# Liquid Democracy: A comparison of delegation mechanisms in a misinformed and communicative world.

Bachelor's Project Mathematics

June 2024

Student: L. Nieuwenhout

First supervisor: Prof. J. Peypouquet

Second assessor: Prof.dr.ir. B. Besselink

## Contents

**Abstract**

Liquid democracy is often seen as the middle ground between representative democracy and direct democracy. We will look at ways to model different delegation mechanisms for liquid democracy and devise a way to deal with delegation cycles. The importance of misinformation and the ability of voters influencing each other over time is taken into account, the latter through a process called opinion diffusion. Using this model we examine which delegation mechanisms, if any at all, can result in a better voting outcome compared to a direct democracy.

## 1 Introduction

In 2024 democracy is the most common form of government on a national level[9]. Most of these democracies are so-called representative democracies. Once every couple of years, depending on the country, there will be national elections where every citizen can vote for a political representative belonging to a political party. This representative, when elected, is then supposed to fight for the wants and needs of the people who voted for them. Opposite to representative democracy stands direct democracy, which is as the name implies, a more direct form of democracy. Within this version of democracy citizens themselves have the ability to vote on legislation, instead of having to vote for representatives who then cast their vote on the legislation. An occurence of a push toward more direct democracy is currently happening in the Netherlands. During the last two national elections, some political parties were advocating for more binding referenda i.e. more direct democracy[12][11].

But there exist alternatives that can bridge the gap between representative democracy and direct democracy. Liquid democracy is one of these alternatives. This is a form of democracy that uses direct democracy as its basis, hence all voters are allowed to vote directly. In addition to this, a voter can also delegate their vote to a proxy. This proxy is a voter too, and can also decide to either delegate or vote directly. This system has been applied by the German Pirate Party who have used it for decision-making within the party itself[7]. It has also been used by Google, which implemented it in an app where employees could vote or delegate on certain decisions within the company[3].

Besides a call for direct democracy by certain political parties and speakers, other developments such as the spread of misinformation have been on the rise as well. This misinformation has mainly been employed in an effort to sway the opinions of voters on parties who participate in the elections. Such misinformation campaigns have been reported during the 2016 USA elections, the 2017 French elections and the 2019 Indian elections, but is not limited to just these three elections. One form of this kind of misinformation is fake news which uses untrue statements in order to rile up an audience. Although fake news generally has a short lifespan, by the time it has been verified as misinformation and discredited, it can already have reached its goal[8].

This misinformation can just as well be spread by word of mouth, either online or physically, and be used within echo chambers to strengthen already established beliefs[8]. This way of information spread is one that is difficult put into numbers, but may be one of the most effective ways to spread this information.

This paper was initially inspired by the work of Becker et al.[1] who considered the case where voter probabilities could be lower than 0.5, representing a voter to be misinformed. This stands out because most other research that has been done on liquid democracy only considered voter probabilities greater or equal to 0.5. In addition to allowing misinformation in their simulations, Becker et al.[1] looked at different network structures and deterministic delegation mechanisms such as the capped delegation, where a delegate is not allowed to obtain more than a capped number of votes. With these three parts they try to solve the optimal delegation problem, and compare their results to a direct democracy.

Two other studies, one by Kahng, MacKenzie and Procaccia[4] and the other by Campbell et al.[3], also discuss liquid democracy. Both of these studies make one big assumption which is that all delegation mechanisms used are created in such a way that delegation cycles cannot occur. This is a common trend among research into liquid democracy, yet is a deterministic way of looking at things, leaving voters out of control of their delegation choices.

## 2 Liquid Democracy

What exactly is liquid democracy? This is a question that can have many different answers according to whom you ask. Many researchers have come up with their own definitions and requirements for something to be allowed to be called a liquid democracy. Due to this issue, we will use the definition put forward by Valsangiacomo[10], who tried to create a single well-constructed definition of liquid democracy and set its basic requirements. These requirements are first: Delegations need to be transitive, and secondly: Delegations need to be voluntary.

Transitivity of voting implies that if we have three voters A, B and C, if A delegates his vote to B and B delegates its vote to C, then A's vote also gets delegated to voter C through voter B. One way of looking at this is that after voter A delegated his vote to voter B, B has both his own vote and A's vote to either delegate or directly vote with. But then if voter B also chooses to delegate his vote, in this case to voter C, he must delegate both the vote that he started with and the one he received from voter A.

Voluntary delegation implies that if a voter delegates their vote, he has full agency to retract their delegation whenever their vote gets delegated to a person they do not approve of. In addition to this, a voter should not be forced by an institution, entity or deterministic process to delegate in a particular way. To use the same example that we used for the transitive delegation case, A's vote ends up at voter C. But if voter A does not approve of voter C, then voter A is allowed to retract their vote and either redelegate their vote to someone new or vote directly.

The voluntary delegation requirement also implies that looking for optimal delegation results within liquid democracy and steering to such solution should not be allowed. This is because such algorithms and methods rely on deterministic non-local delegation mechanisms which force voters to vote along a set path[4][1], and thus the voter loses their agency which violates the voluntary delegation principle.

## 3 Problem Statement

In much of the current research on liquid democracy, only the case where a voter's probability of picking the "correct" option is at least 0.5 is considered. In addition to this, they usually also only consider optimal delegations. For example in the paper by Becker et al[1] they do consider a probability lower than 0.5, yet are only concerned about the optimal outcomes of delegations. They even state that although their outcomes are the optimal ones, there might not actually exist a delegation mechanism that can reach these outcomes.

To more accurately give a realistic description of the real world, we will be considering the case where voters can be misinformed and also use non-optimized delegation mechanisms; e.g. a voter will only base their decision to delegate on their direct neighbourhood instead of considering the entire population. Important to notice is that society isn't static and that people can influence each other's opinions. We will model the misinformedness by letting people have probabilities between 0 and 1 where a probability of 0.5 implies that a voter is equally as misinformed as it is informed. It is also important to model how people can influence each other's opinions. Through this we can discuss if an outcome of a vote done according to liquid democracy changes if people have had time to discuss the topic at hand. We have chosen to model these discussions through so-called "opinion diffusion" for which we use the Hegselmann-Kreuse model. The workings of this model will be discussed in section 4.2.

Furthermore, we need our model to be in line with our definition of liquid democracy. This means we need to account for voluntary delegation and transitivity. Transitivity is built into our choices of delegation mechanisms so that will not be an issue. Voluntary delegation will make the model more complex and we will tackle this complexity in two ways: First we use a method where after the delegation process is finished, if a voter is not happy with who got to delegate their vote, they can retract it. Secondly we will use a viscous voting system where votes will not be retracted, but instead, the impact of a voter's delegated vote will diminish the longer the chain of delegations becomes.

Finally, we will use python to simulate liquid democracy according to all the criteria discussed above. In addition to simulating liquid democracy, we will also simulate a direct democracy, so that we can compare the both results with each other. Through this comparison, within our model, we can see if a liquid democracy is more likely to give a preferable voting result compared to a direct democracy.

# 4 Preliminaries

## 4.1 Contact Graphs

To be able to model liquid democracy we need both a vector showing the informedness of each voter and a graph structure on how these voters are connected to each other. Both undirected and directed graphs will be used, but in this section, we only discuss the undirected graphs. If we have $n$ voters, then we need a vector of length $n$ called $\boldsymbol{x}$ where each $x_i$ represents how informed or misinformed a voter is. Then we need to know for each voter $i$ with which other voters $j$ they are in contact with. We define "being in contact with" as a binary relation. If voter $i$ is in contact with voter $j$, then voter $j$ is also in contact with voter $i$. We assume that voters are always in contact with themselves. This means that the graph that represents the structure of all contact links is shown as a self-connected undirected graph. Knowing the structure of the graph we can construct the adjacency matrix.

**Definition 1.** *An adjacency matrix $A$ is an $n \times n$ matrix where each $A_{ij} = 1$ if $i$ is connected to $j$ and $A_{ij} = 0$ if $i$ is not connected to $j$.*

Definition 1 implies that the adjacency matrix for the contact graph will be symmetric as the binary relation tells us that $A_{ij} = A_{ji}$ and each $A_{ii} = 1$ as each voter $i$ is in contact with themselves.

**Definition 2.** *The outdegree $Out(i)$ represents the number of connections from $i$ to any $j$ and the indegree $In(i)$ is the number of connections from any $j$ to $i$. They can be computed from the adjacency matrix $A$ by taking:*

$$Out(i) = \sum_{j=0}^{n} A_{ij}$$

$$In(i) = \sum_{j=0}^{n} A_{ji}$$

Because $A_{ij} = A_{ji}$ is a property of our adjacency matrix we have as a consequence that $Out(i) = In(i)$ is true for all $i$. This stops being the case when we will be looking at delegation matrices in section 4.3. It is also important to grasp that the rows of the adjacency matrix represent all outgoing connections of a voter and the columns represent all incoming connections of a voter.

## 4.2 Opinion Diffusion

We want to give a somewhat realistic view of how liquid democracy works in the real world through our simulations. To do this we need to take into account that voters have the ability to change each other's opinion. Let's say that voter $i$ is in contact with voter $j$ then it is likely that they, either knowingly or unknowingly, can change each others opinion on the topic at hand. To be able to simulate this we will use the agent-based Hegselmann-Krause model with heterogenous bounds. Before we will discuss this model we have to define what a confidence bound and an interval confidence set are.

**Definition 3.** ***Confidence bound***
*Let $\boldsymbol{x}(t)$ be the $n$-vector of voter probabilities after $t$ diffusion steps. Define $\epsilon_i(t)$ as the confidence bound of voter $i$ with $\epsilon_i = 0.5 - |0.5 - x_i(t)|$. Then $\epsilon(t)$ is the $n$-vector consisting of confidence bounds of each $i$ at diffusion step $t$.*

We choose $\epsilon_i(t)$ in this manner to simulate how voters who are more unequally (mis)informed (e.g. either more informed or more misinformed and thus have a probability further from 0.5) are less likely to be influenced by different opinions. In a way they are more headstrong in their beliefs and are only nudged by people somewhat equally (mis)informed as them. Meanwhile people who are about equally as informed as misinformed, and thusly have probabilities closer to 0.5, are more likely to be influenced by people further removed from 0.5.

Next up we want to know for all $i$, which voters $j, k, l, ...$ have probabilities within the confidence interval $\epsilon_i$ centered around $i$'s probability $x_i$. For this we define the following set for each $i$:

**Definition 4.** ***Interval confidence set***
*Let $x_i(t) \in \boldsymbol{x}(t)$ and let $\epsilon_i(t)$ be its corresponding confidence bound. Define $I_\epsilon(i, \boldsymbol{x}, t)$ to be the interval confidence set consisting of all $j$ such that $|x_j(t) - x_i(t)| \leq \epsilon_i(t)$. Formally:*

$$I_\epsilon(i, \boldsymbol{x}, t) := \{j \in \mathbb{Z}_{\geq 0} : |x_j - x_i| \leq \epsilon_i\}$$

Using definitions 3 and 4 we can start constructing the diffusion matrix $C$ and define the HK model:

**Definition 5.** ***HK-model***
*Let $\boldsymbol{x}(t)$ be the vector of voter probabilities at diffusion stage $t$ and $\epsilon(t)$ the vector of all confidence bounds of $\boldsymbol{x}(t)$ at diffusion stage $t$. Then we construct the diffusion matrix $C$ in the following manner:*

$$C_{ij} = \begin{cases} 0 & \text{if } j \notin I_\epsilon(i, \boldsymbol{x}, t) \\ \frac{1}{\# I_\epsilon(i, \boldsymbol{x}, t)} & \text{if } j \in I_\epsilon(i, \boldsymbol{x}, t) \end{cases}$$

*and multiplying our diffusion matrix $C$ with probability vector $\boldsymbol{x}(t)$ lets us obtain the diffused probability vector $\boldsymbol{x}(t+1)$, in short: $C\boldsymbol{x}(t) = \boldsymbol{x}(t+1)$*

During the diffusion process we update the values of each $x_i(t)$ to the value $x_i(t+1)$ by letting $x_i(t+1)$ be the mean of all $x_j(t)$ within the confidence interval centered around $x_i(t)$. For the next step we then need to create another confidence bound vector $\epsilon(t+1)$ and a new interval confidence set $I_\epsilon(i, \boldsymbol{x}, t+1)$.

Because we are taking the mean averages for each step, the values of $\boldsymbol{x}(t)$ will most likely converge to a steady state after a certain number of steps $t$. We say most likely here because there is no proof that confirms this, but extensive numerical tests give reason for this to be true. In the case of homogeneous bounds, there is a proof of convergence[5]. This convergence represents the forming of a consensus within subsets of the voting population because people will no longer be able to change each other's opinions.

## 4.3 Delegations and Cycles

Now that we have discussed adjacency matrices and opinion diffusion we can start talking about delegation mechanisms. A delegation mechanism is nothing more than the conceptualization of a specific decision-making process. After conceptualizing this delegation mechanism, we then model this decision-making process by creating a delegation function that will output a delegation matrix, depending the input data. Within this paper, we will only consider delegation mechanisms for which the delegation function outputs a delegation graph where every vertex $\alpha_i$ has an outdegree $Out(\alpha_i)$=1. An example scenario:
Alice, Bob, Charlie, and Dave all want to have a healthy dinner together and Alice, Bob, and Charlie know each other well with Charlie bringing Dave along as a guest. Dave only knows Charlie, and Alice and Bob don't know Dave either. Now unbeknownst to Alice and Bob, Dave is a health coach, while they themselves don't know much about eating healthily. Alice and Bob do know that Charlie is at least somewhat knowledgeable about healthy food. Now to decide on what to make for dinner Alice and Bob both choose to let Charlie decide on what to have for dinner, yet Charlie has chosen Dave. Transitively Alice- and Bob's votes end up at Dave through Charlie. This is just a specific outcome of a delegation function being applied to a population, with the population consisting of Alice, Bob, Charlie, and Dave. To be able to try and model this we will do the following:

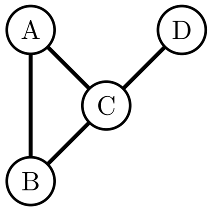
Let $A$ be the adjacency matrix, $\boldsymbol{x}$ be the probability vector of choosing a healthy dinner where $x_1, x_2, x_3$, and $x_4$ respectively represent Alice's, Bob's, Charlie's, and Dave's probabilities. We let $d(A, \boldsymbol{x})$ be the delegation function defined by the decision making process of choosing the highest adjacent probability. Then after conjuring up some values for $x_i$ we get:

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

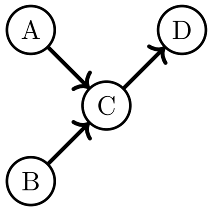
$$\boldsymbol{x} = \begin{pmatrix} 0.25 \\ 0.45 \\ 0.70 \\ 0.93 \end{pmatrix}$$

$$d(A, \boldsymbol{x}) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

With the graphs from before and after applying the delegation function:

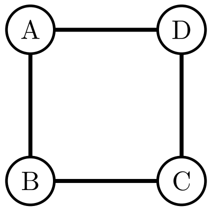


*Contact graph*



*Delegation graph*

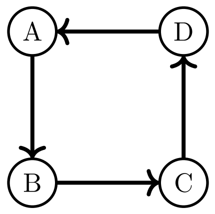
Using a delegation mechanism where voters delegate at random we define a different scenario: A is only in contact with B and D, B is only in contact with A and C, C is only in contact with B and D and D is only in contact with A and C. This gives the following graph and adjacency matrix:



*Contact Graph*

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Then if we apply the delegation function $d_{random}(A, \boldsymbol{x})$ we cannot exclude the possibility of obtaining the following delegation graph and corresponding delegation matrix $D$:



*Delegation Graph*

$$D = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Within this example, we see that we have obtained a cycle! We want to avoid this when trying to model liquid democracy, because within cycles no actual voting occurs. All the votes that enter the cycle will be stuck in there which might skew the results if we just simply exclude these votes after we detect the cycle. To solve this problem we create a method to both detect and deal with cycles within delegations. It is very important that the implementation of this will not, or at least try very hard not to, violate the voluntary delegation principle discussed in section 1. Before we can do this, we first need to be sure that any vertex ending in a guru (a person who votes and does not delegate) cannot be part of a cycle.

**Definition 6.** ***Delegation Cycle***
*Let $d$ be a delegation function and $\alpha$ a subgraph of the contact graph $A$ such that $Out(\alpha_i) = 1$ for all $\alpha_i \in \alpha$. If for some $n \geq 2$ we have $d^n(\alpha_i) = \alpha_i$ then $\alpha_i$ is part of a delegation cycle for the delegation function $d$. If for all $\alpha_i \in \alpha$ this property applies, then $\alpha$ is a delegation cycle.*

**Corollary 4.0.1.** *A delegation cycle $\alpha$ has length $k$ if and only if $k$ is the minimal $k \geq 2$ such that $d^k(\alpha_i) = \alpha_i$ for all $\alpha_i \in \alpha$.*

*Proof.* If $k \geq 2$ is the minimal $k$ such that $d^k(\alpha_i) = \alpha_i$ for all $\alpha_i \in \alpha$ then $\alpha$ is a delegation cycle by definition.

If $\alpha$ is a delegation cycle of length $k$ under $d$, then $\alpha = \{\alpha_1, ..., \alpha_k\}$. If $t < k$ exists such that $d^t(\alpha_i) = \alpha_i$ then only a subset of $\alpha$ can be reached through repeated application of the function $d$. For example, $\alpha_1$ will only range from $\alpha_1$ through $\alpha_{t-1}$ after an arbitrary number of applications of $d$. This means the elements $\alpha_t, ..., \alpha_k$ are skipped. Thus we have arrived at a contradiction, proving that $k$ is the minimal $k \geq 2$ such that $d^k(\alpha_i) = \alpha_i$. □

**Theorem 4.1.** *Let $d$ be a delegation function such that $Out(\alpha_i) = 1$ for all $i \in \{0, ..., n\}$ and let $d(\alpha^*) = \alpha^*$ and $d^n(\alpha_i) = \alpha^*$ for some $n, i \in \mathbb{Z}_{\geq 0}$. Then $\alpha_i$ is not part of a cycle.*

*Proof.* proof by induction:
Case $n$=1: Let $d(\alpha_0)$=$\alpha_0$, we see that $\alpha_0$ is a guru and because $Out(\alpha_0)$=1 it

cannot delegate to anyone other that itself. If $\alpha_1$ exists such that $d(\alpha_1) = \alpha_0$, then as $Out(\alpha_1)$=1 it can only form a cycle with $\alpha_0$. But we have $Out(\alpha_0) = 1$ and $d(\alpha_0) = \alpha_0$, thus $\alpha_0$ does not delegate back to $\alpha_1$ which means $\alpha_1$ and $\alpha_0$ are not in a cycle.

Case $n + 1$: Assume the theorem holds for $n$, then we have $d^i(\alpha_i) = \alpha_0$ for $i \in \{1, ..., n\}$ and none of the $\alpha_i$ are in a cycle. Then $d^{n+1}(\alpha_{n+1}) = \alpha_0$ and $d^{n+1}(\alpha_{n+1}) = d^n(d(\alpha_{n+1})) = d^n(\alpha_j)$ for an arbitrary $\alpha_j$ on a path of length $n$ towards $\alpha_0$. Thus by the same argument as for the case $n = 1$, $\alpha_{n+1}$ and $\alpha_j$ cannot be in a cycle together as both have an outdegree of 1. $\square$

Using theorem 4.1 we know that if some voter $i$ is a guru, then this voter cannot be part of a cycle. But also that any $j$ along a path that ends at a guru cannot be part of a cycle. Using this we can state a corollary to the theorem.

**Corollary 4.1.1.** *Let $\alpha_i$ be a vertex on a graph of finite size and $d$ be a delegation function such that $Out(\alpha_i) = 1$ for all $i$. If $d(\alpha_i) \neq \alpha_i$ and $d^k(\alpha_i) \neq \alpha_i$ for all $k \in \mathbb{Z}_{>1}$, then $\alpha_i$ is not in a cycle.*

*Proof.* $d(\alpha_i) \neq \alpha_i$ means $\alpha_i$ is not a guru. Let $n$ be the number of verteces in the graph. If for some $k < n - 2$ we have $d^k(\alpha_i) = d^{k+1}(\alpha_i)$ then we can apply theorem 4.1 and conclude that $\alpha_i$ is not part of a cycle.
If there is no such a $k$ then $\alpha_i$ is not connected to a guru, thus it is connected to a cycle instead. But $d^k(\alpha_i) \neq \alpha_i$ for all $k \in \mathbb{Z}_{>1}$, thus $\alpha_i$ is not within this delegation cycle. $\square$

With corollary 4.1.1 we can show that a vertex is not part of a cycle. This also means that if we adjust the cycle to no longer incorporate a certain vertex, while keeping the vertex to which it delegates the same, that it will be impossible for it to be in a delegation cycle.

