Styx DAO

Arthur Vinciguerra
École Normale Supérieure de Lyon

October 2022

Contents

1 Theoretical Approach

In this section, we make a first purely theoretical approach of whats is a DAO and properties of liquid democracies. STYX is built upon this approach.

1.1 Common goods

In economy, we use two properties to classify goods. The first one is called *excludability* and characterizes the capacity to exclude people from using a certain type of good. The second one is called *rivalry* and expresses the fact that the consumption of a good can reduce the ability of another party to also consume it.

In the case of a DAO, the DAO holds some assets and spending them reduces the ability to use them again. Moreover, anyone with DAO tokens can participate in the DAO. Therefore, a DAO deals with a common goods: its assets are both rivalrous and non-excludable. To give another example, the set of all fishes in the ocean is also a common good.

Now that we know what kind of goods the DAO will deal with, the question we may ask is: what is the best way to deal with a common good ?

In her book : Governing the commons [3] the Nobel prized Elinor Ostrom identified eight principles at the center of stable common goods:

1. Clearly defined the group boundaries (and effective exclusion of external un-entitled parties) and the contents of the common pool resource;
2. The appropriation and provision of common resources that are adapted to local conditions;
3. Collective-choice arrangements that allow most resource appropriators to participate in the decision-making process;
4. Effective monitoring by monitors who are part of or accountable to the appropriators;
5. A scale of graduated sanctions for resource appropriators who violate community rules;
6. Mechanisms of conflict resolution that are cheap and of easy access;
7. Self-determination of the community recognized by higher-level authorities;
8. In the case of larger common-pool resources, organization in the form of multiple layers of nested enterprises, with small local CPRs at the base level.

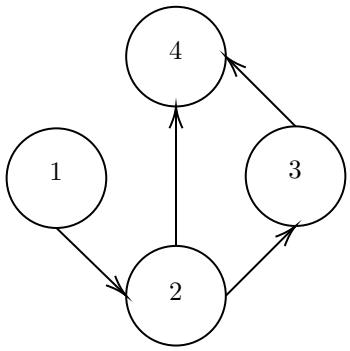

Figure 1: Graph representing a social network made of 4 voters

In the case of a DAO on a blockchain rules 7. and 8. don't really make sense. Moreover, one cannot breach rules because on a smart contract on a blokchain "code is law". We can therefore rely on CERBERUS [1] to make sure that there is no breach in the smart contract logic.

We therefore conclude that a good DAO should at least deal with principles 1-4 and 6.

1.2 Liquid Democracy

For the liquid democracy, our approach is based on the work of Kahng, Mackenzie and Procaccia [2]. We recall here their main results.

1.2.1 Definitions

We consider a *binary* vote on a decision and suppose that there is an objective correct decision. We let n be the number of voters and we label them with a *competence level* p_i, which is probability that they will vote for the correct decision. We suppose that voters both have a certain degree (see later for more precision) of knowledge of their competence level and that of other voters they know of. We represent a social network of n voters by a directed graph $G = (V, E)$, where each vertices in $V = \{1, ..., n\}$ represents a voter and every edge (i, j) represents the relation "i knows of j" (see fig. (1.2.1)).
We also suppose that there is a common threshold α such that if a voter knows of another having a competence level greater than theirs by α, then they will approve them. In mathematical term, a voter i will approve another voter j that they know of if $p_j \geq p_i + \alpha$. A voter also approves themselves. Voters only know if they approve a voter that they know of but a priori they cannot determine which one is the best.

The goal is to find an algorithm $\mathcal{A}$ that a voter should follow to decide, from the people they approve, to whom they should delegate their vote to. We finally define the notion of gain for a graph (social network) G of n voters and a voting

algorithm $\mathcal{A}$:

$$gain(\mathcal{A}, G) = P(\mathcal{A}, G) - P(\mathcal{D}, G),$$

where $\mathcal{D}$ is the algorithm where no one delegates their vote and $P(\mathcal{B}, G)$ is the probability of making the correct decision using a voting scheme $\mathcal{B}$ in the graph G.

To determine what is a great voting algorithm, we define two properties:

- *Positive Gain*(PG) : An algorithm $\mathcal{A}$ has the *positive gain* property if there exist a constant $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ there exist a graph G with n voters where $gain(\mathcal{A}, G)$ is positive.
- *Do No Harm*(DNH) : An algorithm $\mathcal{A}$ has the *do no harm* property if for all $\varepsilon > 0$, there exists a constant $n_1 \in \mathbb{N}$ such that for all graphs G with n voters ($n \geq n_1$), we have $gain(\mathcal{A}, G) \geq \varepsilon$.

