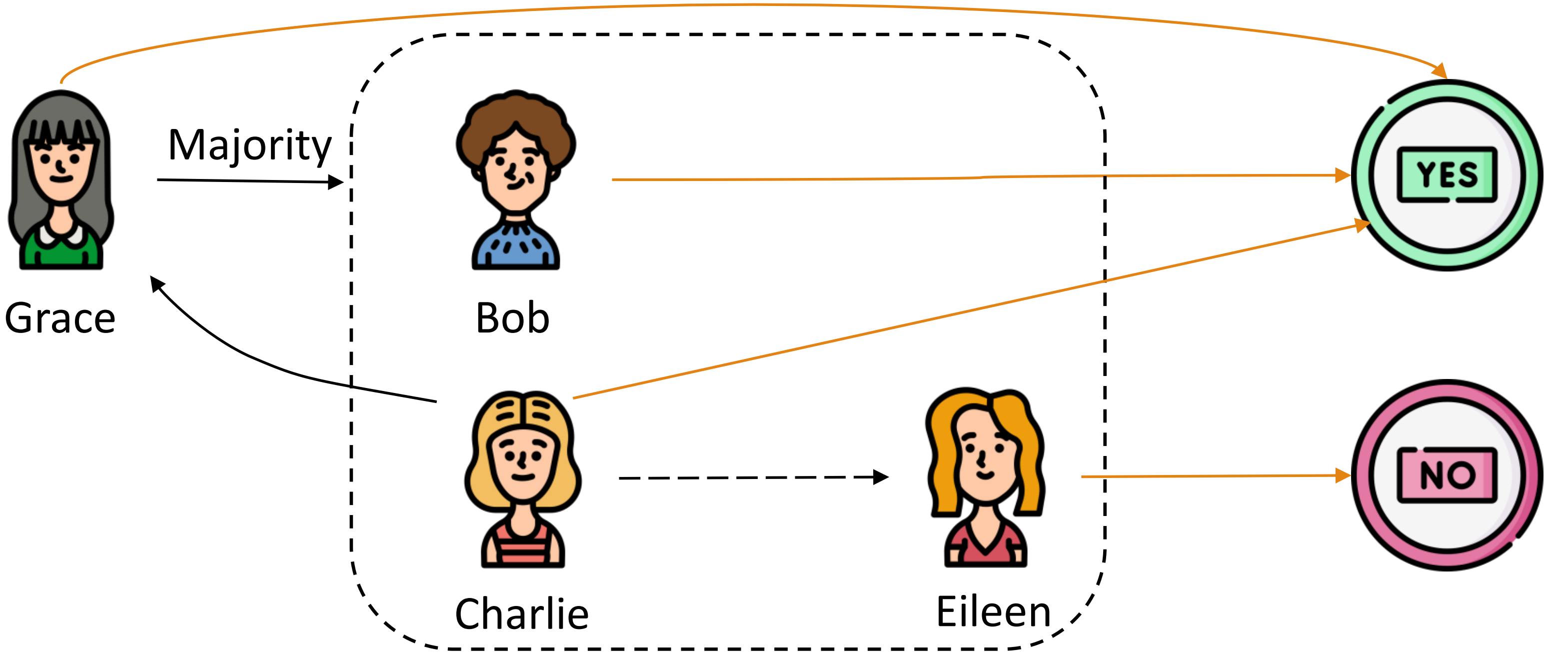
First preferences lead to cycles

$$B_B = (\textbf{Yes})$$
$$B_E = (\textbf{No})$$
$$B_C = (\boldsymbol{G} > E > \text{Yes})$$
$$B_G = (\textbf{Maj}(\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{E}) > \text{Yes})$$