**Corollary 4.1.2.** *Let $\alpha$ be a delegation cycle of length $n$ and have each $\alpha_i \in \alpha$ also be attached to arbitrary paths $\beta$, e.g. $In(\alpha_i) \geq 1$. If according to a uniform distribution we randomly change a $d(\alpha_i) = \alpha_j$ into $d(\alpha_i) = \alpha_k$ for some $\alpha_k \in \alpha$, we can iteratively reduce the length of the cycle until some $\alpha_i$ becomes a guru.*

*Proof.* Case 1:
Because the uniform random variable $X$ can be any $X \in \{0, ..., n\}$ if $d(\alpha_i) = \alpha_j$ is changed into $d(\alpha_i) = \alpha_{X=i}$ then $\alpha_i$ has become a guru and using theorem 4.1 all other $\alpha_j \in \alpha$ can no longer be in a cycle.

Case 2:
Let $X$ be a uniformly distributed random variable depending on $n$ where $n$ is the length of the cycle $\alpha$ and let $\alpha$ be indexed such that $d(\alpha_i) = \alpha_{i+1}$. W.l.o.g, we let $\alpha_0$ be the vertex we apply the delegation change to. If $d(\alpha_0) = \alpha_{X=0}$, refer back to case 1. If $d(\alpha_0) = \alpha_{X=i}$ for some $1 < i \leq n$ then the verteces $\alpha_1, ..., \alpha_{i-1}$ are no longer part of the cycle and instead are connected to a path $\beta$ connecting to $\alpha_i$. According to corollary 4.1.1 these verteces are no longer

part of a cycle. We now re-index our cycle $\alpha$ and repeat this process until $n = 2$, in which case we pick one of the two remaining verteces to be the new guru. Then we can apply theorem 4.1 and show that none of the $\alpha_i$ are in a cycle anymore. $\square$

Now we can construct an algorithm for finding cycles. For this, we use the fact that on the delegation matrix, the only nonzero diagonal elements will be gurus. Using theorem 4.1 we know that any path connected to a guru cannot be part of a cycle. From corollary 4.1.1 we see that if a voter sends his vote into a cycle, yet does not get it back again, the voter is not defined to be in the cycle. And finally using corollary 4.1.2 we can create an algorithm that guarantees the breakup of cycles.

To identify the cycles in a delegation matrix $D$ we first take the transpose of $D$ to make the operations easier to work within our python environment. Starting with the first entry of the set $\bar{n} = \{0, 1, ..., n-1\}$ we find the only nonzero entry in that column of $D^T$, which we will label as column $i$ and row $j$. We add the column number to an ordered set $\eta$ that gathers all the column indexes that are examined. The next step is to check column $j$ and find in which row its nonzero value is located. Again we add the column number to $\eta$ and repeat the entire process until either we reach a nonzero entry on the diagonal of our matrix D (a guru) or until we reach a column number which is in $\eta$. If we do get a repeating column as in the latter case, then we have found a cycle. We will then remove all entries of $\eta$ from $\bar{n}$. In case of a cycle we add the ordered values from $\eta$ into a master array $\bar{\eta}$, which is where we will store our cycles. Remove all entries within $\eta$. In the case that the first occurrence of the repeated value is not at the first entry of $\eta$ we only store the entries of $\eta$ starting from the first occurrence of the repeated number. For example if $\eta = \{1, 2, 3, 4, 5, 3\}$ we will only add $\{3, 4, 5, 3\}$ to $\bar{\eta}$. We keep repeating the algorithm until $\bar{n}$ is empty.

$\bar{\eta}$ now contains all cycles in the delegation graph, we just have to break these cycles up. For this, we use a technique proposed by Campbell et al.[3] where each member of a cycle randomly redistributes their vote to somebody within their approval range. Repeat these redistributions until either the cycles disappear or the cycle remains the same. In case the cycles remain stable, each member abstains from voting. In contrast to this technique by Campbell et al., we let a randomly selected member of the cycle become a guru. We choose for this option, because in our model each voter will always be in their approval range.

# 5 The Model

## 5.1 Initial Data

Before we can create our model, we need to have data. We will be working with 100 voters on an undirected random graph. This random graph will be our contact graph, with each vertex representing a voter $i$. Each voter $i$ will have a probability associated with it which both represents the probability of voting for the preferred option and represents how informed this voter is. Probabilities lower than 0.5 mean that the voter is more misinformed than informed and vice versa. These probabilities are generated from a truncated normal distribution between 0 and 1 with a mean $\mu \in \{0.5, 0.6, 0.7\}$ and standard deviation $\sigma = 0.25$. For each $\mu$ we generate the probability vector $\boldsymbol{x}$ 200 times and for each of the 200 $\boldsymbol{x}$'s we also generate a new random undirected graph. As an additional assumption, we assume that each voter accurately knows the probability of all other voters they are in contact with, including themselves.

## 5.2 Delegation Mechanisms

Five delegation mechanisms will be looked at during our simulations. First will be a direct democracy, where everybody votes for themselves and thus no delegation will take place. Second, come three variants of an approval-based delegation mechanism inspired by the paper by Markakis et al[6]. The final delegation mechanism is inspired by Campbell et al[3], which is an expert-based delegation mechanism.

The three delegation mechanisms inspired by Markakis et al will be `markakisrndm`, `markakisdetmax` and `markakisdetmin`. These are approval-based delegation mechanisms, which means that a voter $i$ will only consider delegating to a voter $j$ whenever $j$ is approved by $i$. We choose $j$ being approved by $i$ to mean that the probability of $j$ lies within the confidence interval around the probability of $i$. This confidence interval is the same confidence interval that we use for opinion diffusion to create our interval confidence sets. In other words, $i$ approves of $j$ whenever $x_j \in [x_i - \epsilon_i, x_i + \epsilon_i]$ where $\epsilon_i$ is the confidence bound of voter $i$ as defined in section 4.2.

Now that we know what it means for a voter $j$ to be approved by $i$ we can start defining all three Markakis-based delegation mechanisms. For all three of them, we first need to create sets $I_\epsilon(i, \boldsymbol{x}, t)$ of voters who are in contact with- and within the confidence bound of voter $i$. After we have these sets we can start to differentiate between `markakisrndm`, `markakisdetmin` and `markakisdetmax`. `markakisdetmin` and `markakisdetmax` act similarly but opposite. For `markakisdetmin` we let voter $i$ delegate to voter $j$ where voter $j$ is the voter which has the lowest probability $x_j$ of all voters within $I_\epsilon(i, \boldsymbol{x}, t)$. Likewise, `markakisdetmax` will let $i$ delegate to $j$ whenever $j$ is accompanied by the largest probability of all voters in $I_\epsilon(i, \boldsymbol{x}, t)$. The third delegation mechanism `markakisrndm` is one where voter

$i$ picks a uniformly distributed random $j$ from $I_\epsilon(i, \boldsymbol{x}, t)$ to delegate its vote to.

Both `markakisdetmin` and `markakisdetmax` represent edge-case scenarios that might occur within this approval-based system. `markakisdetmin` represents the worst-case scenario, while `markakisdetmax` represents the best-case scenario for each simulation. `markakisrndm` is a more realistic representation of how this kind of delegation mechanism would work, as it better reflects the unpredictability of human behaviour. We choose to let the approval be based on the confidence bound $\epsilon$ because it is a reasonable assumption that people are only willing to delegate to people who they are willing to be influenced by.

The final delegation mechanism that we use is `Campbell`. This delegation mechanism is not necessarily approval based, as it is more rigid. The entire voter base will first be split up into anti-experts, non-experts, and experts. An anti-expert is a voter who has a probability of less than or equal to 0.25. A non-expert is a voter who has a probability between but not including 0.25 and 0.75, and an expert has an assigned probability of 0.75 or higher[3]. After these distinctions have been made, the voter will uniformly randomly choose one of these groups and after choosing the group they will uniformly randomly choose a voter to delegate to. The groups that a voter can choose from is dependent on which group the voter belongs to. If $i$ is a non-expert, then they can choose from all three groups, yet if $i$ is an (anti-)expert, then $i$ can only pick between (anti-)experts and non-experts.

### 5.3 Two stages

Our elections will be a two-step voting process. First, all delegations will happen including the breaking of delegation cycles, and if the resulting delegation matrix $D$ contains no more cycles, then the first phase of the voting process has ended. The second phase is dedicated to the processing of delegations and the counting of the votes. We apply two different methods to do this, firstly an approval-based method and secondly we apply viscous democracy[2].

The approval-based method stems from the voluntary delegation principle that our definition of liquid democracy relies upon. This is manifested by the idea that a voter $i$ can retract their delegation whenever their delegation gets passed on, or voted with, by another voter $j$ who they disapprove of. In our model, voter $i$ can only disapprove of voter $j$ when $i$ is in contact with $j$ and $x_j$ does not lie within the confidence interval around $x_i$. The confidence interval is defined the same way as it is when applying opinion diffusion. This does mean that if a voter $j$ is delegated $i$'s vote, but $j$ is not in contact with voter $i$, then $i$ cannot disapprove of $j$ even though $x_j$ may lie outside of the confidence interval of $x_i$. This is because we assume that a voter $i$ cannot accurately know how informed or misinformed a voter is without being able to communicate with them. If all voters $i$ have checked their delegations and retracted them if needed, then the votes will be counted. This is done by multiplying the number of votes each guru has obtained with their probability and then dividing this by the population size to obtain a scalar probability.

Our alternative to the approval-based method is a method where no delegations will be retracted, called viscous democracy. This method also relies on the reluctance to delegate to people whom a voter may not approve of, but does this in a different way. When votes within a viscous democracy get delegated, the number of votes that are delegated will be multiplied by a scalar $a$ between 0 and 1. One of the reasons this model came to be was because in our current technological age being in contact with people solely through the internet has become more common than it used to be. This means that people maintaining so called "weak connections" has also become more common. Opposed to "strong connections" which can be family or close friends, these weak connections (distant friends, acquaintances or facebook friends) often outnumber the strong connections as online social networks allow for more of these weak connections to form. As a result, the average trust a voter has in their connections decreases when they have a large online social network[2]. This lower average trust in their connections means that there is a buildup of reluctance when it comes to delegating.

To implement this kind of reluctance we first need a cyclefree delegation matrix $D$. From this delegation matrix we can read of the gurus i.e. if $D_{ii} \neq 0$ then voter $i$ is a guru. As $D$ is just the adjacency matrix of the delegation graph, we can read off the paths that end at each guru $i$. With this information we can construct the following formula for the voting weight $w(i)$ of each guru $i$:

$$w(i) = \sum_{Gu(j)=i} a^{|p(i,j)|}$$

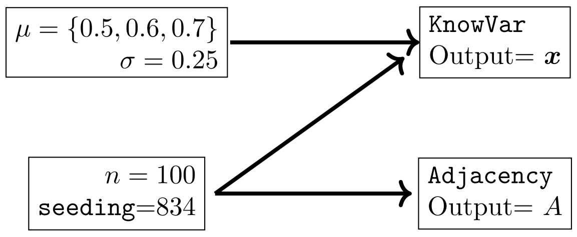
Where $Gu(j) = i$ means that the guru of voter $j$ is voter $i$ and $|p(i,j)|$ is the length of the path from voter $j$ to voter $i$ within the delegation graph. So from this, we can already see that the further a voter is located from their guru, the less of an impact their delegation will make on the final outcome. Now to find the probability $q_{viscous}$ we take the weighted sum of all probabilities of the gurus weighted by the gurus' voting weights and divide them by the total weight of all gurus:

$$q_{viscous} = \frac{\sum_i x_i w(i)}{\sum_i w(i)}$$

## 5.4 Implementation

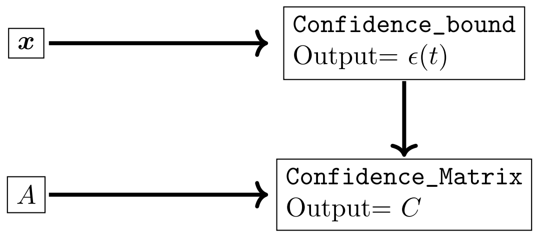
Firstly we use the function `KnowVar` to generate a probability vector $\boldsymbol{x}$ as discussed in section 5.1 and have the function `Adjacency` generate a random matrix $A$ of zeroes and ones which we symmetrize and put all diagonals to be equal to one. This $A$ will be the adjacency matrix of our contact graph. Because we use randomization, it is important that we fix a seed to guarantee reproducability. This fixed seed we call `seeding` which is just a scalar value, namely `seeding`=834 which we increase by one each time we have to generate a new probability vector $\boldsymbol{x}$. The generation of this new $\boldsymbol{x}$ only occurs after all the delegation mechanisms have been applied to it and we have found the outcome probabilities $q$. `KnowVar` has $n = 100$, $\mu$, $\sigma$ and `seeding` as inputs and has as output $\boldsymbol{x}$. The function `Adjacency` has as inputs $n = 100$ and `seeding` and as output the 100×100 matrix $A$.

**Initial Data**



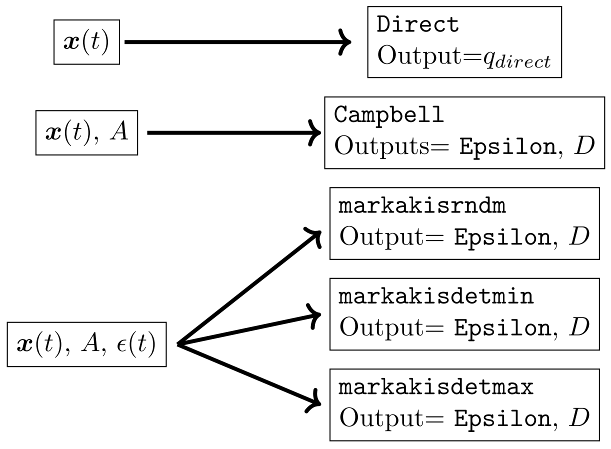
Now we can start to implement the diffusion stages. For our results, we are only interested in the (now time dependent) $\boldsymbol{x}(t)$ whenever $t \in \{0, 1, 5, 10, 20\}$. First we create our confidence bound $\epsilon(t)$ as described in section 4.2 using the function `Confidence_bound` which takes as input $\boldsymbol{x}(t)$. With this $\epsilon$ we can start creating the confidence matrix $C$ using the function `Confidence_Matrix` with inputs $A$, $\boldsymbol{x}(t)$ and $\epsilon(t)$. Within this function first, the index set $I_\epsilon$ gets created for each $i$, and then each row $C_i$ gets created as described in section 4.2. The outputs of `Confidence_Matrix` will just be the created confidence matrix $C$.

**Opinion Diffusion**



Thirdly we implement the five delegation mechanisms from section 5.2: `Direct`, `Campbell`, `markakisrndm`, `markakisdetmin` and `markakisdetmax`. `Direct` takes as input $\boldsymbol{x}(t)$ and outputs the scalar probability $q_{direct}$ by taking the mean of all $x_i(t)$. The other four will function as described in section 5.2. `Campbell` will take as inputs $\boldsymbol{x}(t)$, $A$ and `seeding` and outputs the matrix `Epsilon` where the only nonzero entries in the row $\texttt{Epsilon}_i$ are either all anti-experts, non-experts or experts in contact with voter $i$ depending on which group is chosen. The other output of `Campbell` is `Approv` which is the delegation matrix $D$. `markakisrndm` takes as inputs $\boldsymbol{x}(t)$, $\epsilon(t)$, $A$ and `seeding` and returns as output `Epsilon` which is different from the `Epsilon` outputed by `Campbell` in the sense that now the only nonzero entries in $\texttt{Epsilon}_i$ are the $\texttt{Epsilon}_{ij}$ where $x_j$ is within the confidence interval around $x_i$. Similarly to `Campbell`, `Approv` is the delegation matrix $D$. `Approv` will always be the delegation matrix for the specified delegation mechanism. Next up `markakisdetmin` and `markakisdetmax` both have $\boldsymbol{x}(t)$, $\epsilon(t)$ and $A$ as input and output the exact same `Epsilon` as for `markakisrndm` and output their respective `Approv` delegation matrices.

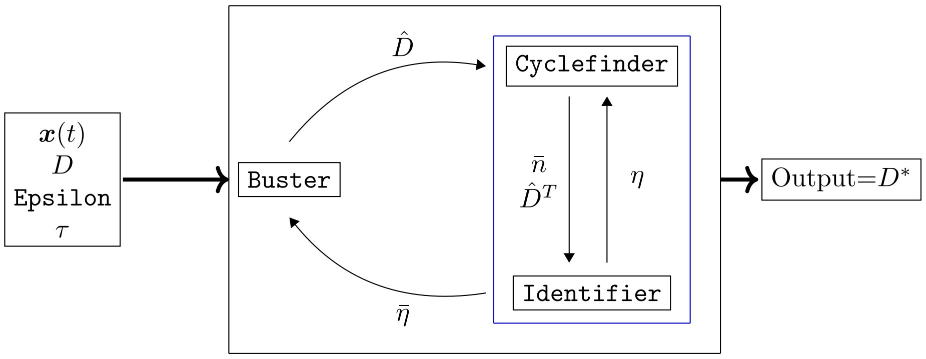
**Delegation mechanisms**



Fourthly we have the cycle breaking function `Buster` and path identifying function `Pathfinder`. The first one will be used for the breaking up of cycles and the second for outputting all unique paths leading to gurus. Starting with `Buster` we use the same algorithm as defined at the end of section 4.3, where we first identify all the cycles present in the delegation matrix $D$ (`Approv`) and secondly we redistribute the delegation within each cycle. To be able to identify the cycles we use the function `CycleFinder` which itself is reliant on the function `Identifier`. `Cyclefinder` only has $D$ as input and creates from this the set $\bar{n}$ which is then used as an input for `Identifier` along with $n$ and $D^T$. `Identifier` then outputs the ordered set $\eta$ as described in section 4.3 which is necessary for finding the ordered set of cycles $\bar{\eta}$ which is output by `Cyclefinder` as `Array`.

Now that we have found all the cycles and have stored them in `Array`, we can use that as an input for `Buster` which takes the inputs `Epsilon`, $D$, `seeding`, and a time variable $\tau$. From the formatting of `Array` we know that each cycle can be identified by their initial and final entry, which should be the same. Picking the first cycle $\alpha$ from `Array` we let each $\alpha_i$ redelegate their votes among one of all $\alpha_j$ for which $\alpha_j$ is represented by a nonzero entry in the row $\texttt{Epsilon}_{\alpha_i}$. We repeat this for each cycle present in `Array` and update our delegation matrix $D$ to now represent these redelegations. Now `Buster` outputs the updated delegation matrix $D^*$. If we apply `Buster` only once to our matrix $D$ then it may not be cycle free yet, but from corollary 4.1.2 we know that if we keep applying `Buster` to $D$ then it will eventually become cyclefree. To make sure that this happens in a timely fashion and to exclude edge cases such as cycles flipflopping, we increase $\tau$ each time `Buster` is applied to $D$. When $\tau$ reaches a value of 20, we force the cycles to break by having a member of each cycle become a guru. This does not violate the voluntary principle for all three markakis functions, as there a voter always approves of themselves. For the case of `Campbell` it can violate the voluntary principle whenever a voter has chosen to delegate to a group to whom they do not belong themselves. That is also why we only apply this method of cycle breaking when the cycles are still present after lots of iterations.

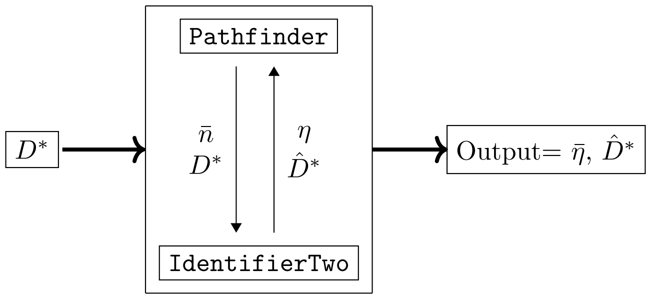
**Cycle Breaking**



After we have obtained the cyclefree delegation matrix $D^*$ as output from iteratively applying `Buster`, we use `Pathfinder` to find all unique paths ending in a guru. It does this by first running `Cyclefinder` again just to make sure $D^*$ is actually cyclefree. Then it employs the function `IdentifierTwo` which similarly to `Identifier` takes as input $\bar{n}$, $n$ and $D^*$. `IdentifierTwo` then also outputs $\eta$ but instead of $\eta$ containing cycles, it only contains paths because each path will end in an entry on the diagonal of $D^T$. In addition to $\eta$ it also outputs an augmented version of $(D^*)^T$, where all $D_{ij}^T \neq 0$ in case $i$ is a guru and $j$ is a voter on a path ending at $i$. Now back within `Pathfinder`, we add all paths $\eta$ to the set $\bar{\eta}$ containing all paths. Finally within `Pathfinder` we check for all paths in $\bar{\eta}_i$ if any of them are such that $\bar{\eta}_i \subset \bar{\eta}_j$ for any $\bar{\eta}_j \in \bar{\eta}$. If this is the case, then we remove $\bar{\eta}_i$ from $\bar{\eta}$. This way all our paths within $\bar{\eta}$ will be unique.

Finally, `Pathfinder` outputs the augmented $D^T$ and $\bar{\eta}$ as `Array`.