Intuitively, the first property (positive gain) makes sure that if there is sufficiently many voters, there is a situation (graph) where the given algorithm is going to outperform direct voting. The second one makes sure that as the size of the network grows, the gain should shrink at least towards zero ie. the algorithm should tend to be not worse than direct voting.

The last notion to define is the notion of *local* and *global* voting algorithm. An algorithm is said *local* if, for a given voter v, the voting algorithm makes a decision based only on the knowledge of: (a) the voters v approves, (b) an arbitrary ranking over the voters that v approves, (c) the voters that v knows of.

To have a better grasp of this definition, let's give some examples of local voting algorithms:

- $\mathcal{D}$, the algorithm were voters don't delegate;
- Voters randomly delegates to a random approved voter;
- Voters delegate to a random voter that they approve if and only if there are more voters that they disapprove than voters that they approve;

Other examples of voting algorithms which are not local ie. are *global*:

- Voters delegate to their most competent approved voter (this is not an arbitrary ranking);
- Voters randomly delegate to one of their approved voters that has an even number of voters that they approve;

Now that everything has been defined, we can state the two important results from [2].

1.2.2 Local Voting Algorithms cannot both satisfy *DNH* and *PG*

The most important result from [2] is that a local voting algorithm cannot satisfy both the *Do No Harm* and the *Positive Gain* properties.

Theorem 1.1. *For any $\alpha_0 \in (0,1)$ such that a voter i approves a voter j if i knows of j and $p_j > p_i + \alpha_0$, there is no local voting algorithm that satisfies the PG and DNH properties.*

The proof if this theorem is a bit technical but to get somewhat of an intuition we will give some examples of what can happen.

For example, the algorithm $\mathcal{A}$ where voters delegates randomly to a voter that they approve of (not themselves if possible) is a local mechanism that cannot satisfy the *DNH* property. Consider the following situation:

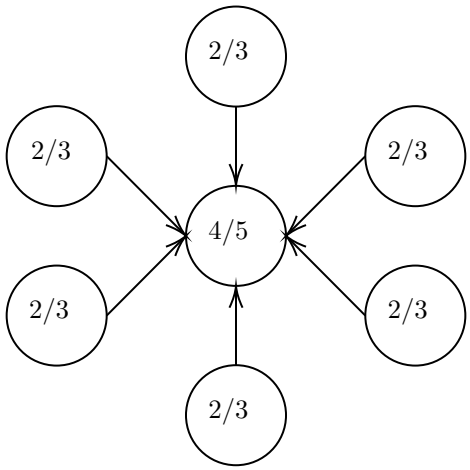

Figure 2: Example 1 - Every voters, with competence $\frac{2}{3}$, knows of one single voter of competence $\frac{4}{5}$

Using algorithm $\mathcal{A}$ whatever the number of nodes of competence $\frac{2}{3}$ the probability of making the right decision will always be $\frac{4}{5}$ because the center node is the only one that will cast a vote. However, the probability that the direct voting algorithm will make the right decision goes to 1 when the number of nodes of competence $\frac{2}{3}$ increases. In facts, direct voting should lead, in expectancy, $\frac{2}{3}$ of correct votes and the actual number of votes should be close to that when the number of nodes increases. Therefore, because the chosen decision is made by the majority of votes, the probability of making the right decision should be close to 1. Therefore, $gain(\mathcal{A}, G)$ converges to $-\frac{1}{5}$ and $\mathcal{A}$ cannot satisfy the *Do No Harm* property.

1.2.3 A global voting algorithm that satisfies both *DNH* and *PG*

In the last section, we saw that no *local* voting algorithm could satisfy both the *PG* and *DNH* properties. The natural question to aks is: what about *global* voting algorithms?

Fortunately, [2] prove that the following algorithm works :

Algorithm 1: GreedyCap

Data: G: graph with n voters, A: list of approved voters, $C : \mathbb{N} \to \mathbb{N}$
Result: a: to whom to delegate

1. $a \leftarrow A[0]$ ($A[0]$ is the voter themself)
2. **for** $b \in A$ **do**
   $n_b \leftarrow$ number of votes of b
   if $n_b > n_a$ *and* $n_b < C(n)$ **then**
      $a \leftarrow b$

The *GreedyCap* algorithm basically chooses the approved voters which has the most number of votes with the condition that their number of votes should not exceed a given cap C.