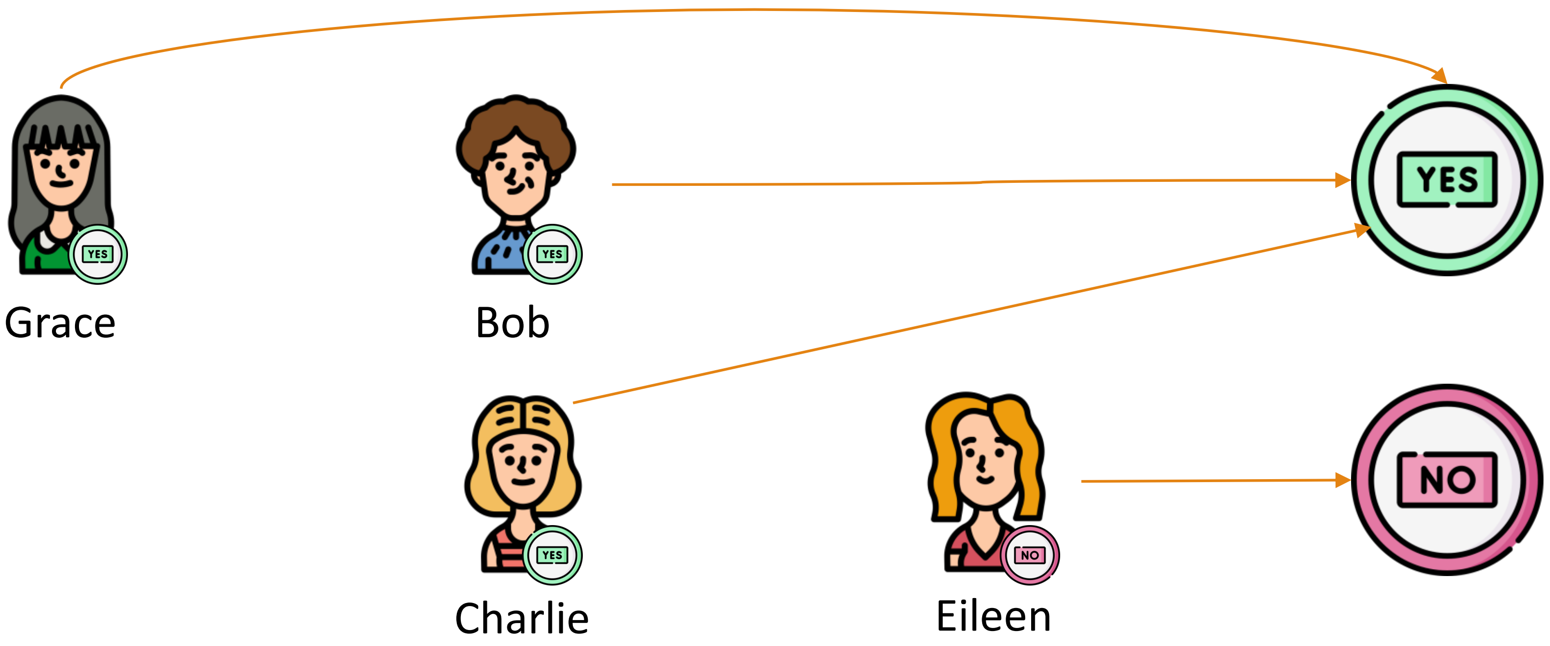
Last preferences are always consistent

$$B_B = (\mathbf{Yes})$$
$$B_E = (\mathbf{No})$$
$$B_C = (G > E > \mathbf{Yes})$$
$$B_G = (\mathrm{Maj}(B, C, E) > \mathbf{Yes})$$