**Pathfinder**

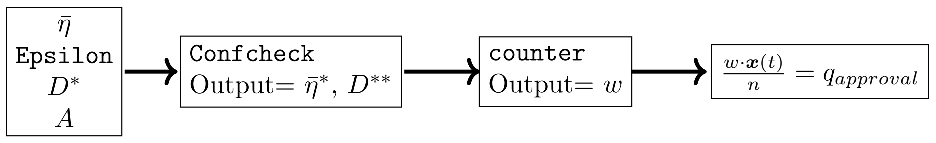


Now that we have all the paths, we can start applying the approval-based method or viscous democracy and subsequently find the probabilities we are looking for. First, for the approval-based method, we use a function called `Confcheck` which takes as inputs $\bar{\eta}$ (`Array`) containing all the unique paths, `Epsilon`, $D^*$ and $A$. What this function does is that it generates $\bar{n}$ from $A$, from which it picks a number $i$ starting at zero up to $n-1$. It takes the first path containing $i$ and checks all entries along the part of the path between $i$ and the guru. If any of these $j$ are in contact with $i$ (i.e. $A_{ij} = 1$) but $\texttt{Epsilon}_{ij} = 0$ with $i$ not being a guru themselves, then we have voter $i$ retract their vote and become a guru. If this happens we also have to change $D^*$ such that $D^*_{ii} = 1$ and $D^*_{ij} = 0$ for all $i \neq j$. But because $i$ is a guru now, we also have to split all the paths in `Array` in such a way that $i$ now becomes a guru for all paths that include $i$. After doing this for all $i \in \bar{n}$, `Confcheck` outputs the new delegation matrix $D^{**}$ and the new `Array` which we now will call `Paths`/$\bar{\eta}^*$ as it contains the final version of all paths. This final $D^{**}$ will still have multiple nonzero entries on the rows corresponding with the gurus. This will be used for the counting of the votes, which is done with the function `counter`. `counter` only takes $D^*$ as input and sets all nonzero entries equal to one. Then it creates a vector $w$ containing all the weights of all the voters. It uses the following formula:

$$w_i = \begin{cases} 0 & \text{if } D^*_{ii} = 0 \\ \sum_{j=0}^{n-1} D^*_{ij} & \text{if } D^*_{ii} = 1 \end{cases}$$

`counter` then outputs this vector $w$ and to find the probability we are looking for we then take the dot product of $w$ and $x(t)$ and divide by $n$ to find the probability $q_{approval}$.

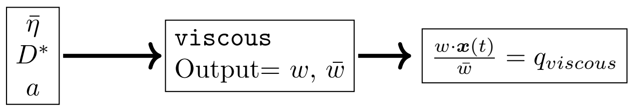
**Approval based method**



$\bar{\eta}$
Epsilon
$D^*$
$A$
Confcheck
Output= $\bar{\eta}^*$, $D^{**}$
counter
Output= $w$
$\frac{w \cdot \boldsymbol{x}(t)}{n} = q_{approval}$

The viscous case is implemented differently. It uses the function `viscous` which takes as input $D^*$, $\bar{\eta}$ (`Array`) containing all the paths and the reluctancy constant $a$. It then finds the weight vector $w$ for each guru $i$ as described in section 5.3 with all nonguru entries being equal to zero. It also calculates the total voter weight $\bar{w}$ which is just the sum of all $w_i$'s. `viscous` then outputs $w$ and the weight $\bar{w}$. From this we then find $q_{viscous}$ as the dot product of $w$ and $\boldsymbol{x}(t)$ and then divide by the weight $\bar{w}$.

**Viscous Democracy**

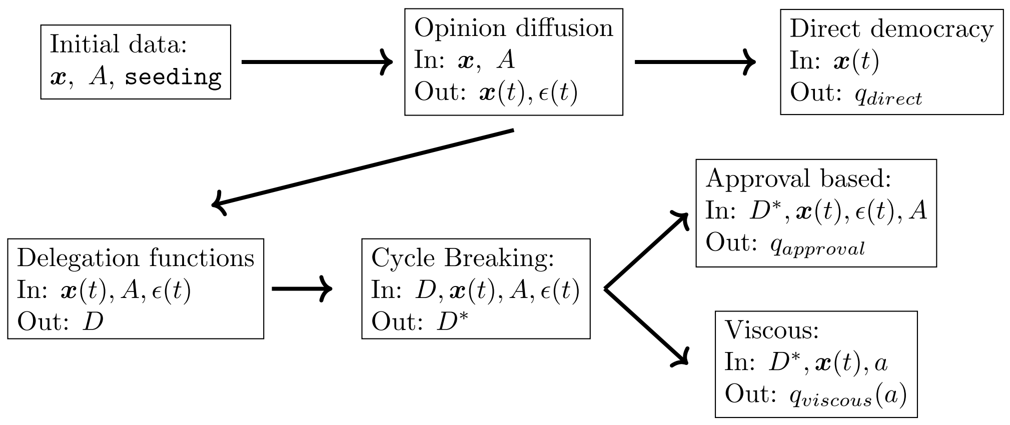


Finally we can start putting all of these function into a general framework to find the probability vector $q$ containing the probabilities for 0, 1, 5, 10 and 20 diffusion steps. This general framework will be used by twelve different delegation mechanisms (including the approval method and viscous democracy with a=0.2 and a=0.8 as seperate delegation mechanisms) and are given by the following aggregate functions:

- `Aggregate_Camp`
- `Aggreg_markrnd`
- `Aggreg_markmax`
- `Aggreg_markmin`
- `Aggreg_markrnd_visc`
- `Aggreg_markmax_visc`
- `Aggreg_markmin_visc`
- `Aggreg_Camp_visc`

Where the first four aggregate functions are for the approval method, and the latter four are for viscous democracy. The approval-based aggregate functions take as input $\boldsymbol{x}$, $A$ and `seeding` and the viscous democracy aggregate functions take the same data as input but also have $a = 0.2$ or $a = 0.8$ as an additional input. The structure of these functions is the same and use all of the other functions that we have already described. First, we apply diffusion twenty times and save the diffused $\boldsymbol{x}(t)$ for $t \in \{0, 1, 5, 10, 20\}$ and save the corresponding $\epsilon(t)$ as well. Then we apply the delegation functions, after which the cycle breaking functions do their work and the path-finding algorithm finds the paths and augmented delegation matrix for each of the specified diffusion stages. From here it depends on if the aggregate function relies on the approval method or viscous democracy, but in either case, we obtain a probability vector $q = (q_0, q_1, q_5, q_{10}, q_{20})$. A flowchart of how the aggregate functions, including the direct democracy case, work can be seen below.

**Flowchart for `Aggreg_xxxx`**


Initial data: $\boldsymbol{x}$, $A$, `seeding`
Opinion diffusion In: $\boldsymbol{x}$, $A$ Out: $\boldsymbol{x}(t), \epsilon(t)$
Direct democracy In: $\boldsymbol{x}(t)$ Out: $q_{direct}$
Delegation functions In: $\boldsymbol{x}(t), A, \epsilon(t)$ Out: $D$
Cycle Breaking: In: $D, \boldsymbol{x}(t), A, \epsilon(t)$ Out: $D^*$
Approval based: In: $D^*, \boldsymbol{x}(t), \epsilon(t), A$ Out: $q_{approval}$
Viscous: In: $D^*, \boldsymbol{x}(t), a$ Out: $q_{viscous}(a)$

Ultimately we now have each of these aggregate functions (including both viscous cases and direct democracy) run 200 times, each time with new initial data generated by adding 1 to the seed `seeding` every run. This will give us thirteen $200 \times 5$ matrices, one for each aggregate function, where each row represents one run. We do the same thing two more times, but this time we generate $\boldsymbol{x}$ with a $\mu$ of 0.6 and 0.7 respectively.

## 6 Results

To analyze our results, we split it up into two cases, the case where $\mu = 0.5$ and where $\mu \neq 0.5$. We do this because the results from the latter case are broadly similar and can be explained by the same arguments. All tables can be found within **Appendix A** and all graphs can be found in **Appendix B**. The first result we will be looking at is the case $\mu = 0.5$ without diffusion. This case represents a state where people have not yet had time to discuss before the vote happens and the population is not skewed to be more informed or misinformed.

From the graph in figure 1 and table 1 we see that direct democracy has a very narrow spread centered around 0.5 with a minimum of 0.443 and a maximum of 0.549. This means if people will vote directly, without delegating their vote, the probability of a preferred outcome is about 50% give or take 5%. This result is to be expected, given our chosen distribution is a normal distribution centered at $\mu = 0.5$. The minimal and maximal Markakis delegation mechanisms also act as expected. Both show a very low and very high chance to vote for a preferred outcome respectively. For these two delegation mechanisms, we can also see that when we let the viscosity constant be $a = 0.2$, then they have a respectively higher and lower chance to vote for a preferred outcome. In the maximal case, this is because each voter $i$ delegates to the voter $j$ who has the highest probability to vote for a preferred outcome, while also being in the confidence interval of voter $i$. So the voters with the highest probabilities will have a much larger voting weight than voters with probabilities not as high. But if $a = 0.2$ then the weight of the voters with the highest probabilities will be lessened, which leads to a worse result. The opposite argument happens for the minimal case.

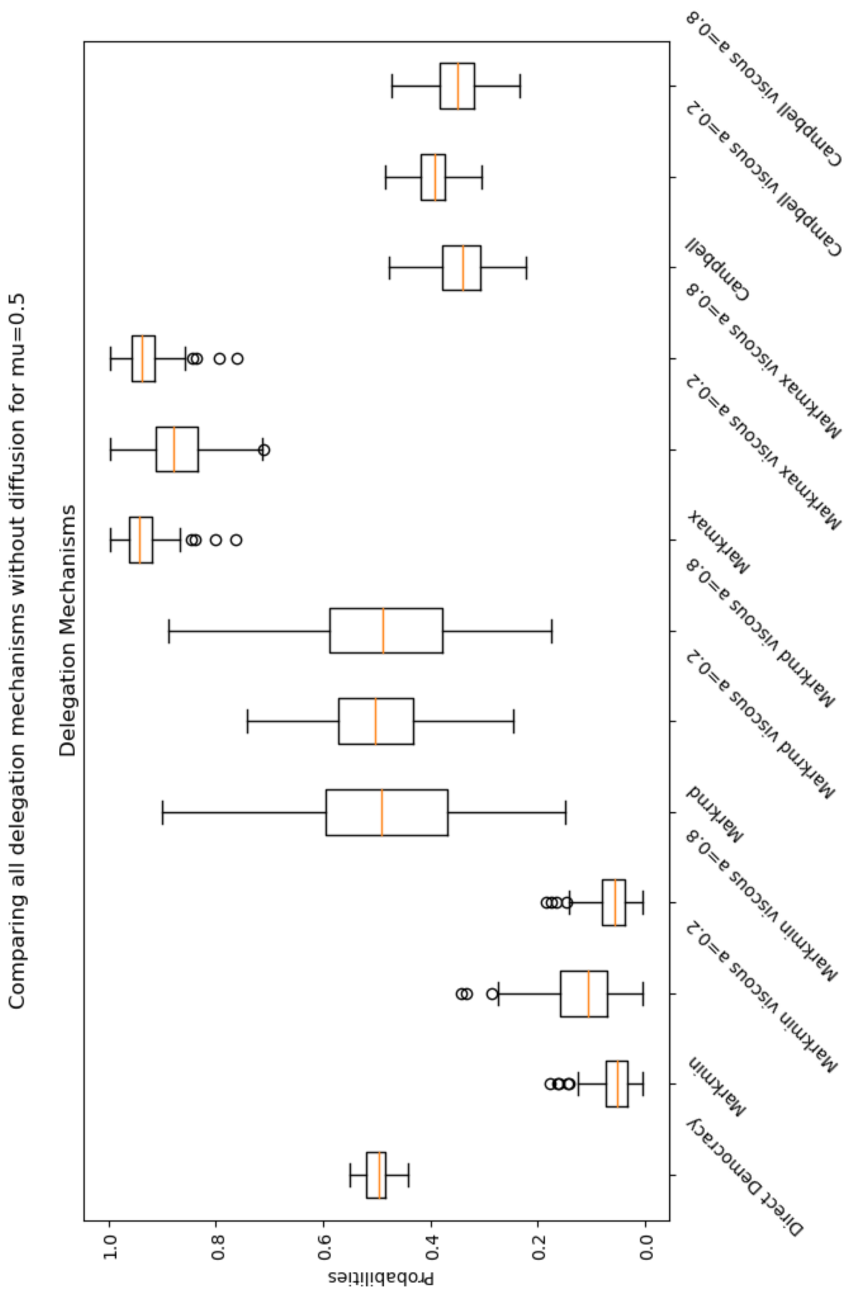


Figure 1: No diffusion

When we look at the random markakis delegation mechanism we see that the median is very close to the direct case, with the direct median being 0.497 and the random median being 0.492 for the non-viscous case. But from the graph we see that this delegation mechanism has a much larger spread compared to direct democracy. Due to this spread this delegation mechanism is not preferable over direct democracy, as it can result in a much worse result if implemented. To drive this point home, we see that for random markakis Q2=0.369 and Q3=0.596, while the low and high whiskers of direct democracy are at 0.443 and 0.549 respectively. Campbell is not a preferred delegation mechanism either in this case as from the graph and table we can see that the high whisker of Campbell never exceeds Q2 of direct democracy.

Table 1: Results for all delegation mechanisms without diffusion and $\mu$=0.5

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| Markmin | 0.004 | 0.034 | 0.052 | 0.074 | 0.125 |
| Markmin Viscous a=0.2 | 0.006 | 0.071 | 0.107 | 0.157 | 0.275 |
| Markmin Viscous a=0.8 | 0.004 | 0.037 | 0.056 | 0.079 | 0.141 |
| Markrnd | 0.149 | 0.369 | 0.492 | 0.596 | 0.901 |
| Markrnd Viscous a=0.2 | 0.245 | 0.433 | 0.503 | 0.572 | 0.741 |
| Markrnd Viscous a=0.8 | 0.174 | 0.379 | 0.488 | 0.589 | 0.888 |
| Markmax | 0.868 | 0.918 | 0.943 | 0.962 | 0.996 |
| Markmax Viscous a=0.2 | 0.714 | 0.833 | 0.879 | 0.913 | 0.996 |
| Markmax Viscous a=0.8 | 0.857 | 0.914 | 0.938 | 0.957 | 0.996 |
| Campbell | 0.223 | 0.307 | 0.34 | 0.379 | 0.478 |
| Campbell Viscous a=0.2 | 0.304 | 0.372 | 0.393 | 0.418 | 0.483 |
| Campbell Viscous a=0.8 | 0.234 | 0.318 | 0.349 | 0.383 | 0.473 |

Now if we look at the graphs in figure 3,4,5 and 6 in Appendix B and tables 3,4,5 and 6 in Appendix A we can see that after all diffusion steps, the medians remain roughly the same value except for the minimal and maximal markakis delegations whose medians both move closer to 0.5. Something that does happen regardless of the delegation mechanism chosen, is that opinion diffusion causes the spread to increase as the probability vector gets more diffused. The only delegation mechanism that has a higher chance of resulting in a better result than direct democracy, with and without opinion diffusion, is the maximal delegation mechanism. But this is not a realistic mechanism, so in the case that voter probabilities are normally distributed with $\mu = 0.5$ then no realistic mechanism that we have tested is preferable to direct democracy.

The second case we consider is when $\mu = 0.6$ and $\mu = 0.7$. When no diffusion has taken place we can see that both markakis random and markakis maximal have better chances to pick a preferred outcome when comparing them to direct democracy. Only the low whisker of markakis random is lower than that of direct democracy, with the low whisker of markakis random having a value of 0.408 and 0.5 and the low whisker of direct democracy having a value of 0.521 and 0.596 for $\mu = 0.6$ and $\mu = 0.7$ respectively. This means that there is only a slim chance that markakis random will result in a worse result than direct democracy. Campbell and markakis minimal both have a worse chance to vote for the preferred outcome than direct democracy. This changes when we apply the opinion diffusion. After only 5 stages of diffusion we see in figure 9 and table 9 that the medians of markakis minimal (0.639) are nearly identical to the median of direct democracy (0.672). A possible explanation for this is that because of opinion diffusion and our definition of $\epsilon$ that the probabilities $x_i \in \boldsymbol{x}(t)$ of voters get "pulled" towards the mean $\mu$. Because opinion diffusion transforms a probability $x_i$ into the average of all voter probabilities $x_j$ where $j$ is in contact with $i$ and with $x_j$ within the confidence interval around $x_i$; And because all $x_i$ are chosen from a normal distribution, there are more voters with probabilities around 0.6 and 0.7 compared to 0.5 or 0.4. Thus the averages are more likely to be skewed towards 0.6 and 0.7. Another factor that helps explain this increase for markakis minimal is that we define $\epsilon_i$ as $\epsilon_i = 0.5 - |0.5 - x_i|$ and this value decreases as $x_i$ moves away from 0.5. Combine these two observations together and we have a reasonable explanation on why the minimal markakis mechanism starts to compete with direct democracy when opinion diffusion takes place. The same can't be said for Campbell. This delegation mechanism keeps underperforming, which is most likely because voters with probabilities between 0.75 and 0.25 have a $\frac{1}{3}$ probability of delegating their vote to a voter with a probability lower than 0.25 if they are in contact with at least one such voter.

Then if we look at figure 11, which shows the boxplot for $\mu = 0.6$ and 20 diffusion stages, and table 11, we see that all delegation mechanisms lie very close together. Nearly all of them have a median probability higher than 0.8 which is 0.2 higher than the mean of our original probability vector. This means that none of them have any significant advantage over direct democracy, and in some cases, it would seem that direct democracy might be the better choice, albeit slightly. The most likely explanation for this is the convergence of the probability vector due to diffusion. Because each new diffused probability vector is just a vector of averages, we can expect that the rate of change between $x_i(t)$ and $x_i(t+1)$ decreases each time the vector is diffused. This is in line with what is discussed in the paper by Lorenz[5] who describes bifurcations of the HK model. In this paper it is touched upon that opinion diffusion using the HK model with heterogeneous bounds have bifurcations that converge. Although no formal proof of this is known, it is safe to assume that something similar is happening in our model too. Using figure 17 we may also assume that the choice of $\mu$ also influences the speed at which $\boldsymbol{x}(t)$ will converge. For $\mu = 0.5$ we see that after each diffusion step the median is slowly increasing. We assume that this median will keep increasing up to 0.5 as long as it keeps getting diffused, but as our simulation only went up to twenty diffusion steps, this is just speculation. When we compare the case of $\mu = 0.6$ and $\mu = 0.7$ from the same figure we see that the medians for $\mu = 0.7$ increase much more quickly than for $\mu = 0.6$. This speed of convergence is most likely caused by the size of the confidence interval, and the distribution of voter probabilities. As a voter $i$ with a high probability can still influence a voter $j$ with lower probability as $x_i$ can still be within the confidence interval for voter $j$. But the opposite might not be the case, as the further removed from 0.5 a voter is, the smaller their confidence interval is. This may lead to a positive feedback loop, where such voters $j$ can only increase their probabilities.

## 7 Conclusion

From our results we see that not all delegation mechanisms for liquid democracy can outperform a direct democracy. The Campbell and Markakis minimal delegation mechanisms always seem to have lesser results, although this was to be expected for the Markakis minimal delegation mechanism as it was a worst case scenario for the Markakis variants. Something to take note of for the Markakis minimal mechanism is that given a normal distribution with a mean $\mu = 0.6$ or higher we see that after 20 diffusion steps, this mechanism is nearly on par with direct democracy. The Markakis maximal delegation mechanism always outperformed a direct democracy, but this was the best case scenario mechanism and thus not very realistic. The Markakis random delegation mechanism is the most promising of the four liquid democracy delegation mechanisms. This delegation mechanism was able to show us better results compared to a direct democracy whenever $\mu = 0.6$ and $\mu = 0.7$ and diffusion was limited to at most 10 stages, with the best result occurring when diffusion had not taken place. From this, we can conclude that, if our model is a realistic enough representation of the real world, liquid democracy can give better results than a direct democracy can in cases where minimal contact is had before people delegate their votes. As an additional assumption, the voter population must be positively skewed and normally distributed and randomly delegate their votes to other voters within their confidence interval.

## 8 Discussion

It would be interesting to see further research done on this subject. Because we have chosen to only simulate the populations on random graphs, seeing how the results could change if they were applied to other kinds of graphs that are designed to be closer to real-world population networks as was done in the paper by Becker et al[1] would be an intriguing study. Also, it would be very interesting to see what would change if other confidence bounds were chosen for the HK model and how the results would change in comparison to homogeneous bounds.

Because we have limited ourselves to just considering a voter population of 100 voters, increasing or decreasing the number of voters might also change the results' behaviour. Yet for the smaller sizes to give any meaningful results, the chosen distribution for the probability vector $\boldsymbol{x}$ would need some more careful consideration.

Another assumption that we made in our model is that voters know the exact probabilities of other voters they are in contact with. This may not always be realistic, so a model where this is not the case might give us even more insight into how liquid democracy behaves in the real world. The uncertainty of voter's probabilities might then also start affecting the diffusion process, which could then lead to a different convergence pattern, and subsequently to different voting results.

## References

[1] Ruben Becker, Gianlorenzo D'Angelo, Esmaeil Delfaraz, and Hugo Gilbert. Unveiling the truth in liquid democracy with misinformed voters. *Algorithmic Decision Theory*, 7:132–146, 2021.

[2] Paolo Boldi, Francesco Bonchi, Carlos Castillo, and Sebastiano Vigna. Viscous democracy for social networks. *Communications of the ACM*, pages 129–137, 2011.