Theorem 1.2. *GreedyCap algorithm with a cap* $C : \mathbb{N} \leftarrow \mathbb{N}$ *such that* $\lim_{n\to+\infty} C(n) = +\infty$ *and* $C(n) \in o(\sqrt{\ln n})$ *satisfies the PG and DNH properties.*

For more intuition on how the Cap function should grow, we plot the function $\sqrt{\ln n}$ in fig. (1.2.3).

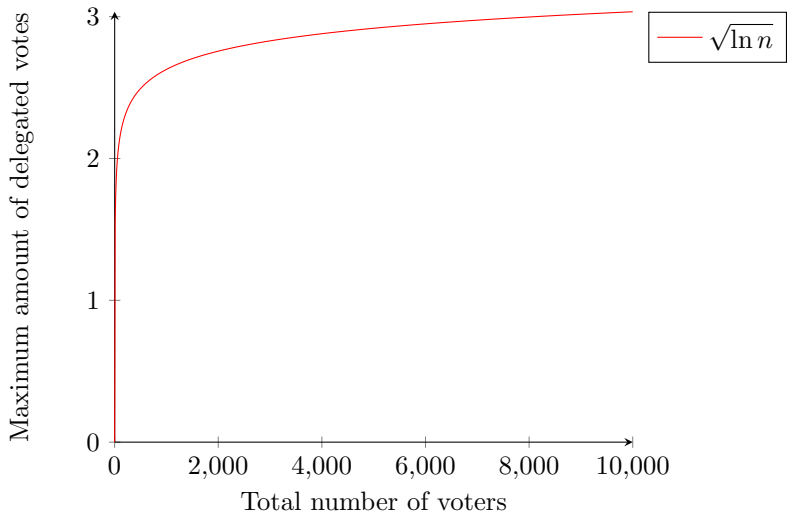

Figure 3: Plot of the function $\sqrt{ln}$

2 Styx

The previous theoretical approach enabled us to understand how to make a proper DAO. We learnt that we needed to make sure to implement some principles regarding common goods and that we should implement some sort of algorithm for the liquid staking part.

Styx enables a group of person to manage some assets as a community in a DAO way. It can also be used by another protocol as a DAO entity.

2.1 Common goods management

We explain here how Styx implements the principles of Elinor Ostrom.

1. Boundaries: We should clearly define what are the resources and who are the members.

   In Styx, the resources are the assets under management. To be a member of the DAO, one has to own some Styx tokens and lock them in the DAO blueprint to get a VoterCard. VoterCards act as Soulbound Tokens and cannot be traded. Locked tokens gain value with time to limit members changes.

2. Resource Appropriation: We should define how members can use the assets.

   In Styx, members can vote to spend assets that are managed by the DAO.

3. Collective-choice arrangement: We should clearly define how members participate to the DAO.

   In Styx, members can vote using their VoterCard and the voting power associated to the DAO tokens they have locked on it (more details in 2.2).

4. Monitoring: monitors (possibly members) should be able to monitor the DAO.

   In Styx, the instantiator of the blueprint has extra control thanks to an admin badge. Users can also make proposal to monitor the DAO.

5. Graduated Sanctions: we should implement sanctions for users that breach the rules of the DAO.

   We rely on Cerberus [1] to make sure that the code logic of the blueprint is respected.

6. Conflict resolution: We should implement a way to solve conflicts

   In Styx, every decision is made through a vote. Therefore, conflicts are solved by voting.

7. Not Applicable.

8. Not Applicable.

For more details on how Styx implements these principles, see 3.