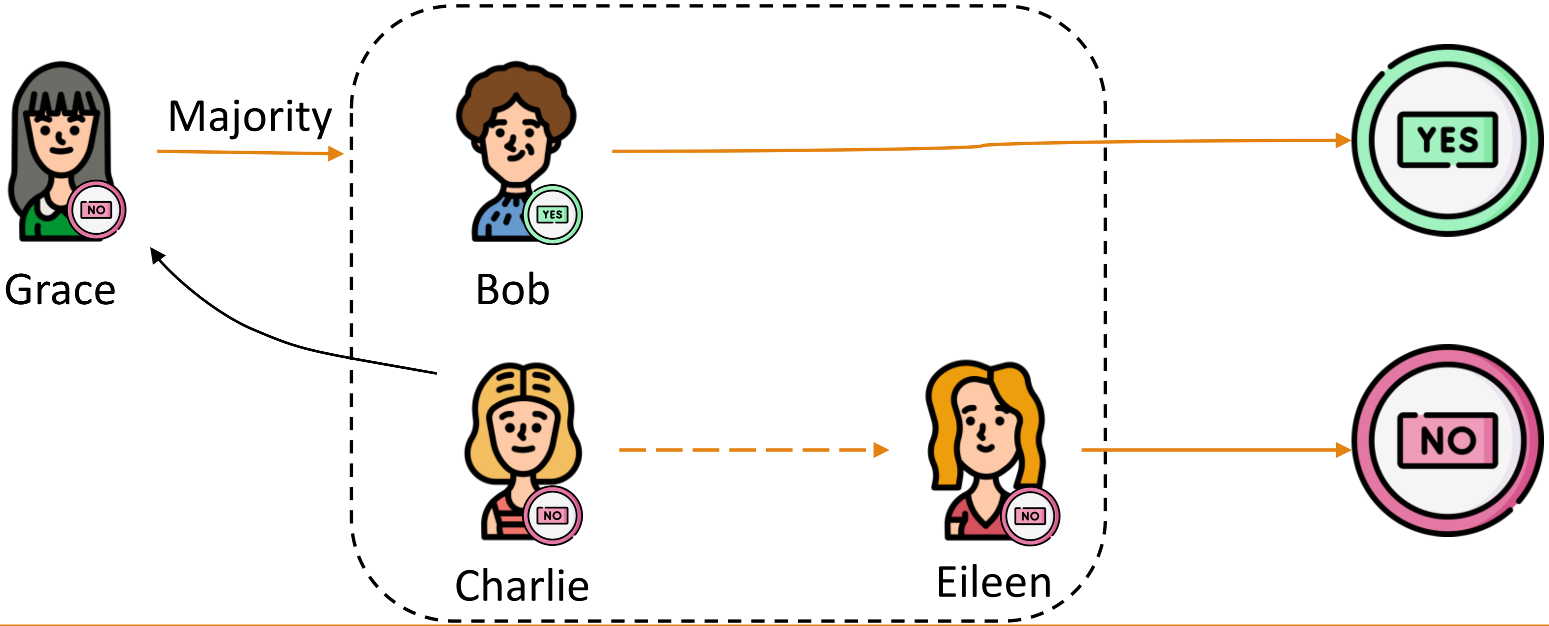
Last preferences are always consistent

$$B_B = (\textbf{Yes})$$
$$B_E = (\textbf{No})$$
$$B_C = (G > E > \textbf{Yes})$$
$$B_G = (\text{Maj}(B, C, E) > \textbf{Yes})$$



Better consistent certificate

$$B_B = (\mathbf{Yes})$$
$$B_E = (\mathbf{No})$$
$$B_C = (G > \boldsymbol{E} > \text{Yes})$$
$$B_G = (\mathbf{Maj}(\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{E}) > \text{Yes})$$



The problem

There are a lot of valid preference assignments. How can we pick the “best”?

Colley et al. introduce two notions of “**best**”:

- **MinMax:** Minimise the maximum preference level used
- **MinSum:** Minimise the sum of preference levels used

Are there **efficient** algorithms to compute these?

It turns out that the complexity of the problem depends on what **functions** agents can delegate to.

Results in Colley et al.

LIQUID: Agents can only delegate to a single other agent

$\vee$: *Binary* Boolean OR

$\wedge$: *Binary* Boolean AND

Bool: all Boolean functions

	LIQUID	$LIQUID \cup \{\vee\}$ or $LIQUID \cup \{\wedge\}$	$LIQUID \cup \{\wedge_{k=1}^{n}\}$	$LIQUID \cup \{\vee, \wedge\}$	Monotone $f \notin \{\vee^n, \wedge^n, id\}$ $LIQUID \cup \{f\}$	*Bool*
MinMax	EASY	?	?	?	?	HARD
MinSum	EASY	?	HARD	?	?	HARD

Our results

LIQUID: Agents can only delegate to a single other agent

$\vee$: *Binary* Boolean OR

$\wedge$: *Binary* Boolean AND

Bool: all Boolean functions

This is a **complete computational dichotomy** for monotone functions

	LIQUID	$LIQUID \cup \{\vee\}$ or $LIQUID \cup \{\wedge\}$	$LIQUID \cup \{\wedge_{k=1}^{n}\}$	$LIQUID \cup \{\vee, \wedge\}$	Monotone $f \notin \{\vee^n, \wedge^n, id\}$ $LIQUID \cup \{f\}$	*Bool*
MinMax	EASY	EASY	EASY	HARD	HARD	HARD
MinSum	EASY	HARD	HARD	HARD	HARD	HARD

Robustness of hardness

Our hardness results are **robust**.

When we identify hardness for a class of functions $\mathcal{F}$ then:

MinSum$_{\mathcal{F}}$ is NP-hard even if agents are only allowed **one** non-constant delegation.