[3] Joseph Campbell, Alessandra Casella, Lucas de Lara, Victoria R. Mooers, and Dilip Ravindran. Liquid democracy. two experiments on delegation in voting. *NBER WORKING PAPER SERIES*, 2022.

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

[5] Jan Lorenz. Continuous opinion dynamics under bounded confidence: A survey. *International Journal of Modern Physics C*, 18(12):1819–1838, 2007.

[6] Evangelos Markakis and Georgios Papasotiropoulos. An approval-based model for single-step liquid democracy. *Algorithmic Game Theory*, pages 360–375, 2021.

[7] Bjorn Swierczek. 5 years of liquid democracy in germany. *The Liquid Democracy journal*, pages 8–20, 2014.

[8] Sadiq Muhammed T and Saji K. Matthew. The disaster of misinformation: a review of research in social media. *International Journal of Data Science and Analytics*, 2022.

[9] V-Dem. V-dem (2024) – processed by our world in data, 2024. https://ourworldindata.org/democracy.

[10] Chiara Valsangiacomo. Clarifying and defining the concept of liquid democracy. *Swiss Political Science Review*, 28(1):61–80, 2022.

[11] Partij van de Vrijheid. Pvv-verkiezingsprogramma, 2023.

[12] Forum van Democratie. Verkiezingsprogramma-fvd, 2021.

## 9 Appendix

### 9.1 A: Tables

Table 2: Results for all delegation mechanisms without diffusion and $\mu$=0.5

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| Markmin | 0.004 | 0.034 | 0.052 | 0.074 | 0.125 |
| Markmin Viscous a=0.2 | 0.006 | 0.071 | 0.107 | 0.157 | 0.275 |
| Markmin Viscous a=0.8 | 0.004 | 0.037 | 0.056 | 0.079 | 0.141 |
| Markrnd | 0.149 | 0.369 | 0.492 | 0.596 | 0.901 |
| Markrnd Viscous a=0.2 | 0.245 | 0.433 | 0.503 | 0.572 | 0.741 |
| Markrnd Viscous a=0.8 | 0.174 | 0.379 | 0.488 | 0.589 | 0.888 |
| Markmax | 0.868 | 0.918 | 0.943 | 0.962 | 0.996 |
| Markmax Viscous a=0.2 | 0.714 | 0.833 | 0.879 | 0.913 | 0.996 |
| Markmax Viscous a=0.8 | 0.857 | 0.914 | 0.938 | 0.957 | 0.996 |
| Campbell | 0.223 | 0.307 | 0.34 | 0.379 | 0.478 |
| Campbell Viscous a=0.2 | 0.304 | 0.372 | 0.393 | 0.418 | 0.483 |
| Campbell Viscous a=0.8 | 0.234 | 0.318 | 0.349 | 0.383 | 0.473 |

Table 3: All delegation mechanisms after 1 step of diffusion and $\mu$=0.5

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| Markmin | 0.002 | 0.041 | 0.067 | 0.099 | 0.176 |
| Markmin Viscous a=0.2 | 0.005 | 0.071 | 0.113 | 0.162 | 0.291 |
| Markmin Viscous a=0.8 | 0.005 | 0.038 | 0.059 | 0.089 | 0.159 |
| Markrnd | 0.052 | 0.372 | 0.488 | 0.592 | 0.912 |
| Markrnd Viscous a=0.2 | 0.238 | 0.428 | 0.494 | 0.561 | 0.751 |
| Markrnd Viscous a=0.8 | 0.102 | 0.385 | 0.489 | 0.579 | 0.844 |
| Markmax | 0.823 | 0.905 | 0.935 | 0.963 | 0.999 |
| Markmax Viscous a=0.2 | 0.695 | 0.82 | 0.864 | 0.913 | 0.999 |
| Markmax Viscous a=0.8 | 0.82 | 0.897 | 0.931 | 0.959 | 0.999 |
| Campbell | 0.211 | 0.312 | 0.349 | 0.381 | 0.482 |
| Campbell Viscous a=0.2 | 0.297 | 0.382 | 0.411 | 0.442 | 0.519 |
| Campbell Viscous a=0.8 | 0.243 | 0.325 | 0.362 | 0.39 | 0.48 |

Table 4: All delegation mechanisms after 5 steps of diffusion for mu=0.5

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| Markmin | 0.006 | 0.075 | 0.105 | 0.155 | 0.248 |
| Markmin Viscous a=0.2 | 0.002 | 0.111 | 0.173 | 0.281 | 0.535 |
| Markmin Viscous a=0.8 | 0.002 | 0.065 | 0.102 | 0.192 | 0.355 |
| Markrnd | 0.057 | 0.306 | 0.492 | 0.662 | 0.957 |
| Markrnd Viscous a=0.2 | 0.142 | 0.383 | 0.491 | 0.591 | 0.844 |
| Markrnd Viscous a=0.8 | 0.077 | 0.312 | 0.494 | 0.647 | 0.938 |
| Markmax | 0.656 | 0.819 | 0.891 | 0.929 | 0.993 |
| Markmax Viscous a=0.2 | 0.367 | 0.669 | 0.81 | 0.874 | 0.994 |
| Markmax Viscous a=0.8 | 0.077 | 0.312 | 0.494 | 0.647 | 0.938 |
| Campbell | 0.157 | 0.283 | 0.331 | 0.414 | 0.596 |
| Campbell Viscous a=0.2 | 0.266 | 0.365 | 0.423 | 0.489 | 0.672 |
| Campbell Viscous a=0.8 | 0.175 | 0.299 | 0.347 | 0.428 | 0.596 |

Table 5: All delegation mechanisms after 10 steps of diffusion for mu=0.5

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| Markmin | 0.002 | 0.081 | 0.132 | 0.604 | 0.823 |
| Markmin Viscous a=0.2 | 0.002 | 0.123 | 0.193 | 0.591 | 0.801 |
| Markmin Viscous a=0.8 | 0.002 | 0.073 | 0.134 | 0.61 | 0.822 |
| Markrnd | 0.046 | 0.251 | 0.503 | 0.731 | 0.965 |
| Markrnd Viscous a=0.2 | 0.109 | 0.337 | 0.488 | 0.651 | 0.866 |
| Markrnd Viscous a=0.8 | 0.06 | 0.251 | 0.494 | 0.721 | 0.949 |
| Markmax | 0.197 | 0.357 | 0.856 | 0.919 | 0.99 |
| Markmax Viscous a=0.2 | 0.215 | 0.37 | 0.786 | 0.867 | 0.994 |
| Markmax Viscous a=0.8 | 0.198 | 0.367 | 0.834 | 0.924 | 0.994 |
| Campbell | 0.123 | 0.235 | 0.343 | 0.457 | 0.779 |
| Campbell Viscous a=0.2 | 0.179 | 0.291 | 0.439 | 0.536 | 0.815 |
| Campbell Viscous a=0.8 | 0.137 | 0.242 | 0.362 | 0.471 | 0.776 |

Table 6: All delegation mechanisms after 20 steps of diffusion for mu=0.5

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |
| Markmin | 0.008 | 0.094 | 0.206 | 0.738 | 0.92 |
| Markmin Viscous a=0.2 | 0.002 | 0.127 | 0.268 | 0.711 | 0.88 |
| Markmin Viscous a=0.8 | 0.002 | 0.082 | 0.216 | 0.737 | 0.915 |
| Markrnd | 0.022 | 0.168 | 0.465 | 0.797 | 0.978 |
| Markrnd Viscous a=0.2 | 0.1 | 0.273 | 0.489 | 0.708 | 0.91 |
| Markrnd Viscous a=0.8 | 0.031 | 0.185 | 0.464 | 0.792 | 0.964 |
| Markmax | 0.101 | 0.228 | 0.688 | 0.907 | 0.989 |
| Markmax Viscous a=0.2 | 0.102 | 0.259 | 0.655 | 0.868 | 0.99 |
| Markmax Viscous a=0.8 | 0.082 | 0.24 | 0.69 | 0.916 | 0.99 |
| Campbell | 0.076 | 0.207 | 0.361 | 0.74 | 0.919 |
| Campbell Viscous a=0.2 | 0.076 | 0.219 | 0.427 | 0.722 | 0.908 |
| Campbell Viscous a=0.8 | 0.076 | 0.208 | 0.377 | 0.737 | 0.917 |

Table 7: All delegation mechanisms after 0 steps of diffusion for mu=0.6

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| Markmin | 0.008 | 0.061 | 0.086 | 0.12 | 0.2 |
| Markmin Viscous a=0.2 | 0.012 | 0.11 | 0.164 | 0.219 | 0.377 |
| Markmin Viscous a=0.8 | 0.008 | 0.065 | 0.093 | 0.129 | 0.212 |
| Markrnd | 0.408 | 0.603 | 0.698 | 0.783 | 0.96 |
| Markrnd Viscous a=0.2 | 0.403 | 0.576 | 0.639 | 0.71 | 0.901 |
| Markrnd Viscous a=0.8 | 0.354 | 0.601 | 0.694 | 0.771 | 0.957 |
| Markmax | 0.919 | 0.951 | 0.969 | 0.981 | 0.999 |
| Markmax Viscous a=0.2 | 0.777 | 0.888 | 0.925 | 0.972 | 0.999 |
| Markmax Viscous a=0.8 | 0.91 | 0.947 | 0.966 | 0.979 | 0.999 |
| Campbell | 0.261 | 0.338 | 0.374 | 0.413 | 0.521 |
| Campbell Viscous a=0.2 | 0.364 | 0.415 | 0.442 | 0.465 | 0.536 |
| Campbell Viscous a=0.8 | 0.284 | 0.349 | 0.386 | 0.421 | 0.529 |

Table 8: All delegation mechanisms after 1 step of diffusion for mu=0.6

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| Markmin | 0.004 | 0.07 | 0.129 | 0.203 | 0.381 |
| Markmin Viscous a=0.2 | 0.018 | 0.115 | 0.177 | 0.248 | 0.44 |
| Markmin Viscous a=0.8 | 0.009 | 0.068 | 0.108 | 0.154 | 0.273 |
| Markrnd | 0.38 | 0.625 | 0.726 | 0.813 | 0.967 |
| Markrnd Viscous a=0.2 | 0.397 | 0.583 | 0.656 | 0.726 | 0.924 |
| Markrnd Viscous a=0.8 | 0.397 | 0.629 | 0.72 | 0.798 | 0.96 |
| Markmax | 0.902 | 0.944 | 0.964 | 0.979 | 0.999 |
| Markmax Viscous a=0.2 | 0.773 | 0.882 | 0.924 | 0.961 | 0.999 |
| Markmax Viscous a=0.8 | 0.897 | 0.939 | 0.962 | 0.977 | 0.999 |
| Campbell | 0.234 | 0.358 | 0.402 | 0.457 | 0.601 |
| Campbell Viscous a=0.2 | 0.375 | 0.458 | 0.489 | 0.526 | 0.61 |
| Campbell Viscous a=0.8 | 0.263 | 0.372 | 0.415 | 0.467 | 0.597 |

Table 9: All delegation mechanisms after 5 steps of diffusion for mu=0.6

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| Markmin | 0.035 | 0.419 | 0.639 | 0.681 | 0.757 |
| Markmin Viscous a=0.2 | 0.524 | 0.614 | 0.653 | 0.691 | 0.776 |
| Markmin Viscous a=0.8 | 0.503 | 0.613 | 0.658 | 0.695 | 0.795 |
| Markrnd | 0.459 | 0.687 | 0.788 | 0.866 | 0.979 |
| Markrnd Viscous a=0.2 | 0.433 | 0.641 | 0.703 | 0.783 | 0.941 |
| Markrnd Viscous a=0.8 | 0.444 | 0.686 | 0.78 | 0.851 | 0.974 |
| Markmax | 0.858 | 0.917 | 0.938 | 0.96 | 0.997 |
| Markmax Viscous a=0.2 | 0.724 | 0.847 | 0.896 | 0.93 | 1.0 |
| Markmax Viscous a=0.8 | 0.865 | 0.917 | 0.943 | 0.967 | 1.0 |
| Campbell | 0.264 | 0.427 | 0.498 | 0.633 | 0.823 |
| Campbell Viscous a=0.2 | 0.426 | 0.55 | 0.604 | 0.655 | 0.805 |
| Campbell Viscous a=0.8 | 0.303 | 0.445 | 0.518 | 0.634 | 0.822 |

Table 10: All delegation mechanisms after 10 steps of diffusion for mu=0.6

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| Markmin | 0.569 | 0.694 | 0.744 | 0.792 | 0.896 |
| Markmin Viscous a=0.2 | 0.524 | 0.614 | 0.653 | 0.691 | 0.776 |
| Markmin Viscous a=0.8 | 0.554 | 0.686 | 0.739 | 0.788 | 0.908 |
| Markrnd | 0.512 | 0.735 | 0.832 | 0.892 | 0.979 |
| Markrnd Viscous a=0.2 | 0.464 | 0.677 | 0.735 | 0.821 | 0.965 |
| Markrnd Viscous a=0.8 | 0.542 | 0.733 | 0.82 | 0.886 | 0.978 |
| Markmax | 0.827 | 0.905 | 0.931 | 0.96 | 0.997 |
| Markmax Viscous a=0.2 | 0.766 | 0.851 | 0.898 | 0.933 | 1.0 |
| Markmax Viscous a=0.8 | 0.832 | 0.909 | 0.939 | 0.967 | 1.0 |
| Campbell | 0.238 | 0.517 | 0.724 | 0.795 | 0.915 |
| Campbell Viscous a=0.2 | 0.393 | 0.623 | 0.718 | 0.786 | 0.901 |
| Campbell Viscous a=0.8 | 0.26 | 0.531 | 0.724 | 0.793 | 0.913 |

Table 11: All delegation mechanisms after 20 steps of diffusion for mu=0.6

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |
| Markmin | 0.642 | 0.783 | 0.836 | 0.878 | 0.965 |
| Markmin Viscous a=0.2 | 0.586 | 0.744 | 0.794 | 0.856 | 0.942 |
| Markmin Viscous a=0.8 | 0.606 | 0.765 | 0.823 | 0.879 | 0.965 |
| Markrnd | 0.621 | 0.792 | 0.864 | 0.92 | 0.985 |
| Markrnd Viscous a=0.2 | 0.495 | 0.697 | 0.771 | 0.852 | 0.979 |
| Markrnd Viscous a=0.8 | 0.581 | 0.779 | 0.857 | 0.913 | 0.985 |
| Markmax | 0.762 | 0.876 | 0.928 | 0.957 | 0.996 |
| Markmax Viscous a=0.2 | 0.721 | 0.838 | 0.899 | 0.928 | 1.0 |
| Markmax Viscous a=0.8 | 0.812 | 0.901 | 0.936 | 0.962 | 1.0 |
| Campbell | 0.519 | 0.736 | 0.822 | 0.881 | 0.955 |
| Campbell Viscous a=0.2 | 0.543 | 0.733 | 0.809 | 0.866 | 0.945 |
| Campbell Viscous a=0.8 | 0.533 | 0.735 | 0.82 | 0.878 | 0.953 |

Table 12: All delegation mechanisms after 0 steps of diffusion for mu=0.7

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| Markmin | 0.017 | 0.11 | 0.147 | 0.185 | 0.293 |
| Markmin Viscous a=0.2 | 0.026 | 0.169 | 0.234 | 0.291 | 0.465 |
| Markmin Viscous a=0.8 | 0.018 | 0.115 | 0.154 | 0.192 | 0.303 |
| Markrnd | 0.5 | 0.7 | 0.824 | 0.872 | 0.929 |
| Markrnd Viscous a=0.2 | 0.537 | 0.687 | 0.752 | 0.799 | 0.95 |
| Markrnd Viscous a=0.8 | 0.567 | 0.749 | 0.817 | 0.882 | 0.983 |
| Markmax | 0.941 | 0.97 | 0.982 | 0.99 | 1.0 |
| Markmax Viscous a=0.2 | 0.854 | 0.926 | 0.952 | 0.987 | 1.0 |
| Markmax Viscous a=0.8 | 0.941 | 0.967 | 0.98 | 0.988 | 1.0 |
| Campbell | 0.27 | 0.376 | 0.423 | 0.472 | 0.593 |
| Campbell Viscous a=0.2 | 0.377 | 0.46 | 0.489 | 0.525 | 0.62 |
| Campbell Viscous a=0.8 | 0.291 | 0.389 | 0.433 | 0.475 | 0.59 |

Table 13: All delegation mechanisms after 1 step of diffusion for mu=0.7

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| Markmin | 0.026 | 0.163 | 0.246 | 0.387 | 0.639 |
| Markmin Viscous a=0.2 | 0.032 | 0.203 | 0.279 | 0.366 | 0.57 |
| Markmin Viscous a=0.8 | 0.021 | 0.13 | 0.19 | 0.265 | 0.459 |
| Markrnd | 0.669 | 0.724 | 0.79 | 0.854 | 0.917 |
| Markrnd Viscous a=0.2 | 0.588 | 0.729 | 0.772 | 0.824 | 0.963 |
| Markrnd Viscous a=0.8 | 0.626 | 0.779 | 0.846 | 0.891 | 0.978 |
| Markmax | 0.938 | 0.967 | 0.981 | 0.99 | 1.0 |
| Markmax Viscous a=0.2 | 0.84 | 0.925 | 0.953 | 0.988 | 1.0 |
| Markmax Viscous a=0.8 | 0.933 | 0.965 | 0.979 | 0.989 | 1.0 |
| Campbell | 0.266 | 0.442 | 0.504 | 0.599 | 0.697 |
| Campbell Viscous a=0.2 | 0.438 | 0.539 | 0.578 | 0.618 | 0.679 |
| Campbell Viscous a=0.8 | 0.29 | 0.458 | 0.517 | 0.603 | 0.693 |

Table 14: All delegation mechanisms after 5 steps of diffusion for mu=0.7

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| Markmin | 0.65 | 0.723 | 0.751 | 0.781 | 0.846 |
| Markmin Viscous a=0.2 | 0.656 | 0.735 | 0.767 | 0.796 | 0.876 |
| Markmin Viscous a=0.8 | 0.66 | 0.744 | 0.774 | 0.8 | 0.872 |
| Markrnd | 0.718 | 0.801 | 0.854 | 0.889 | 0.961 |
| Markrnd Viscous a=0.2 | 0.634 | 0.771 | 0.816 | 0.866 | 0.986 |
| Markrnd Viscous a=0.8 | 0.713 | 0.831 | 0.88 | 0.92 | 0.988 |
| Markmax | 0.902 | 0.948 | 0.966 | 0.98 | 1.0 |
| Markmax Viscous a=0.2 | 0.795 | 0.891 | 0.935 | 0.958 | 1.0 |
| Markmax Viscous a=0.8 | 0.896 | 0.948 | 0.969 | 0.983 | 1.0 |
| Campbell | 0.302 | 0.579 | 0.758 | 0.809 | 0.902 |
| Campbell Viscous a=0.2 | 0.516 | 0.68 | 0.756 | 0.803 | 0.896 |
| Campbell Viscous a=0.8 | 0.308 | 0.595 | 0.759 | 0.808 | 0.901 |

Table 15: All delegation mechanisms after 10 steps of diffusion for mu=0.7

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.721 | 0.812 | 0.848 | 0.877 | 0.958 |
| Markmin | 0.714 | 0.801 | 0.838 | 0.869 | 0.952 |
| Markmin Viscous a=0.2 | 0.682 | 0.786 | 0.822 | 0.856 | 0.958 |
| Markmin Viscous a=0.8 | 0.696 | 0.8 | 0.838 | 0.873 | 0.955 |
| Markrnd | 0.745 | 0.839 | 0.878 | 0.91 | 0.969 |
| Markrnd Viscous a=0.2 | 0.635 | 0.789 | 0.841 | 0.893 | 0.994 |
| Markrnd Viscous a=0.8 | 0.749 | 0.855 | 0.902 | 0.938 | 0.994 |
| Markmax | 0.895 | 0.942 | 0.962 | 0.979 | 1.0 |
| Markmax Viscous a=0.2 | 0.785 | 0.887 | 0.932 | 0.957 | 1.0 |
| Markmax Viscous a=0.8 | 0.898 | 0.946 | 0.966 | 0.981 | 1.0 |
| Campbell | 0.608 | 0.766 | 0.832 | 0.873 | 0.963 |
| Campbell Viscous a=0.2 | 0.618 | 0.766 | 0.823 | 0.869 | 0.96 |
| Campbell Viscous a=0.8 | 0.609 | 0.762 | 0.83 | 0.872 | 0.962 |

Table 16: All delegation mechanisms after 20 steps of diffusion for mu=0.7

| **Delegation Mechanism** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| Direct Democracy | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |
| Markmin | 0.768 | 0.864 | 0.903 | 0.93 | 0.991 |
| Markmin Viscous a=0.2 | 0.721 | 0.832 | 0.876 | 0.911 | 0.991 |
| Markmin Viscous a=0.8 | 0.737 | 0.852 | 0.904 | 0.933 | 0.991 |
| Markrnd | 0.767 | 0.852 | 0.918 | 0.944 | 0.974 |
| Markrnd Viscous a=0.2 | 0.659 | 0.805 | 0.859 | 0.909 | 0.997 |
| Markrnd Viscous a=0.8 | 0.782 | 0.879 | 0.922 | 0.951 | 0.997 |
| Markmax | 0.848 | 0.925 | 0.956 | 0.977 | 1.0 |
| Markmax Viscous a=0.2 | 0.782 | 0.884 | 0.925 | 0.953 | 1.0 |
| Markmax Viscous a=0.8 | 0.879 | 0.937 | 0.962 | 0.978 | 1.0 |
| Campbell | 0.59 | 0.796 | 0.889 | 0.933 | 0.992 |
| Campbell Viscous a=0.2 | 0.617 | 0.8 | 0.878 | 0.925 | 0.992 |
| Campbell Viscous a=0.8 | 0.592 | 0.794 | 0.888 | 0.931 | 0.992 |