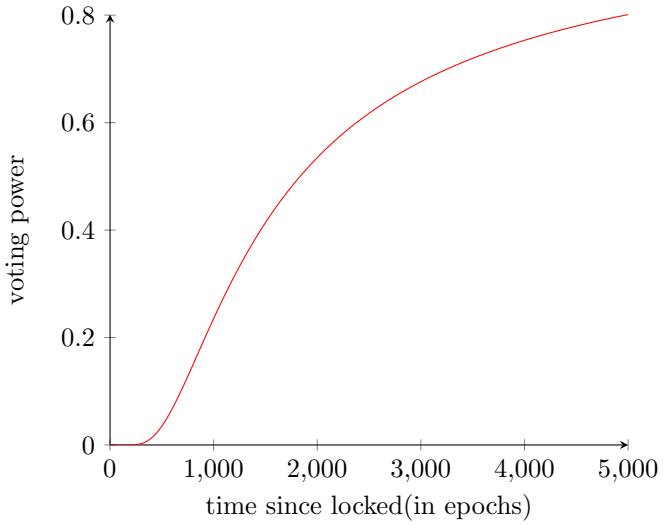

Figure 4: Voting power of token since it was locked

2.2 Voting Power

In Styx, the number of tokens locked on a VoterCard is not the same thing as voting power. We describe here how Styx assigns a *voting power* to a VoterCard.

2.2.1 Time Value

Because DAO tokens are tradable, defining clearly who is or who is not a member of a DAO is difficult. Moreover, we don't want people using freshly bought tokens to vote in a crucial Proposal. Therefore, to limit the impact of traders on the DAO, tokens locked on a voter card gain voting power with time. Ideally, we would like a token to take some time to be valuable and then have a voting power increasing up to one.This is why we use the function:

$$\text{th}(-\frac{a}{t}) + 1 = \frac{\exp(-\frac{a}{t}) - 1}{\exp(-\frac{a}{t}) + 1} + 1,$$

where a is a constant and t the time when the tokens were locked. Styx uses $a = 2016$. Because t is in epochs, it means that a is approximately equal to 3 months. We plot this function in fig (2.2.1). When a member locks tokens at different times, their VoterCard keeps track of it and sums the voting power of each group of tokens.

2.2.2 Real-life implementation of the GreedyCap algorithm

Our prior study showed that if we want liquid democracy to be efficient, we needed the voters to implement some voting mechanism. In real-life, choosing

how voters should delegate with an algorithm seems a bit too autocratic. Still, the way people vote is close to the core of the *GreedyCap* algorithm.
In facts, we can assume that most people will want to delegate to the most competent voter they approve. We can also assume that people tend to associate approval with competence. Therefore, in practice, people are likely to delegate to the voter they approve that has the most votes.

Therefore, if we want to respect the *GreedyCap* algorithm, we need to make sure that the total voting power of an individual does not exceed $C(tot)$, where tot is the total amount of tokens in circulation and C such that $\lim_{n\to+\infty} C(n) = +\infty$ and $C(n) \in o(\sqrt{\ln n})$.

We choose C in the following way:

$$C(x) = 2\sqrt[3]{\ln(x)}.$$

C trivially verifies the required properties. And we compute the user's final voting power F from its total voting power T (counting delegated voting power) in the following way:

$$F = \begin{cases} T & \text{if } T \leq \sqrt[3]{\ln(tot)} \\ \sqrt[3]{\ln(tot)} + \sqrt[3]{\ln\left(TVP - \sqrt[3]{\ln(tot)}\right)} & \text{otherwise} \end{cases}$$

3 Implementation

Styx is composed of five modules: *StyxDAO, BallotBox, Proposal, VoterCard* and *DecimalMaths*. *StyxDAO* is the main module and the blueprint with which users will interact whereas *BallotBox, Proposal, VoterCard* and *DecimalMaths* are utilitary submodules.

StyxDAO can hold a variety of assets and manages a DAO token called Styx. Users can make proposals to spend some amount of assets, mint new tokens or make changes to the voting system.

A Proposal goes through different phases described in fig.(5) and if it is accepted, the list of proposed changes are implemented. Users can support or vote for a Proposal using a VoterCard Non Fungible Resource. To do so, they first have to lock some tokens on the StyxDAO blueprint, which will be recorded on their VoterCard. Locked tokens gain power with time (see 2.2). By default, suggestion and voting phases last 168 epochs which is approximately a week. A Proposal also needs to be supported by 0.15% of total voting power to go through the voting phase and there is no minimum vote threshold. The original 0.15% support is based on the situation in the UK: a petition reaching 100k signatures will be debated in parliament [4]. To deal with Proposals, users interact with the StyxDAO blueprint which acts as an interface of the BallotBox module.

A BallotBox manages Proposals and make sure they move forward in the appropriate way. When a Proposal has reached its period expiration, users can make the proposal advance in its next phase by calling the blueprint. The BallotBox also makes sure that during the vote no one has too much power (see 2.2) using functions implemented in the module DecimalMaths.