MinMax$_{\mathcal{F}}$ is NP-hard even if agents are only allowed **two** non-constant delegations.

A constant factor **approximation** of either problem is NP-hard.

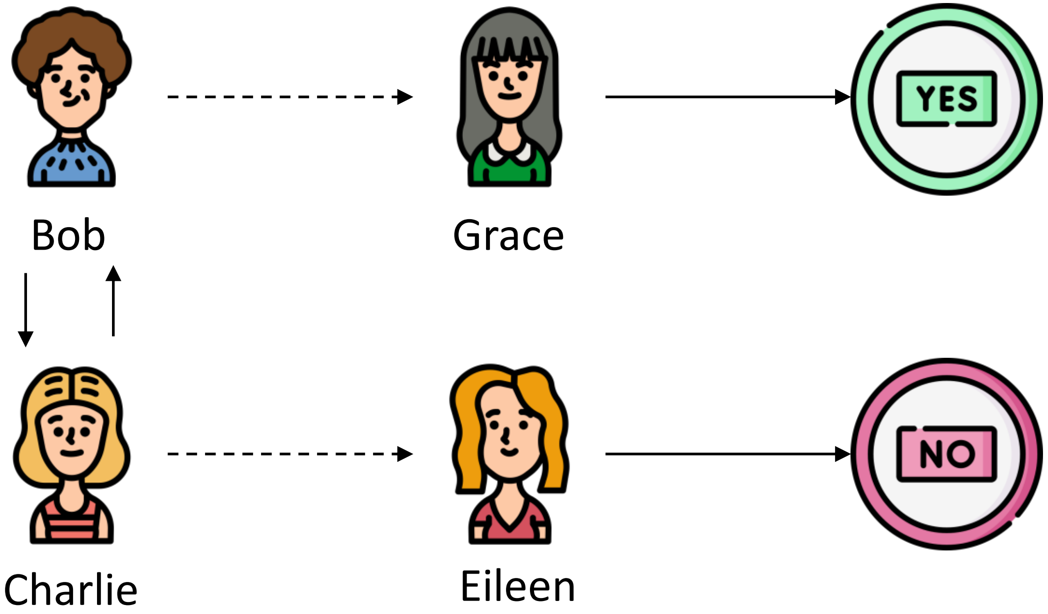
Focusing on the simpler model

Given this hardness, let’s focus on the **simpler** model.

In the simple setting, we can **efficiently** compute a MinSum and a MinMax outcome.

However, there are **multiple** such outcomes. How should we pick one?

Example of tied outcome



Grace always votes for **YES**
Eileen always votes for **NO**
Bob and **Charlie** can vote in some outcomes for **YES** and in some outcomes for **NO**

Structure of MinSum outcomes

There exists a **MinSum** outcome $\boldsymbol{c}_{YES}$ such that if voter v votes for YES in a MinSum outcome they also vote for YES in $\boldsymbol{c}_{YES}$.

Similarly, there exists a $\boldsymbol{c}_{NO}$.

The same result holds for **MinMax**.

The outcomes $\boldsymbol{c}_{YES}$ and $\boldsymbol{c}_{NO}$ can be found in **polynomial time**.

	v_1	v_2	v_3	v_4	v_5
$\mathbf{c}_{YES}$	1	1	0	1	1
$\mathbf{c}_1$	1	0	0	1	1
$\mathbf{c}_2$	0	1	0	1	1
$\mathbf{c}_3$	1	0	0	1	0
$\mathbf{c}_{NO}$	0	0	0	1	0

Biased tie-breaking

So, we introduce new **resolute** rules for **MinMax** and **MinSum** that break ties in favour of a given alternative.

This tie-breaking can be used when there's a **default** option. For example, when voting to change the status quo.