Table 17: Comparison Markmin and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmin mu=0.5** | | | | | |
| **0** | 0.004 | 0.034 | 0.052 | 0.074 | 0.125 |
| **1** | 0.002 | 0.041 | 0.067 | 0.099 | 0.176 |
| **5** | 0.006 | 0.075 | 0.105 | 0.155 | 0.248 |
| **10** | 0.002 | 0.081 | 0.132 | 0.604 | 0.823 |
| **20** | 0.008 | 0.094 | 0.206 | 0.738 | 0.92 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 18: Comparison Markmin and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmin mu=0.5** | | | | | |
| **0** | 0.004 | 0.034 | 0.052 | 0.074 | 0.125 |
| **1** | 0.002 | 0.041 | 0.067 | 0.099 | 0.176 |
| **5** | 0.006 | 0.075 | 0.105 | 0.155 | 0.248 |
| **10** | 0.002 | 0.081 | 0.132 | 0.604 | 0.823 |
| **20** | 0.008 | 0.094 | 0.206 | 0.738 | 0.92 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 19: Comparison Markminvisc a=0.2 and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markminvisc a=0.2 and mu=0.5** | | | | | |
| **0** | 0.006 | 0.071 | 0.107 | 0.157 | 0.275 |
| **1** | 0.005 | 0.071 | 0.113 | 0.162 | 0.291 |
| **5** | 0.002 | 0.111 | 0.173 | 0.281 | 0.535 |
| **10** | 0.002 | 0.123 | 0.193 | 0.591 | 0.801 |
| **20** | 0.002 | 0.127 | 0.268 | 0.711 | 0.88 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 20: Comparison Markrnd and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markrnd mu=0.5** | | | | | |
| **0** | 0.149 | 0.369 | 0.492 | 0.596 | 0.901 |
| **1** | 0.052 | 0.372 | 0.488 | 0.592 | 0.912 |
| **5** | 0.057 | 0.306 | 0.492 | 0.662 | 0.957 |
| **10** | 0.046 | 0.251 | 0.503 | 0.731 | 0.965 |
| **20** | 0.022 | 0.168 | 0.465 | 0.797 | 0.978 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 21: Comparison Markrndvisc a=0.2 and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markrndvisc a=0.2 and mu=0.5** | | | | | |
| **0** | 0.245 | 0.433 | 0.503 | 0.572 | 0.741 |
| **1** | 0.238 | 0.428 | 0.494 | 0.561 | 0.751 |
| **5** | 0.142 | 0.383 | 0.491 | 0.591 | 0.844 |
| **10** | 0.109 | 0.337 | 0.488 | 0.651 | 0.866 |
| **20** | 0.1 | 0.273 | 0.489 | 0.708 | 0.91 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 22: Comparison Markrndvisc a=0.8 and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markrndvisc a=0.8 and mu=0.5** | | | | | |
| **0** | 0.174 | 0.379 | 0.488 | 0.589 | 0.888 |
| **1** | 0.102 | 0.385 | 0.489 | 0.579 | 0.844 |
| **5** | 0.077 | 0.312 | 0.494 | 0.647 | 0.938 |
| **10** | 0.06 | 0.251 | 0.494 | 0.721 | 0.949 |
| **20** | 0.031 | 0.185 | 0.464 | 0.792 | 0.964 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 23: Comparison Markmax and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmax mu=0.5** | | | | | |
| **0** | 0.868 | 0.918 | 0.943 | 0.962 | 0.996 |
| **1** | 0.823 | 0.905 | 0.935 | 0.963 | 0.999 |
| **5** | 0.656 | 0.819 | 0.891 | 0.929 | 0.993 |
| **10** | 0.197 | 0.357 | 0.856 | 0.919 | 0.99 |
| **20** | 0.101 | 0.228 | 0.688 | 0.907 | 0.989 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 24: Comparison Markmaxvisc a=0.2 and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmaxvisc a=0.2 and mu=0.5** | | | | | |
| **0** | 0.714 | 0.833 | 0.879 | 0.913 | 0.996 |
| **1** | 0.695 | 0.82 | 0.864 | 0.913 | 0.999 |
| **5** | 0.367 | 0.669 | 0.81 | 0.874 | 0.994 |
| **10** | 0.215 | 0.37 | 0.786 | 0.867 | 0.994 |
| **20** | 0.102 | 0.259 | 0.655 | 0.868 | 0.99 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 25: Comparison Markmaxvisc a=0.8 and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmaxvisc a=0.8 and mu=0.5** | | | | | |
| **0** | 0.857 | 0.914 | 0.938 | 0.957 | 0.996 |
| **1** | 0.82 | 0.897 | 0.931 | 0.959 | 0.999 |
| **5** | 0.506 | 0.76 | 0.886 | 0.933 | 0.994 |
| **10** | 0.198 | 0.367 | 0.834 | 0.924 | 0.994 |
| **20** | 0.082 | 0.24 | 0.69 | 0.916 | 0.99 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 26: Comparison Campbell and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Campbell mu=0.5** | | | | | |
| **0** | 0.223 | 0.307 | 0.34 | 0.379 | 0.478 |
| **1** | 0.211 | 0.312 | 0.349 | 0.381 | 0.482 |
| **5** | 0.157 | 0.283 | 0.331 | 0.414 | 0.596 |
| **10** | 0.123 | 0.235 | 0.343 | 0.457 | 0.779 |
| **20** | 0.076 | 0.207 | 0.361 | 0.74 | 0.919 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 27: Comparison Campbellvisc a=0.2 and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Campbellvisc a=0.2 and mu=0.5** | | | | | |
| **0** | 0.304 | 0.372 | 0.393 | 0.418 | 0.483 |
| **1** | 0.297 | 0.382 | 0.411 | 0.442 | 0.519 |
| **5** | 0.266 | 0.365 | 0.423 | 0.489 | 0.672 |
| **10** | 0.179 | 0.291 | 0.439 | 0.536 | 0.815 |
| **20** | 0.076 | 0.219 | 0.427 | 0.722 | 0.908 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 28: Comparison Campbellvisc a=0.8 and Direct Democracy for mu=0.5

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Campbellvisc a=0.8 and mu=0.5** | | | | | |
| **0** | 0.234 | 0.318 | 0.349 | 0.383 | 0.473 |
| **1** | 0.243 | 0.325 | 0.362 | 0.39 | 0.48 |
| **5** | 0.175 | 0.299 | 0.347 | 0.428 | 0.596 |
| **10** | 0.137 | 0.242 | 0.362 | 0.471 | 0.776 |
| **20** | 0.076 | 0.208 | 0.377 | 0.737 | 0.917 |
| **Direct Democracy mu=0.5** | | | | | |
| **0** | 0.443 | 0.483 | 0.497 | 0.519 | 0.549 |
| **1** | 0.414 | 0.474 | 0.496 | 0.526 | 0.574 |
| **5** | 0.307 | 0.44 | 0.494 | 0.543 | 0.683 |
| **10** | 0.188 | 0.34 | 0.496 | 0.628 | 0.838 |
| **20** | 0.097 | 0.223 | 0.506 | 0.745 | 0.92 |

Table 29: Comparison Markmin and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmin mu=0.6** | | | | | |
| **0** | 0.004 | 0.034 | 0.052 | 0.074 | 0.125 |
| **1** | 0.002 | 0.041 | 0.067 | 0.099 | 0.176 |
| **5** | 0.006 | 0.075 | 0.105 | 0.155 | 0.248 |
| **10** | 0.002 | 0.081 | 0.132 | 0.604 | 0.823 |
| **20** | 0.008 | 0.094 | 0.206 | 0.738 | 0.92 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 30: Comparison Markrnd and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markrnd mu=0.6** | | | | | |
| **0** | 0.408 | 0.603 | 0.698 | 0.783 | 0.96 |
| **1** | 0.38 | 0.625 | 0.726 | 0.813 | 0.967 |
| **5** | 0.459 | 0.687 | 0.788 | 0.866 | 0.979 |
| **10** | 0.512 | 0.735 | 0.832 | 0.892 | 0.979 |
| **20** | 0.621 | 0.792 | 0.864 | 0.92 | 0.985 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 31: Comparison Markmax and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmax mu=0.6** | | | | | |
| **0** | 0.919 | 0.951 | 0.969 | 0.981 | 0.999 |
| **1** | 0.902 | 0.944 | 0.964 | 0.979 | 0.999 |
| **5** | 0.858 | 0.917 | 0.938 | 0.96 | 0.997 |
| **10** | 0.827 | 0.905 | 0.931 | 0.96 | 0.997 |
| **20** | 0.762 | 0.876 | 0.928 | 0.957 | 0.996 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 32: Comparison Campbell and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Campbell mu=0.6** | | | | | |
| **0** | 0.261 | 0.338 | 0.374 | 0.413 | 0.521 |
| **1** | 0.234 | 0.358 | 0.402 | 0.457 | 0.601 |
| **5** | 0.264 | 0.427 | 0.498 | 0.633 | 0.823 |
| **10** | 0.238 | 0.517 | 0.724 | 0.795 | 0.915 |
| **20** | 0.519 | 0.736 | 0.822 | 0.881 | 0.955 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 33: Comparison Markminvisc a=0.2 and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markminvisc a=0.2 and mu=0.6** | | | | | |
| **0** | 0.012 | 0.11 | 0.164 | 0.219 | 0.377 |
| **1** | 0.018 | 0.115 | 0.177 | 0.248 | 0.44 |
| **5** | 0.524 | 0.614 | 0.653 | 0.691 | 0.776 |
| **10** | 0.547 | 0.68 | 0.726 | 0.777 | 0.881 |
| **20** | 0.586 | 0.744 | 0.794 | 0.856 | 0.942 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 34: Comparison Markrndvisc a=0.2 and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markrndvisc a=0.2 and mu=0.6** | | | | | |
| **0** | 0.403 | 0.576 | 0.639 | 0.71 | 0.901 |
| **1** | 0.397 | 0.583 | 0.656 | 0.726 | 0.924 |
| **5** | 0.433 | 0.641 | 0.703 | 0.783 | 0.941 |
| **10** | 0.464 | 0.677 | 0.735 | 0.821 | 0.965 |
| **20** | 0.495 | 0.697 | 0.771 | 0.852 | 0.979 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 35: Comparison Markmaxvisc a=0.2 and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmaxvisc a=0.2 and mu=0.6** | | | | | |
| **0** | 0.777 | 0.888 | 0.925 | 0.972 | 0.999 |
| **1** | 0.773 | 0.882 | 0.924 | 0.961 | 0.999 |
| **5** | 0.724 | 0.847 | 0.896 | 0.93 | 1.0 |
| **10** | 0.766 | 0.851 | 0.898 | 0.933 | 1.0 |
| **20** | 0.721 | 0.838 | 0.899 | 0.928 | 1.0 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 36: Comparison Campbell a=0.2 and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Campbell a=0.2 and mu=0.6** | | | | | |
| **0** | 0.364 | 0.415 | 0.442 | 0.465 | 0.536 |
| **1** | 0.375 | 0.458 | 0.489 | 0.526 | 0.61 |
| **5** | 0.426 | 0.55 | 0.604 | 0.655 | 0.805 |
| **10** | 0.393 | 0.623 | 0.718 | 0.786 | 0.901 |
| **20** | 0.543 | 0.733 | 0.809 | 0.866 | 0.945 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 37: Comparison Markminvisc a=0.8 and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markminvisc a=0.8 and mu=0.6** | | | | | |
| **0** | 0.008 | 0.065 | 0.093 | 0.129 | 0.212 |
| **1** | 0.009 | 0.068 | 0.108 | 0.154 | 0.273 |
| **5** | 0.503 | 0.613 | 0.658 | 0.695 | 0.795 |
| **10** | 0.554 | 0.686 | 0.739 | 0.788 | 0.908 |
| **20** | 0.606 | 0.765 | 0.823 | 0.879 | 0.965 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 38: Comparison Markminvisc a=0.8 and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markminvisc a=0.8 and mu=0.6** | | | | | |
| **0** | 0.008 | 0.065 | 0.093 | 0.129 | 0.212 |
| **1** | 0.009 | 0.068 | 0.108 | 0.154 | 0.273 |
| **5** | 0.503 | 0.613 | 0.658 | 0.695 | 0.795 |
| **10** | 0.554 | 0.686 | 0.739 | 0.788 | 0.908 |
| **20** | 0.606 | 0.765 | 0.823 | 0.879 | 0.965 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 39: Comparison Markmaxvisc a=0.8 and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmaxvisc a=0.8 and mu=0.6** | | | | | |
| **0** | 0.91 | 0.947 | 0.966 | 0.979 | 0.999 |
| **1** | 0.897 | 0.939 | 0.962 | 0.977 | 0.999 |
| **5** | 0.865 | 0.917 | 0.943 | 0.967 | 1.0 |
| **10** | 0.832 | 0.909 | 0.939 | 0.967 | 1.0 |
| **20** | 0.812 | 0.901 | 0.936 | 0.962 | 1.0 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 40: Comparison Campbellvisc a=0.8 and Direct Democracy for mu=0.6

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Campbellvisc a=0.8 and mu=0.6** | | | | | |
| **0** | 0.284 | 0.349 | 0.386 | 0.421 | 0.529 |
| **1** | 0.263 | 0.372 | 0.415 | 0.467 | 0.597 |
| **5** | 0.303 | 0.445 | 0.518 | 0.634 | 0.822 |
| **10** | 0.26 | 0.531 | 0.724 | 0.793 | 0.913 |
| **20** | 0.533 | 0.735 | 0.82 | 0.878 | 0.953 |
| **Direct Democracy mu=0.6** | | | | | |
| **0** | 0.521 | 0.559 | 0.574 | 0.595 | 0.624 |
| **1** | 0.521 | 0.582 | 0.605 | 0.629 | 0.681 |
| **5** | 0.533 | 0.635 | 0.672 | 0.704 | 0.804 |
| **10** | 0.57 | 0.706 | 0.756 | 0.803 | 0.903 |
| **20** | 0.661 | 0.79 | 0.842 | 0.882 | 0.967 |

Table 41: Comparison Markmin and Direct Democracy for mu=0.7

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmin mu=0.7** | | | | | |
| **0** | 0.017 | 0.11 | 0.147 | 0.185 | 0.293 |
| **1** | 0.026 | 0.163 | 0.246 | 0.387 | 0.639 |
| **5** | 0.65 | 0.723 | 0.751 | 0.781 | 0.846 |
| **10** | 0.714 | 0.801 | 0.838 | 0.869 | 0.952 |
| **20** | 0.768 | 0.864 | 0.903 | 0.93 | 0.991 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 42: Comparison Markrnd and Direct Democracy for mu=0.7

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markrnd mu=0.7** | | | | | |
| **0** | 0.5 | 0.7 | 0.824 | 0.872 | 0.929 |
| **1** | 0.669 | 0.724 | 0.79 | 0.854 | 0.917 |
| **5** | 0.718 | 0.801 | 0.854 | 0.889 | 0.961 |
| **10** | 0.745 | 0.839 | 0.878 | 0.91 | 0.969 |
| **20** | 0.767 | 0.852 | 0.918 | 0.944 | 0.974 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 43: Comparison Markmax and Direct Democracy for mu=0.7

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmax mu=0.7** | | | | | |
| **0** | 0.941 | 0.97 | 0.982 | 0.99 | 1.0 |
| **1** | 0.938 | 0.967 | 0.981 | 0.99 | 1.0 |
| **5** | 0.902 | 0.948 | 0.966 | 0.98 | 1.0 |
| **10** | 0.895 | 0.942 | 0.962 | 0.979 | 1.0 |
| **20** | 0.848 | 0.925 | 0.956 | 0.977 | 1.0 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 44: Comparison Campbell and Direct Democracy for mu=0.7

| Diffusion stage | Low whisker | Q2 | Median | Q3 | High whisker |
|---|---|---|---|---|---|
| **Campbell mu=0.7** | | | | | |
| **0** | 0.27 | 0.376 | 0.423 | 0.472 | 0.593 |
| **1** | 0.266 | 0.442 | 0.504 | 0.599 | 0.697 |
| **5** | 0.302 | 0.579 | 0.758 | 0.809 | 0.902 |
| **10** | 0.608 | 0.766 | 0.832 | 0.873 | 0.963 |
| **20** | 0.59 | 0.796 | 0.889 | 0.933 | 0.992 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 45: Comparison Markminvisc a=0.2 and Direct Democracy for mu=0.7

| Diffusion stage | Low whisker | Q2 | Median | Q3 | High whisker |
|---|---|---|---|---|---|
| **Markminvisc a=0.2 and mu=0.7** | | | | | |
| **0** | 0.026 | 0.169 | 0.234 | 0.291 | 0.465 |
| **1** | 0.032 | 0.203 | 0.279 | 0.366 | 0.57 |
| **5** | 0.656 | 0.735 | 0.767 | 0.796 | 0.876 |
| **10** | 0.682 | 0.786 | 0.822 | 0.856 | 0.958 |
| **20** | 0.721 | 0.832 | 0.876 | 0.911 | 0.991 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 46: Comparison Markrndvisc a=0.2 and Direct Democracy for mu=0.7

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markrndvisc a=0.2 and mu=0.7** | | | | | |
| **0** | 0.537 | 0.687 | 0.752 | 0.799 | 0.95 |
| **1** | 0.588 | 0.729 | 0.772 | 0.824 | 0.963 |
| **5** | 0.634 | 0.771 | 0.816 | 0.866 | 0.986 |
| **10** | 0.635 | 0.789 | 0.841 | 0.893 | 0.994 |
| **20** | 0.659 | 0.805 | 0.859 | 0.909 | 0.997 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 47: Comparison Markmaxvisc a=0.2 and Direct Democracy for mu=0.7

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmaxvisc a=0.2 and mu=0.7** | | | | | |
| **0** | 0.845 | 0.926 | 0.952 | 0.987 | 1.0 |
| **1** | 0.84 | 0.925 | 0.953 | 0.988 | 1.0 |
| **5** | 0.795 | 0.891 | 0.935 | 0.958 | 1.0 |
| **10** | 0.785 | 0.887 | 0.932 | 0.957 | 1.0 |
| **20** | 0.782 | 0.884 | 0.925 | 0.953 | 1.0 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 48: Comparison Campbellvisc a=0.2 and Direct Democracy for mu=0.7

| Diffusion stage | Low whisker | Q2 | Median | Q3 | High whisker |
|---|---|---|---|---|---|
| **Campbellvisc a=0.2 and mu=0.7** | | | | | |
| **0** | 0.377 | 0.46 | 0.489 | 0.525 | 0.62 |
| **1** | 0.438 | 0.539 | 0.578 | 0.618 | 0.679 |
| **5** | 0.516 | 0.68 | 0.756 | 0.803 | 0.896 |
| **10** | 0.618 | 0.766 | 0.823 | 0.869 | 0.96 |
| **20** | 0.617 | 0.8 | 0.878 | 0.925 | 0.992 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 49: Comparison Markminvisc a=0.8 and Direct Democracy for mu=0.7

| Diffusion stage | Low whisker | Q2 | Median | Q3 | High whisker |
|---|---|---|---|---|---|
| **Markminvisc a=0.8 and mu=0.7** | | | | | |
| **0** | 0.018 | 0.115 | 0.154 | 0.192 | 0.303 |
| **1** | 0.021 | 0.13 | 0.19 | 0.265 | 0.459 |
| **5** | 0.66 | 0.744 | 0.774 | 0.8 | 0.872 |
| **10** | 0.696 | 0.8 | 0.838 | 0.873 | 0.955 |
| **20** | 0.737 | 0.852 | 0.904 | 0.933 | 0.991 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 50: Comparison Markrndvisc a=0.8 and Direct Democracy for mu=0.7

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markrndvisc a=0.8 and mu=0.7** | | | | | |
| **0** | 0.567 | 0.749 | 0.817 | 0.882 | 0.983 |
| **1** | 0.626 | 0.779 | 0.846 | 0.891 | 0.978 |
| **5** | 0.713 | 0.831 | 0.88 | 0.92 | 0.988 |
| **10** | 0.749 | 0.855 | 0.902 | 0.938 | 0.994 |
| **20** | 0.782 | 0.879 | 0.922 | 0.951 | 0.997 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 51: Comparison Markmaxvisc a=0.8 and Direct Democracy for mu=0.7

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Markmaxvisc a=0.8 and mu=0.7** | | | | | |
| **0** | 0.941 | 0.967 | 0.98 | 0.988 | 1.0 |
| **1** | 0.933 | 0.965 | 0.979 | 0.989 | 1.0 |
| **5** | 0.896 | 0.948 | 0.969 | 0.983 | 1.0 |
| **10** | 0.898 | 0.946 | 0.966 | 0.981 | 1.0 |
| **20** | 0.879 | 0.937 | 0.962 | 0.978 | 1.0 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