Finally, the module DeicmalMaths implements the following operations for the Decimal type: $\exp(x), \ln(x)$and $\sqrt[3]{x}$.

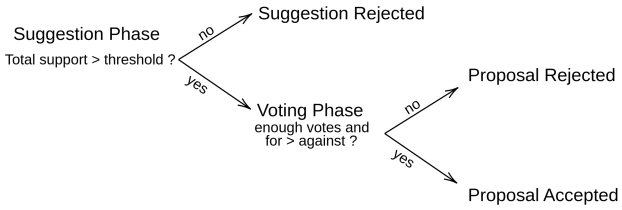

Figure 5: Proposal advancement diagram

4 Interactions with blueprint

We describe here how each modules interact with each other when a call is made to the StyxDAO blueprint.

4.1 Instantiation

When a user or a blueprint instantiates a StyxDAO Component, the Component returns its address and a Bucket containing a Badge that allows to emit or withdraw some DAO tokens. The instantiator can also instantiate the blueprint and give a admin badge to which to give such permissions.
This admin badge enables the blueprint to work side by side with other blueprints. For example, a Decentralised Exchange blueprint can instantiate a new StyxDAO blueprint and use the admin badge to mint new tokens and reward liquidity providers.
An internal admin badge is also created to enable the blueprint to mint DAO tokens, mint and update VoterCards .

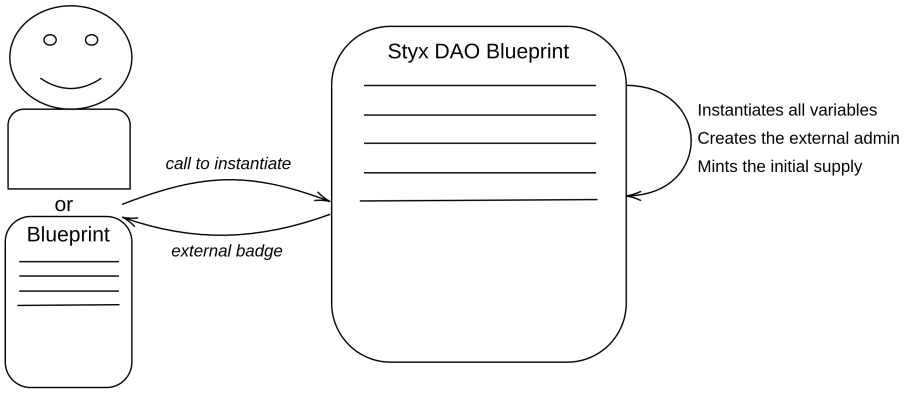

Figure 6: Instantiation of the blueprint

4.2 Minting a VoterCard

When a DAO token holder wants to register as a user of the DAO, they have to lock an initial deposit to the blueprint, which mints a new VoterCard NFT.

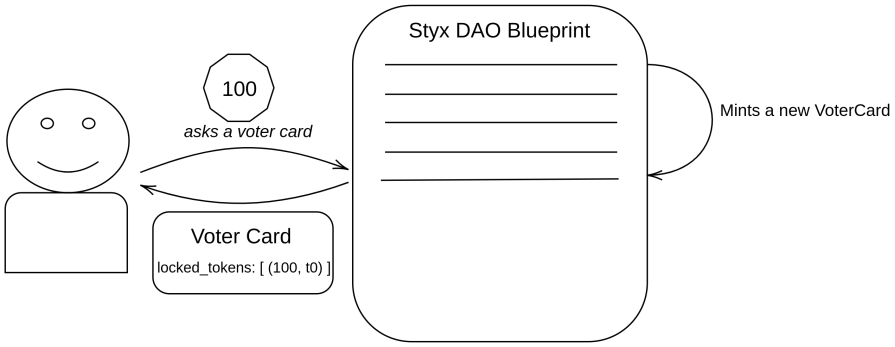

Figure 7: Minting a VoterCard

4.3 Locking tokens

When a user wants to lock new tokens, they have to provide a Proof of their VoterCard and a Bucket containing some Styx tokens to lock.