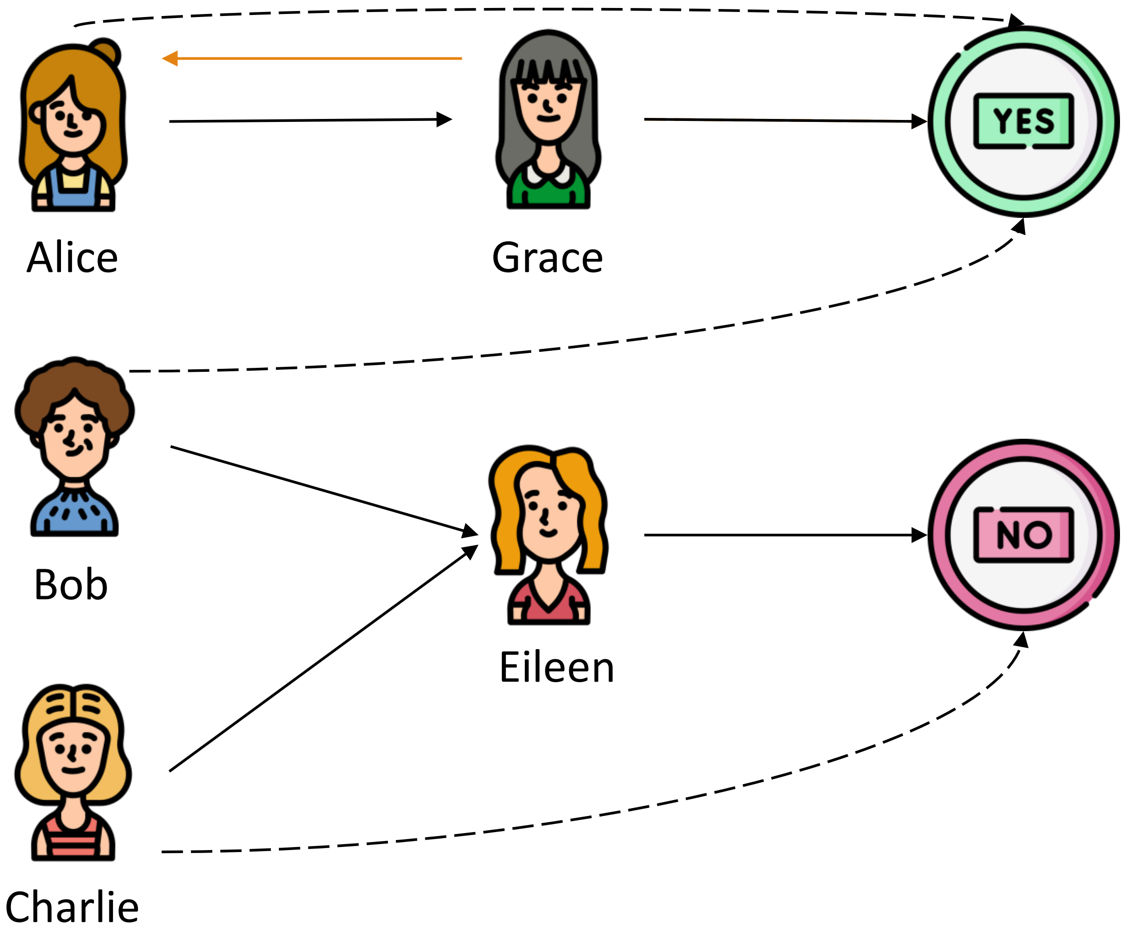
Cast-monotonicity

We introduce a new axiom named **cast-monotonicity**.

It captures the intuition that if agents have a preference over **YES** or **NO**, then their best course of action is to always vote for their preferred outcome.

For irresolute rules we consider that agents who prefer **YES** over **NO** also prefer {**YES**} over {**YES**, **NO**} over {**NO**}.

MinMax does **not** satisfy cast monotonicity



The outcome using only first preferences would result to the majority voting for **NO**. So, the outcome set is {**NO**}.

But Grace is incentivised to introduce a cycle.

If Grace introduces a cycle by voting for Alice, **MinMax** will return all valid outcomes that use at most second preferences. It will also return the outcome where Bob votes for **YES**. Making the outcome set {**YES**, **NO**}.

Cast-monotonicity

So, **MinMax** does not satisfy *cast-monotonicity*.

MinSum and the resolute variants with **biased tie-breaking** satisfy *cast-monotonicity.*

Summary

We prove **a characterisation result** for the complexity of monotone functions for MinSum and MinMax.

We propose **resolute** and efficiently computable rules for **biased tie-breaking**.

We introduce *cast-monotonicity* and prove **MinSum** satisfies it, but **MinMax** does not.

References

Icons were taken from Flaticon

The model we consider was proposed by Colley et al. in:

Colley, Rachael, Umberto Grandi, and Arianna Novaro. "Unravelling multi-agent ranked delegations." *Autonomous Agents and Multi-Agent Systems* 36.1 (2022): 9.