Table 52: Comparison Campbellvisc a=0.8 and Direct Democracy for mu=0.7

| **Diffusion stage** | **Low whisker** | **Q2** | **Median** | **Q3** | **High whisker** |
|---|---|---|---|---|---|
| **Campbellvisc a=0.8 and mu=0.7** | | | | | |
| **0** | 0.291 | 0.389 | 0.433 | 0.475 | 0.59 |
| **1** | 0.29 | 0.458 | 0.517 | 0.603 | 0.693 |
| **5** | 0.308 | 0.595 | 0.759 | 0.808 | 0.901 |
| **10** | 0.609 | 0.762 | 0.83 | 0.872 | 0.962 |
| **20** | 0.592 | 0.794 | 0.888 | 0.931 | 0.992 |
| **Direct Democracy mu=0.7** | | | | | |
| **0** | 0.596 | 0.631 | 0.645 | 0.664 | 0.692 |
| **1** | 0.615 | 0.675 | 0.693 | 0.715 | 0.762 |
| **5** | 0.668 | 0.745 | 0.772 | 0.798 | 0.86 |
| **10** | 0.721 | 0.812 | 0.848 | 0.887 | 0.958 |
| **20** | 0.773 | 0.869 | 0.907 | 0.932 | 0.991 |

## 9.2 B: Graphs

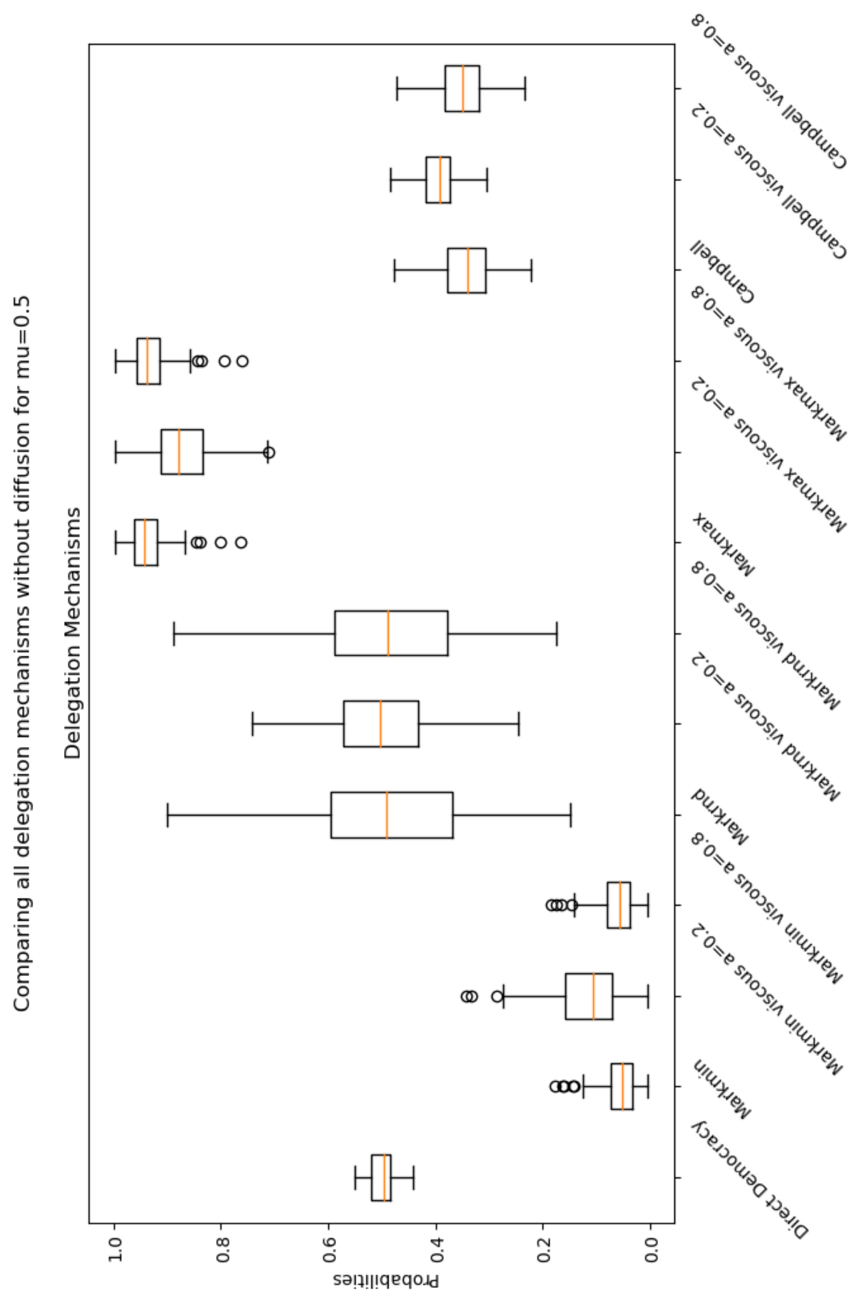


Figure 2:

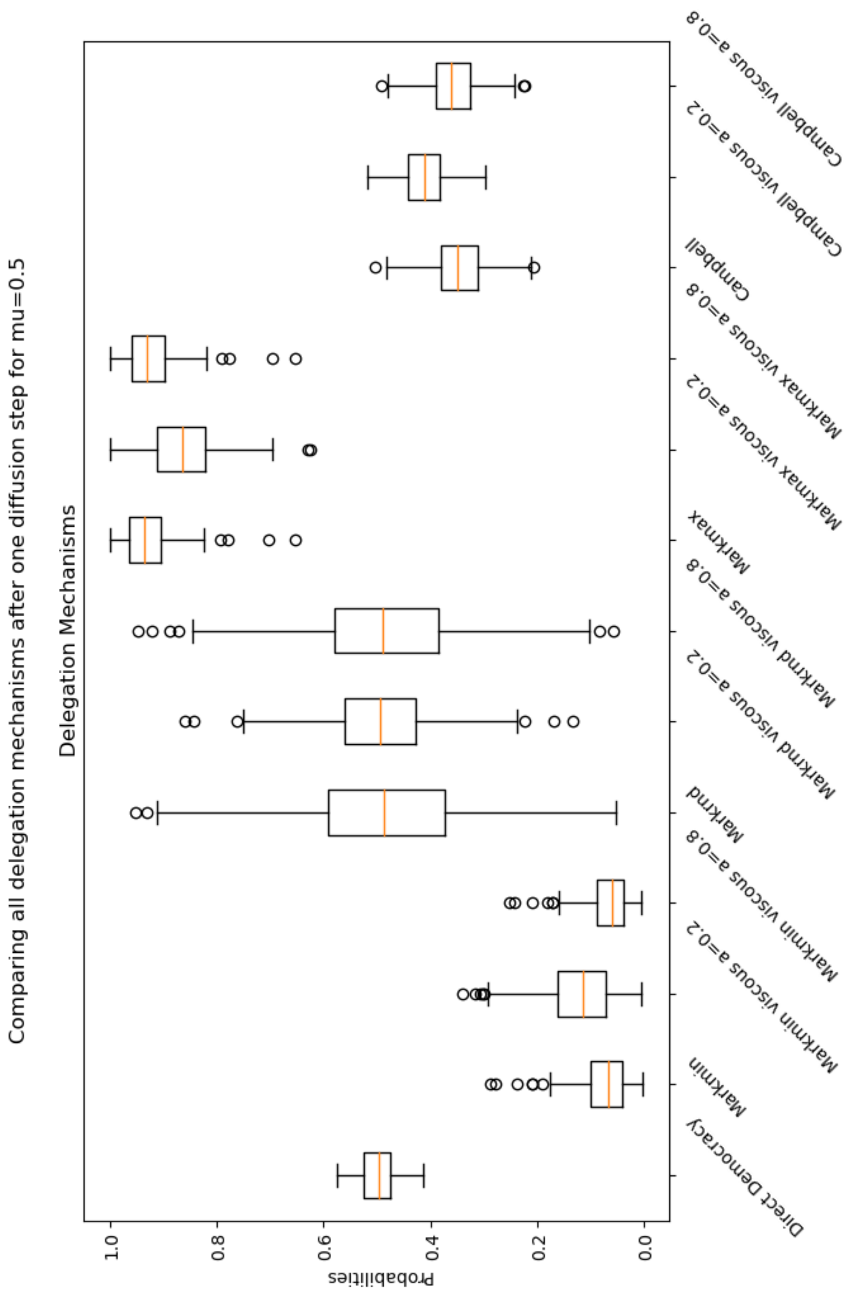

Comparing all delegation mechanisms after one diffusion step for mu=0.5
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 3:

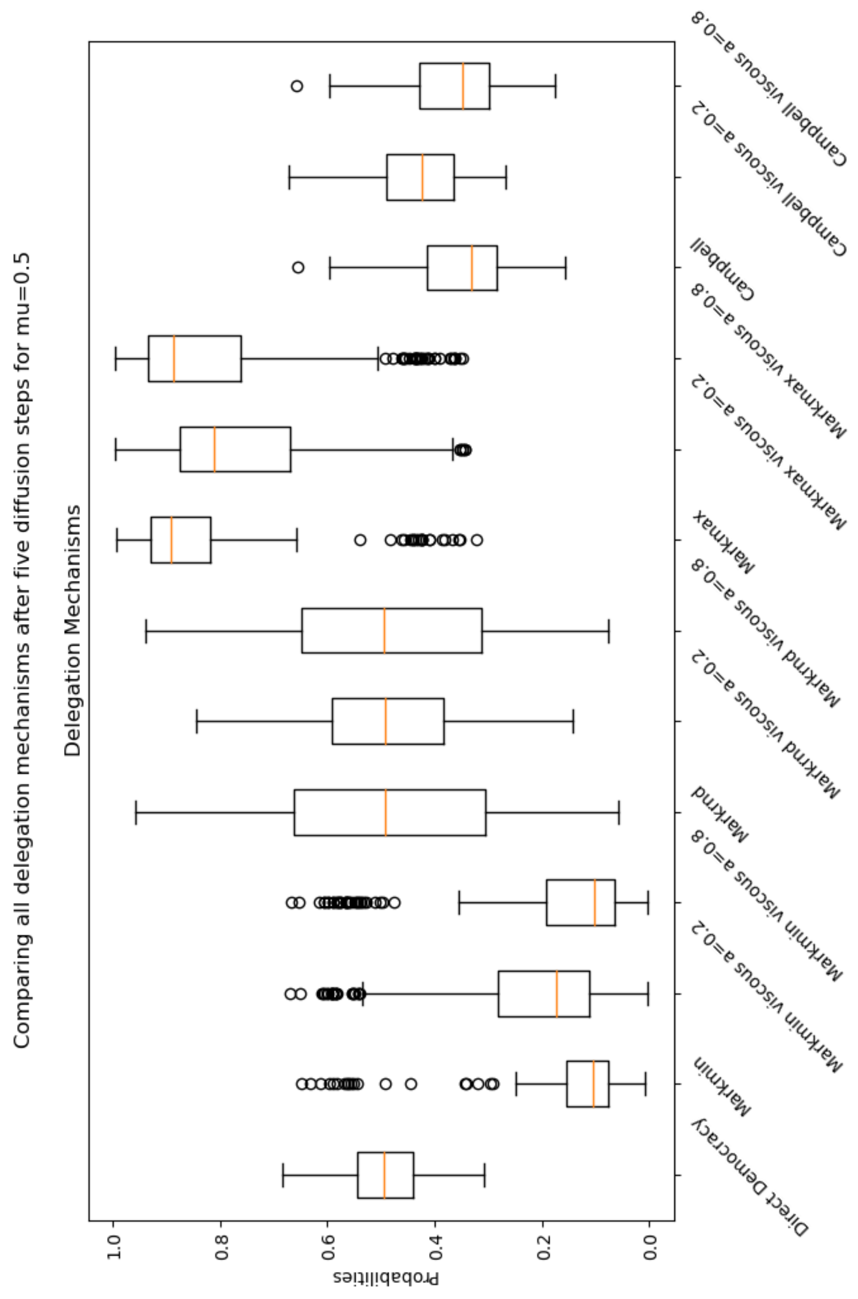

Comparing all delegation mechanisms after five diffusion steps for mu=0.5
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 4:

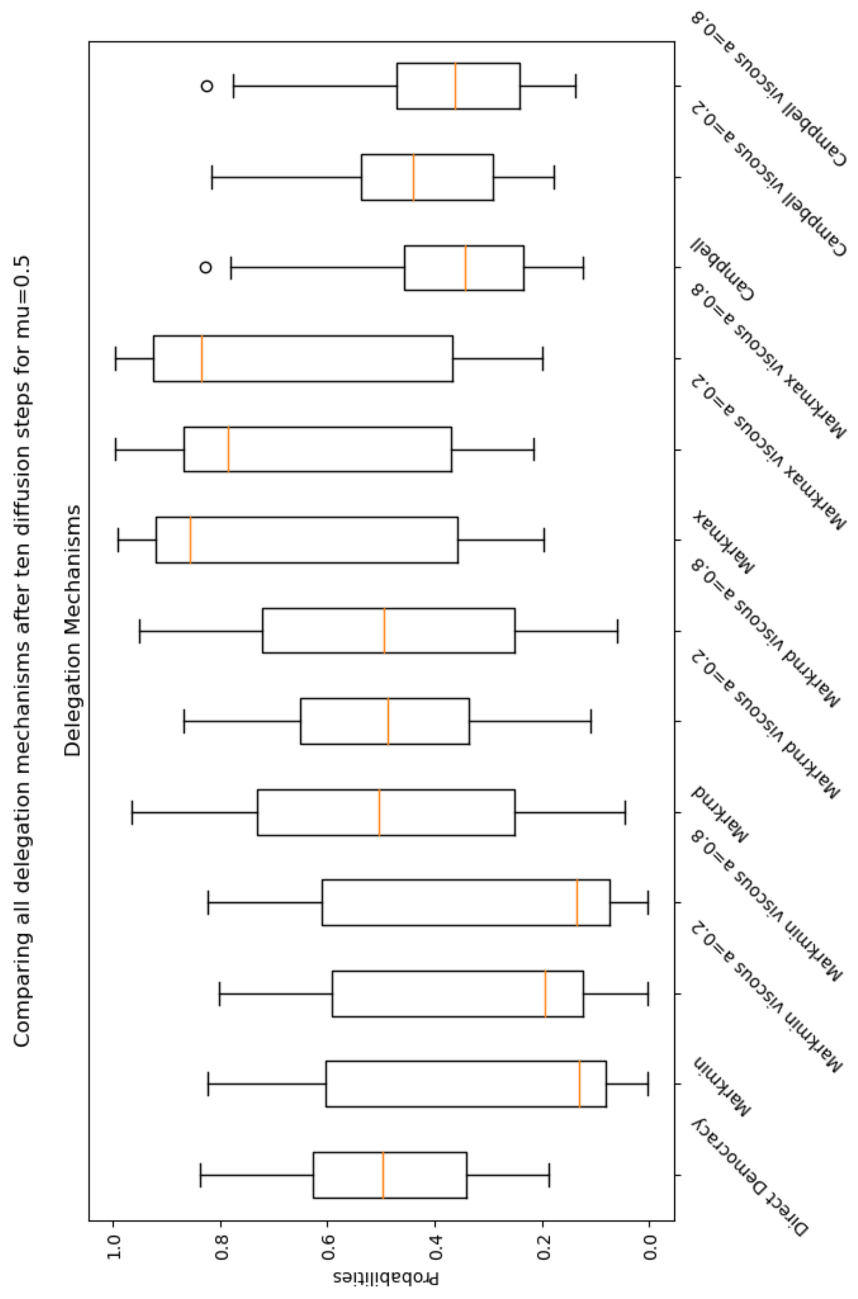

Comparing all delegation mechanisms after ten diffusion steps for mu=0.5
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 5:

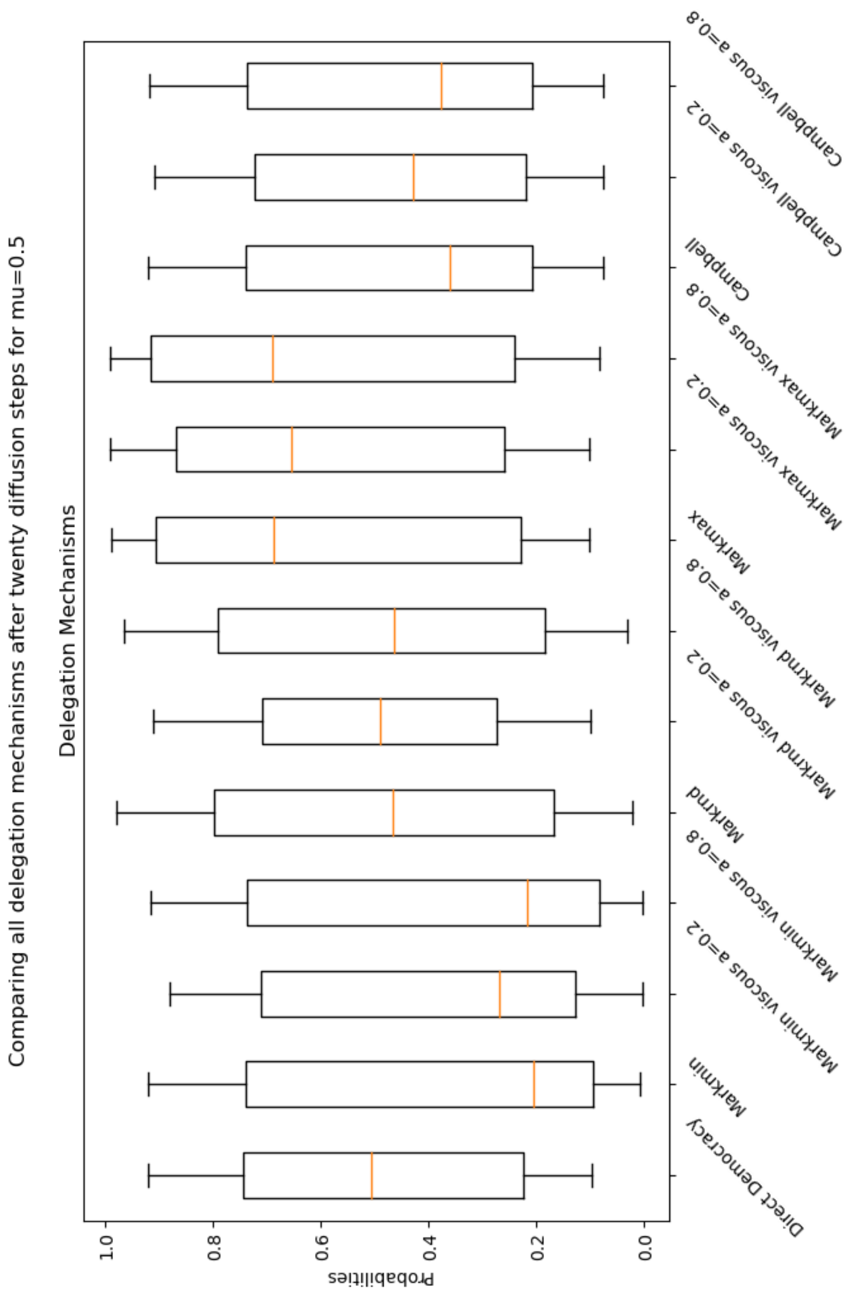
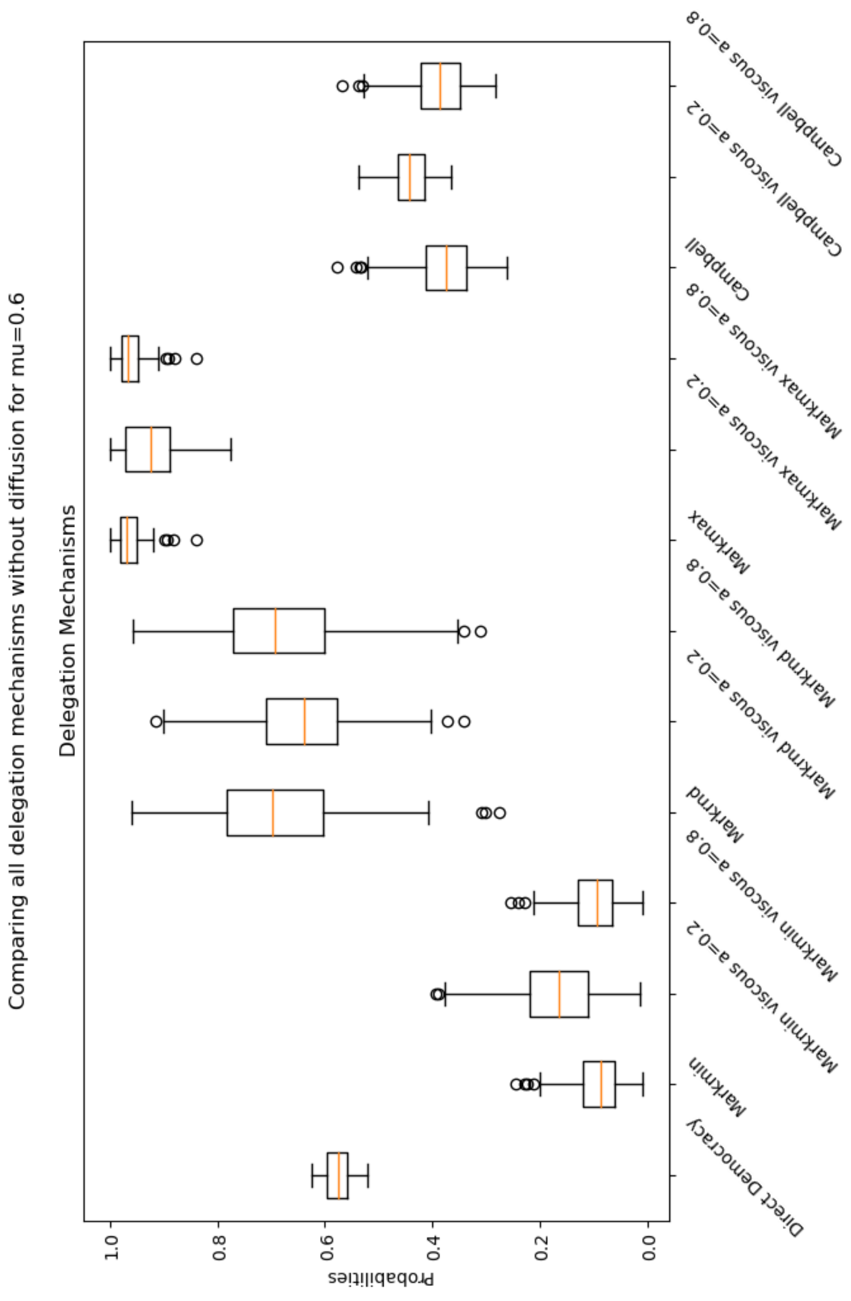