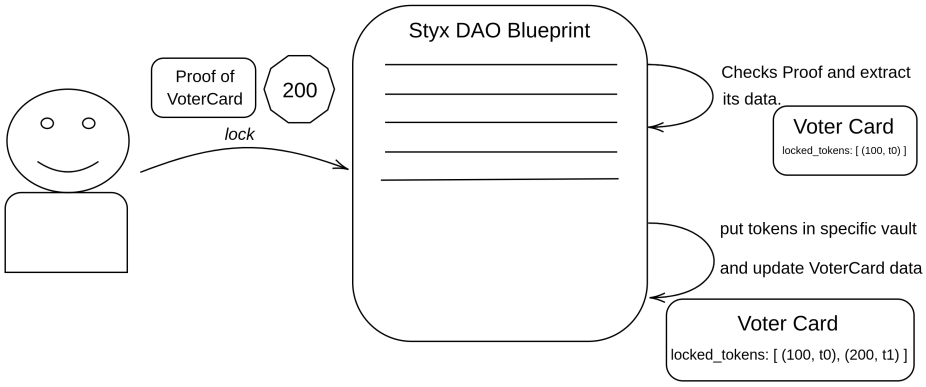

Figure 8: Locking tokens

4.4 Interactions with Proposals

Users can interact with a Proposal in the following ways:

- Make a proposal by giving a description, a list of proposed changes and a Proof of a VoterCard;
- Support a proposal by providing a Proof of a VoterCard;
- Delegate locked tokens for a proposal by providing a Proof of a VoterCard;
- Vote for a proposal by providing a Proof of a VoterCard;

- Try to advance with a proposal;

The general pattern when interacting (except for advancing with a proposal, which we will explain in the next section) is described in the following diagram. The StyxDAO blueprint acts as an interface of the BallotBox module.

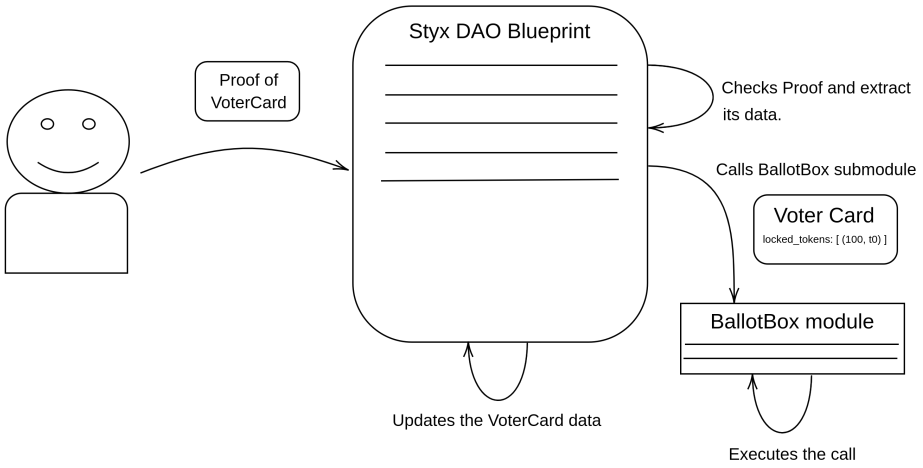

Figure 9: Locking tokens

4.5 Proposals

A Proposal can make the following changes :

- Change the suggestion phase period length;
- Change the voting phase period length;
- Change the support threshold needed in the suggestion phase;
- Change the minimum amount of votes needed to consider a vote valid;
- Spend some of the assets under management;
- Mint new DAO tokens;

Because Scrypto is an asset-oriented language, one cannot simply send assets to an address. Therefore, to still be able to let the blueprint "spend" some tokens, it stores a HashMap mapping VoterCard ids to another Hashmap mapping ResourceAddresses to the amount that the id can claim. Therefore, when the DAO is allowed to send assets to a user, it stores this as an amount of assets the user can claim. The user can then claim theses assets by calling the contract.

It is also important to note that changes to the voting system only impact proposals that are made after the change.

If a proposal is in voting phase, has reached its expiration time, has enough votes and has more votes for than against, a user can make it advance to the the *Proposal Accepted* phase and the blueprint will execute the changes.

References

[1] Florian Cäsar, Daniel P Hughes, Josh Primero, and Stephen J Thornton. A parallelized bft consensus protocol for radix. 2020.

[2] Anson Kahng, Simon Mackenzie, and Ariel Procaccia. Liquid democracy: An algorithmic perspective. *Journal of Artificial Intelligence Research*, 70:1223–1252, 2021.

[3] Elinor Ostrom. *Governing the commons: The evolution of institutions for collective action*. Cambridge university press, 1990.

[4] UK Parliament. Debate on petitions. https://petition.parliament.uk/. Accessed: 2022-10-07.