Comparing all delegation mechanisms after twenty diffusion steps for mu=0.5
Delegation Mechanisms
Probabilities
0.0
0.2
0.4
0.6
0.8
1.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 6:



Comparing all delegation mechanisms without diffusion for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 7:

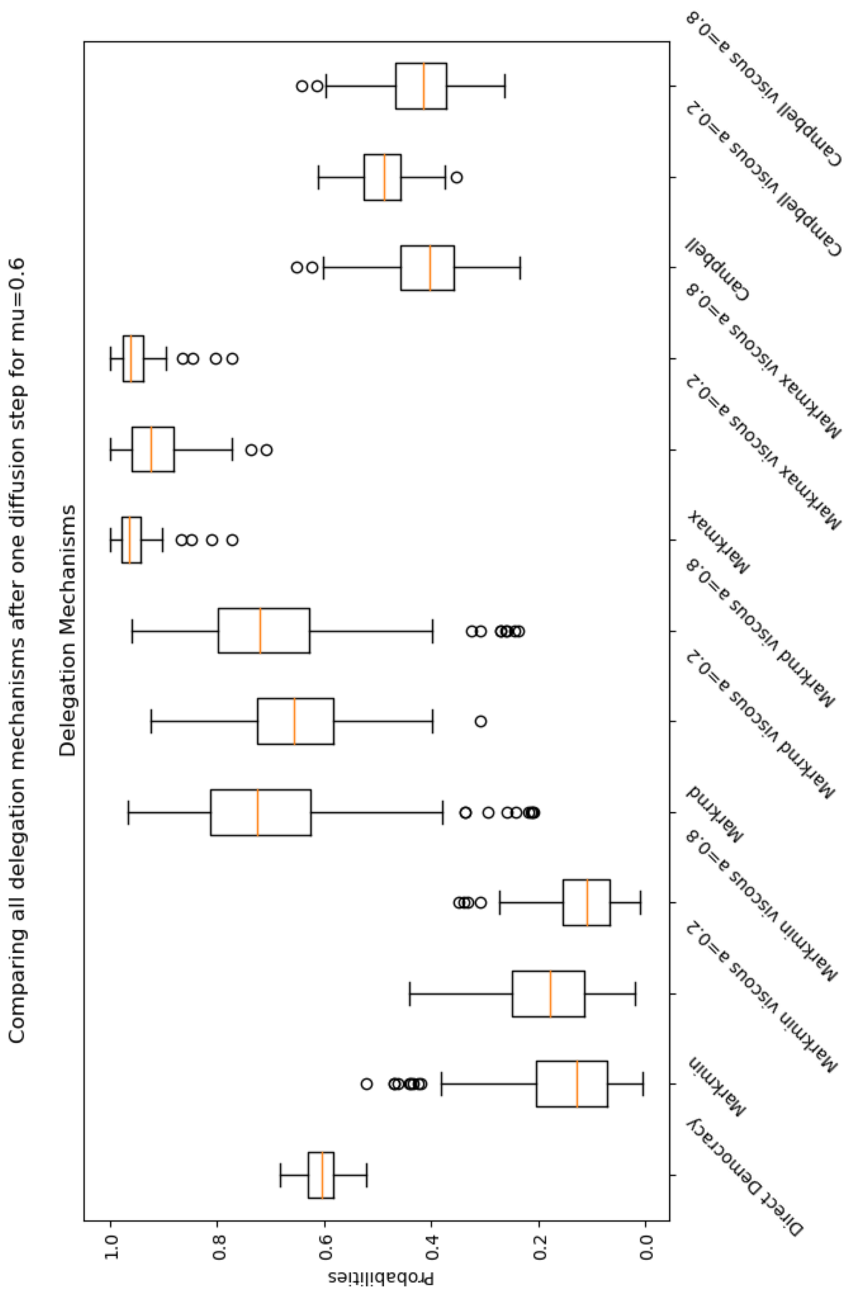

Comparing all delegation mechanisms after one diffusion step for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 8:

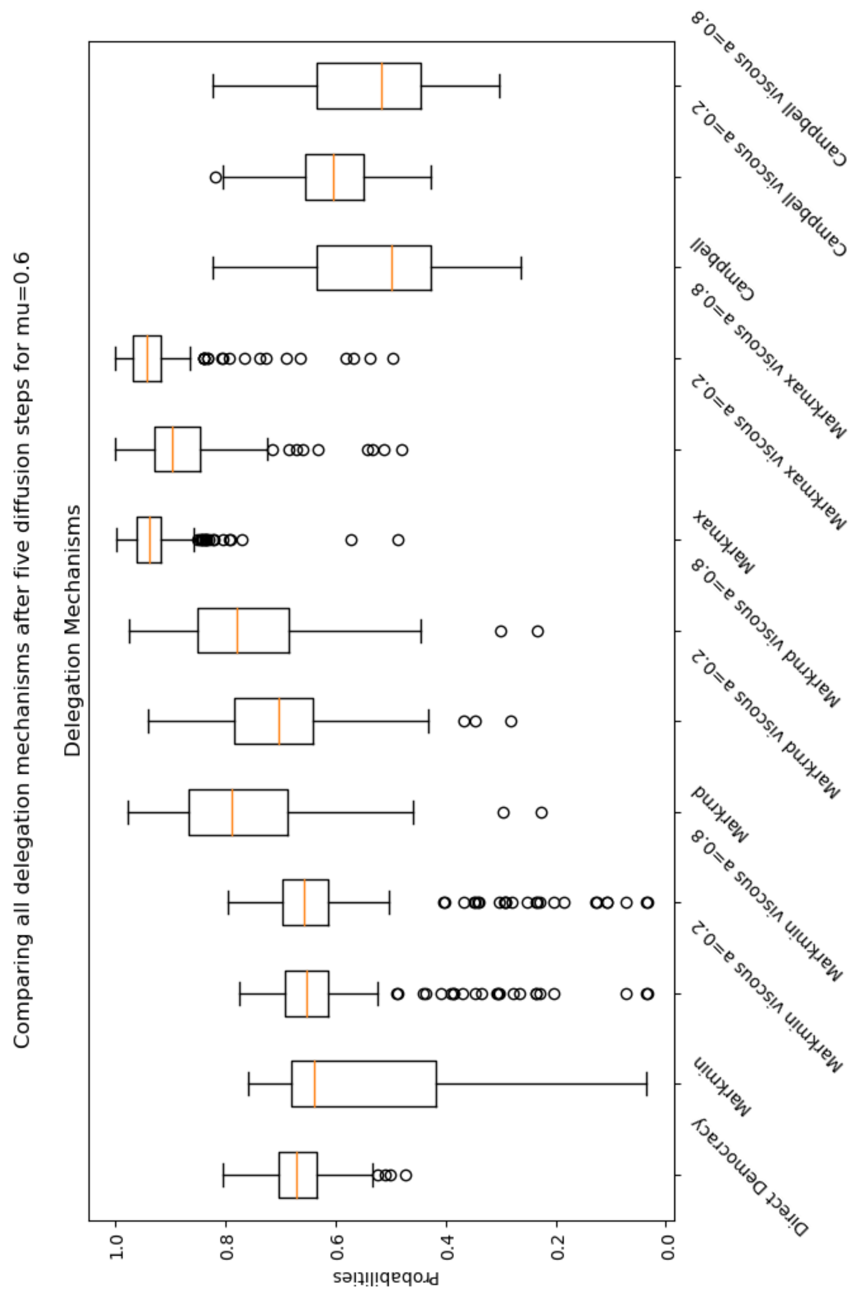

Comparing all delegation mechanisms after five diffusion steps for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 9:

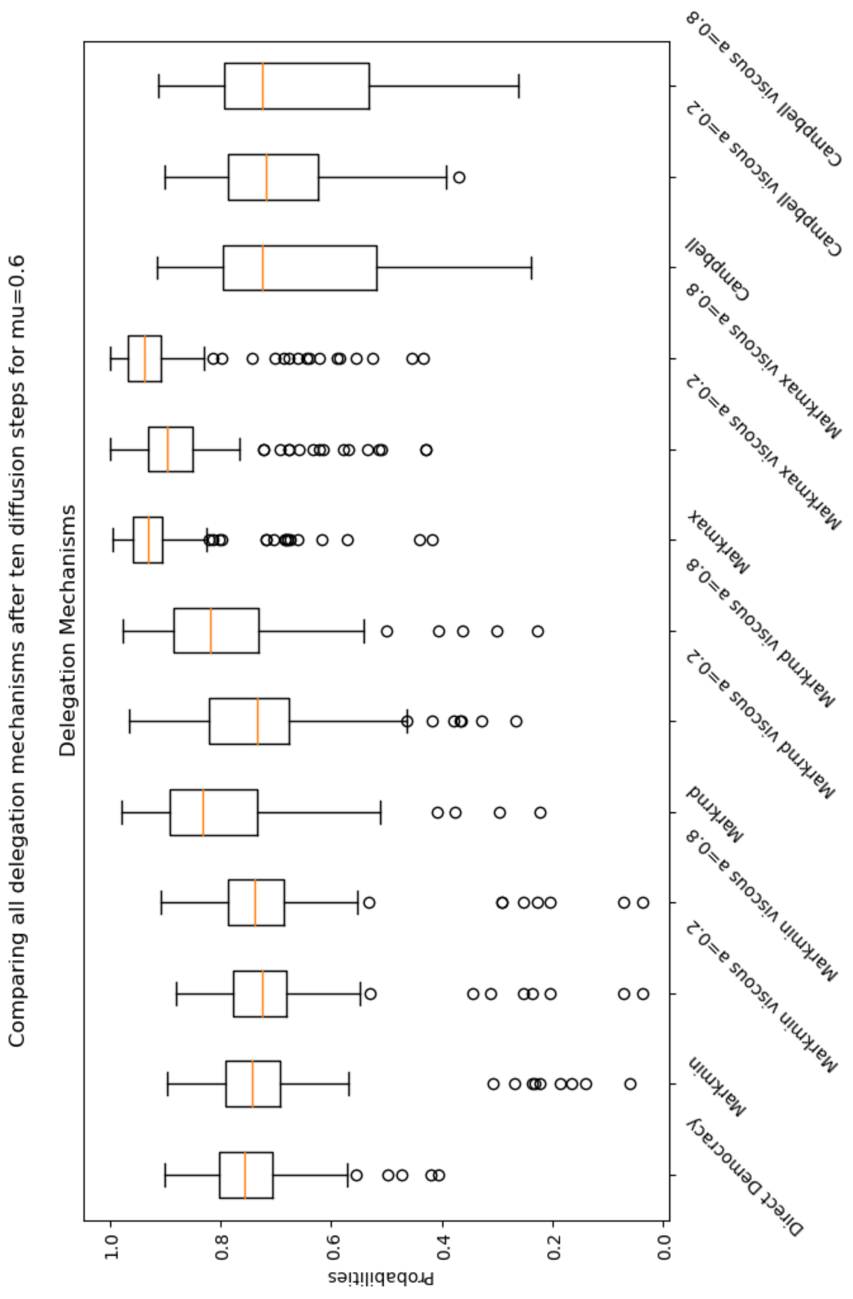
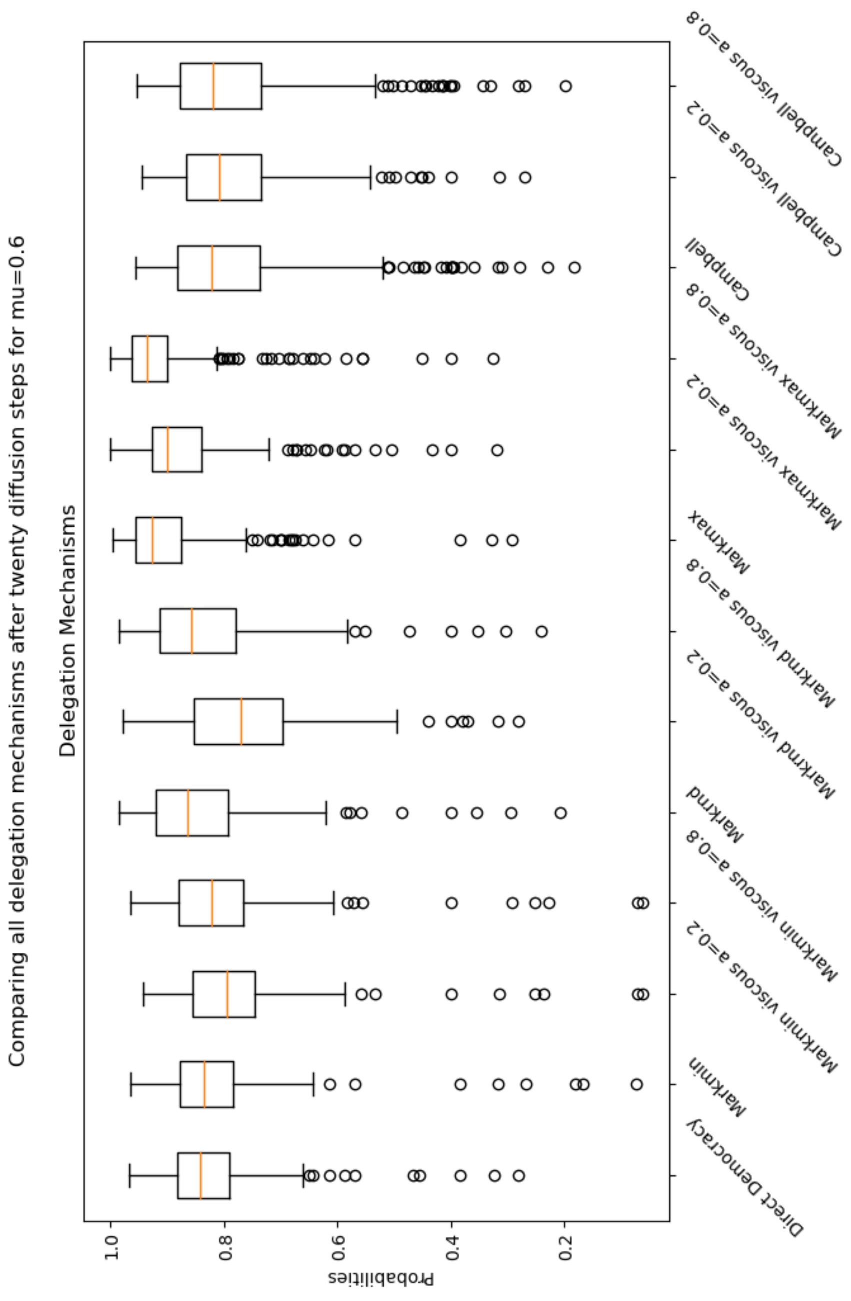

Comparing all delegation mechanisms after ten diffusion steps for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 10:



Comparing all delegation mechanisms after twenty diffusion steps for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 11:

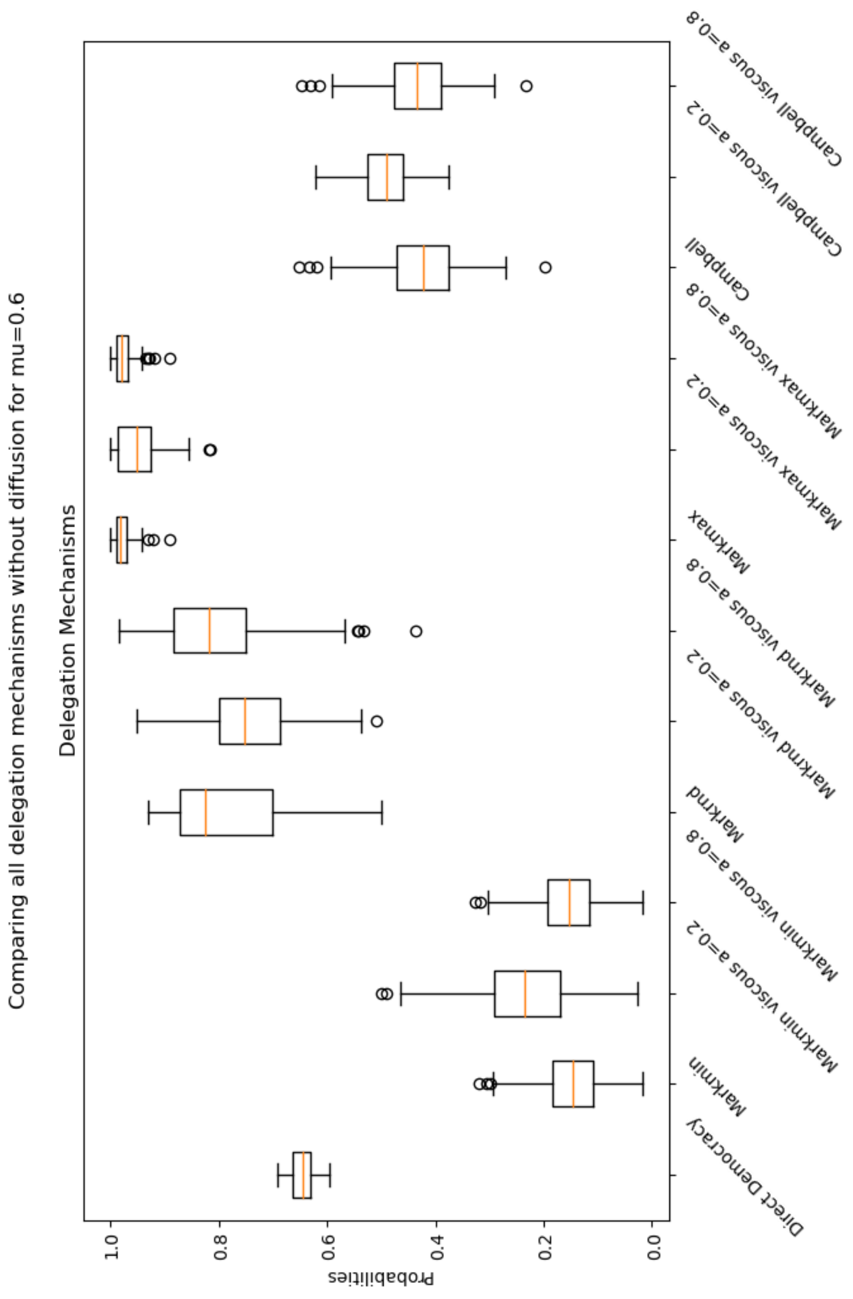

Comparing all delegation mechanisms without diffusion for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 12:

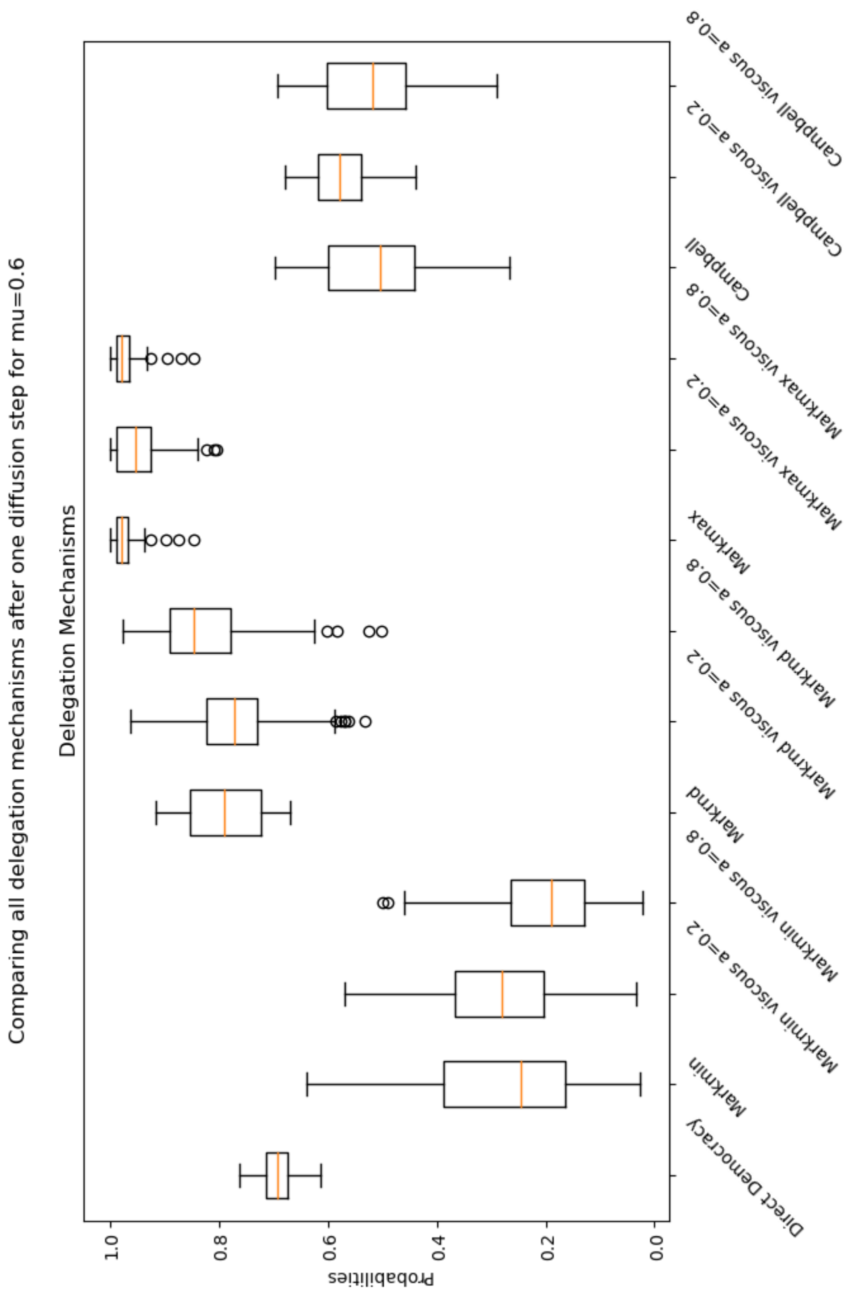

Comparing all delegation mechanisms after one diffusion step for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
0.0
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 13:

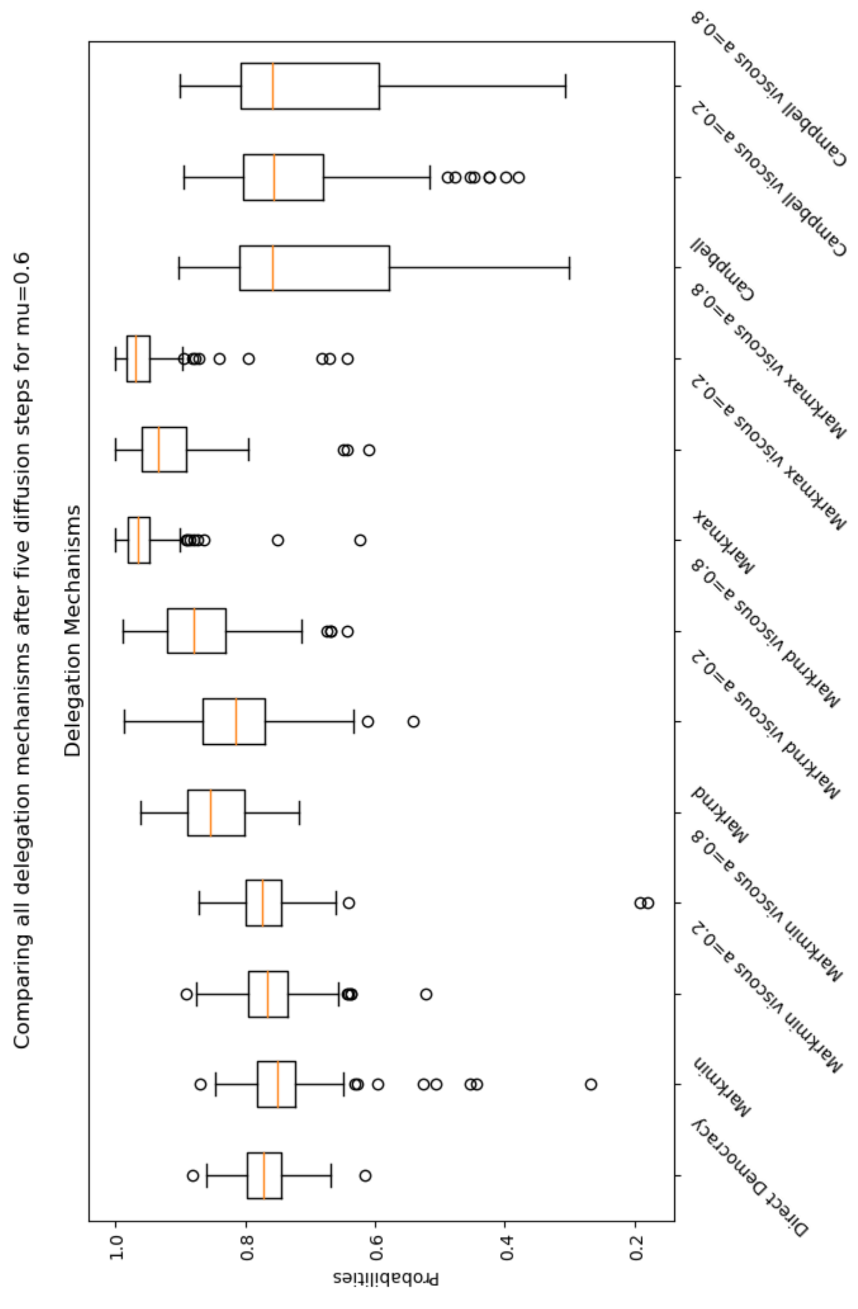

Comparing all delegation mechanisms after five diffusion steps for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.8
0.6
0.4
0.2
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 14:

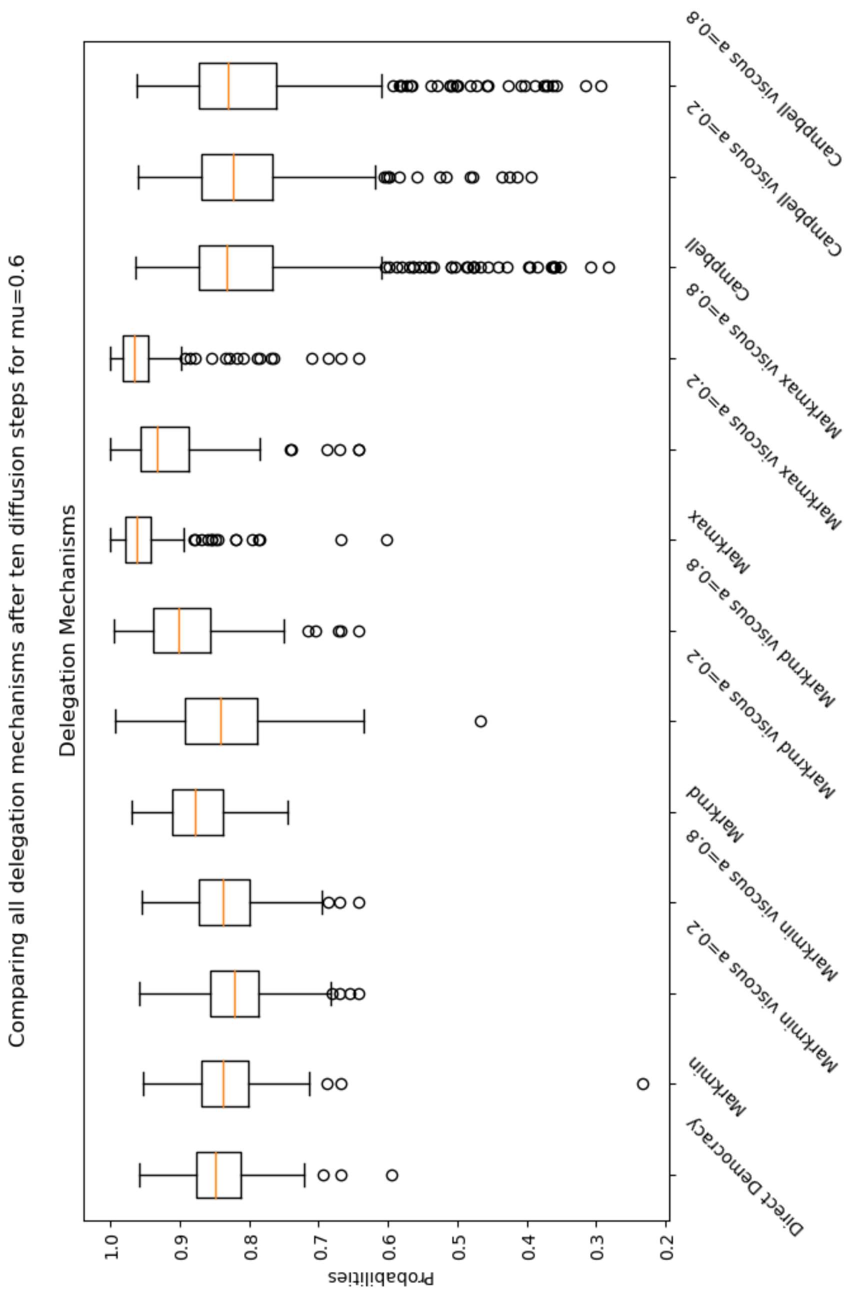

Comparing all delegation mechanisms after ten diffusion steps for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 15:

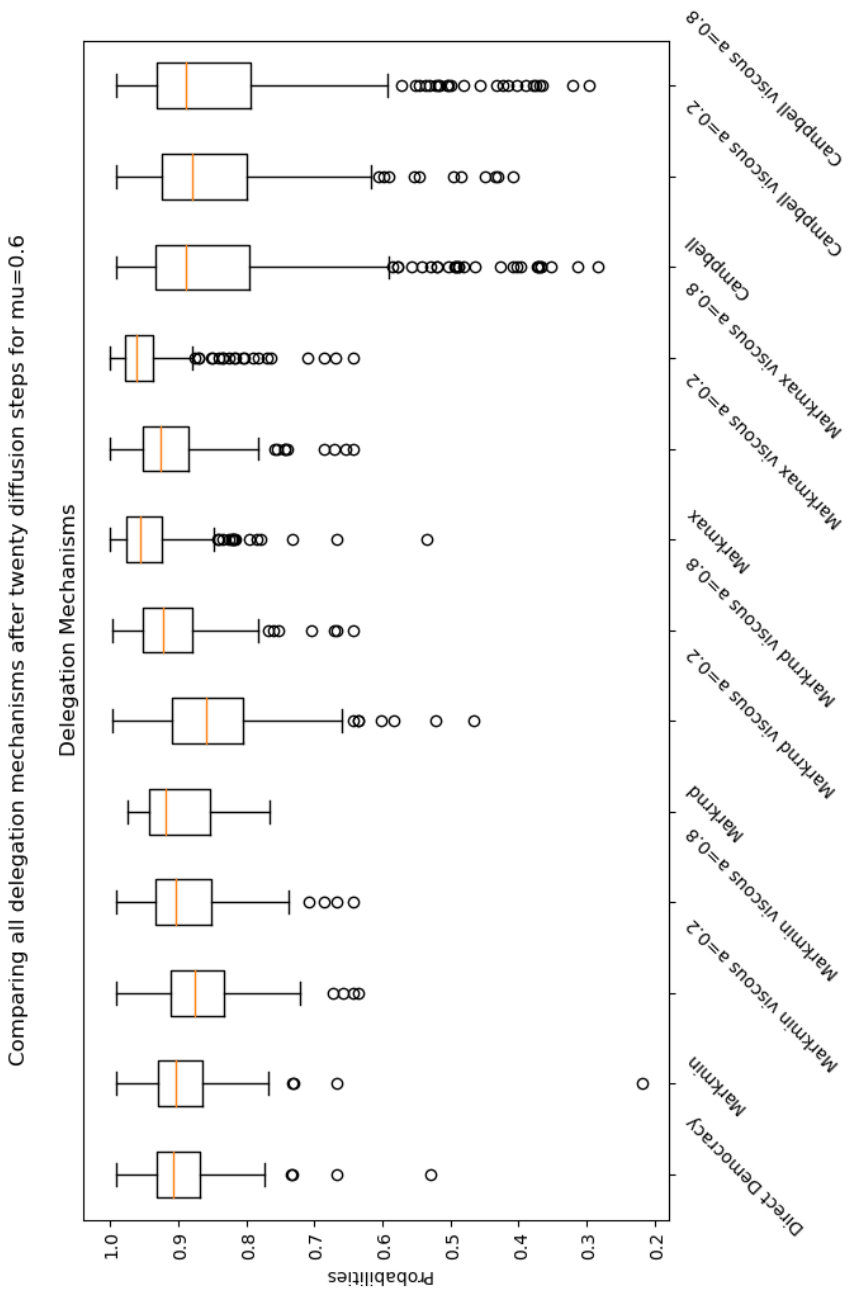

Comparing all delegation mechanisms after twenty diffusion steps for mu=0.6
Delegation Mechanisms
Probabilities
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Direct Democracy
Markmin
Markmin viscous a=0.2
Markmin viscous a=0.8
Markrnd
Markrnd viscous a=0.2
Markrnd viscous a=0.8
Markmax
Markmax viscous a=0.2
Markmax viscous a=0.8
Campbell
Campbell viscous a=0.2
Campbell viscous a=0.8

Figure 16:

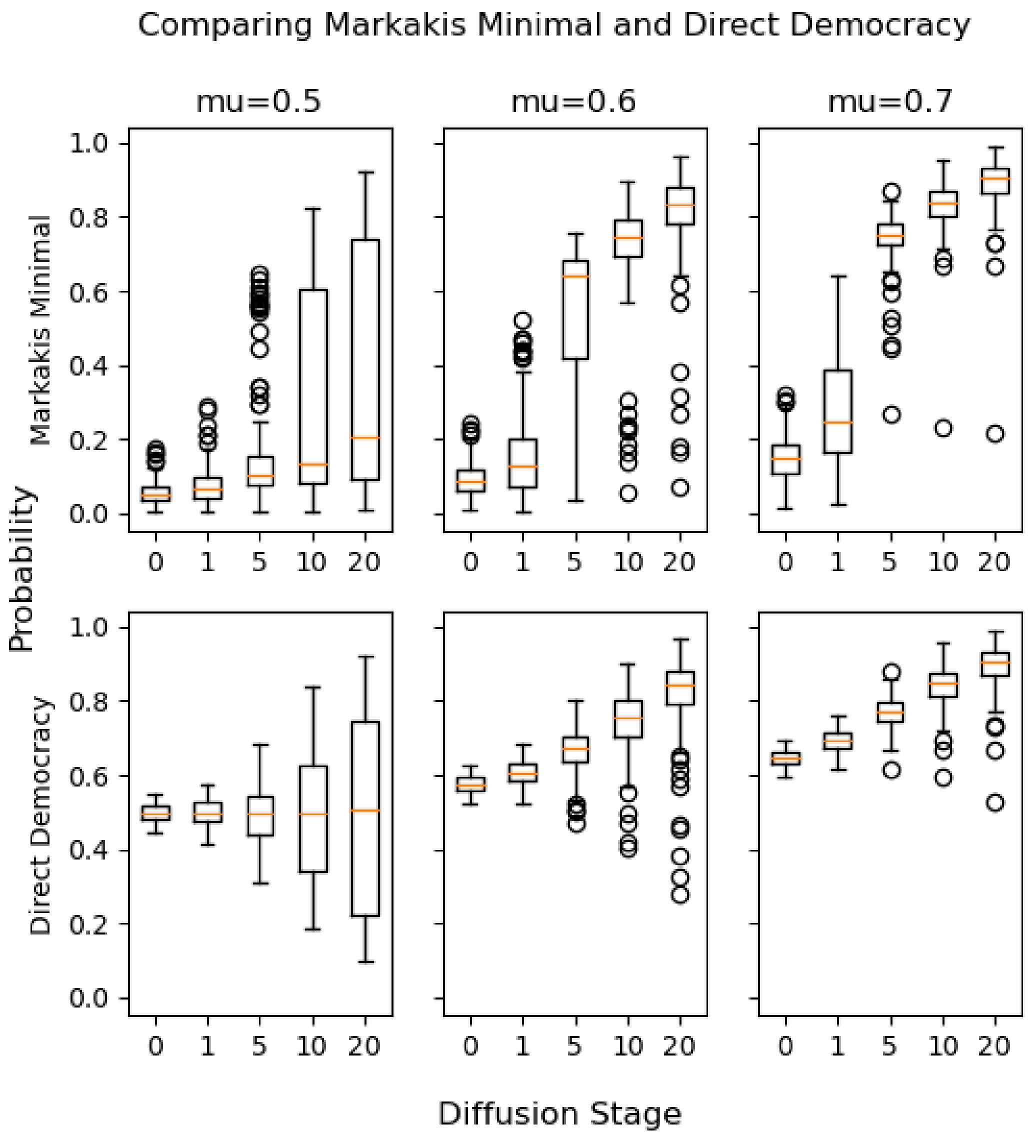

Comparing Markakis Minimal and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Probability
Markakis Minimal
Direct Democracy
Diffusion Stage
0.0
0.2
0.4
0.6
0.8
1.0
0
1
5
10
20

Figure 17:

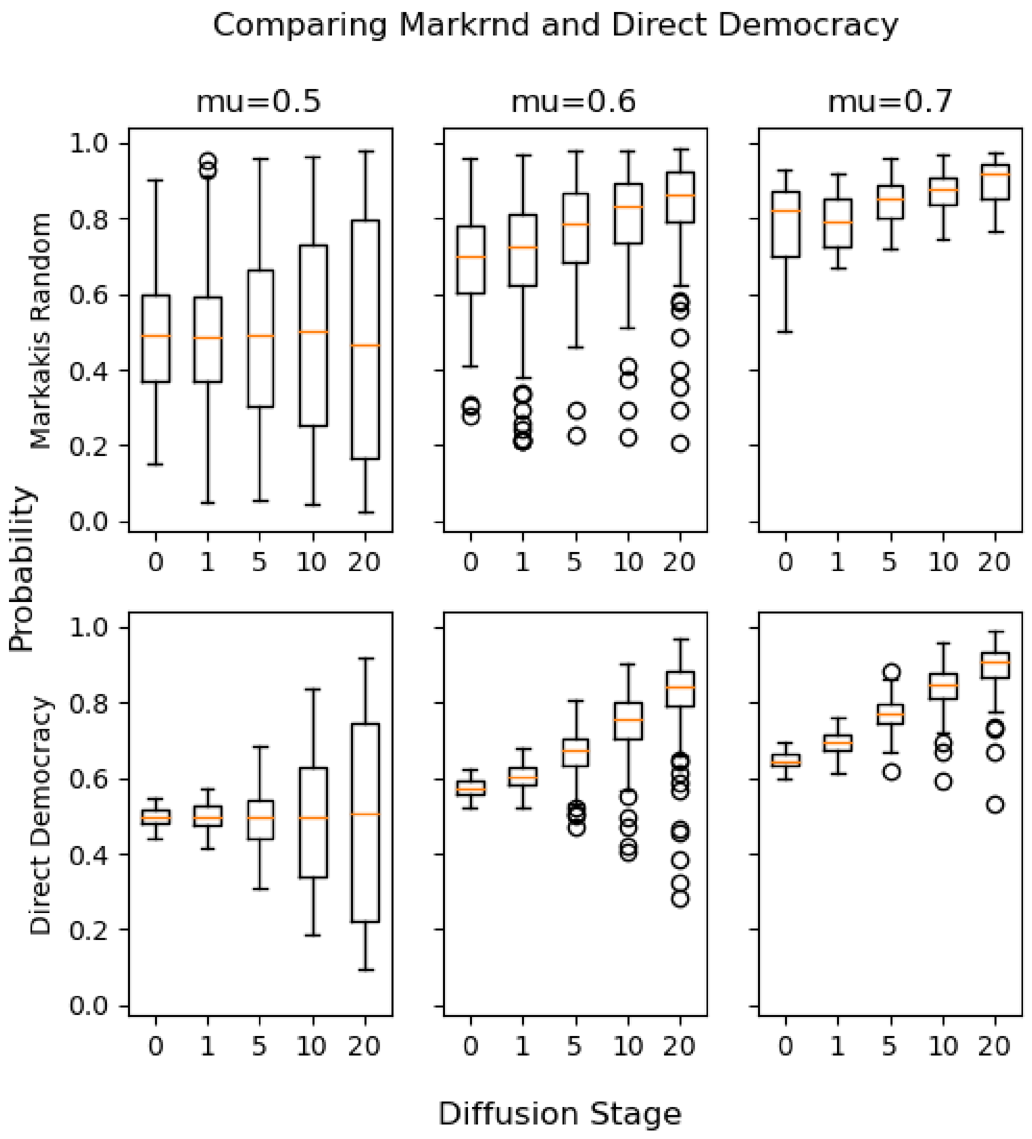

Comparing Markrnd and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Probability
Markakis Random
Direct Democracy
1.0
0.8
0.6
0.4
0.2
0.0
0
1
5
10
20
Diffusion Stage

Figure 18:

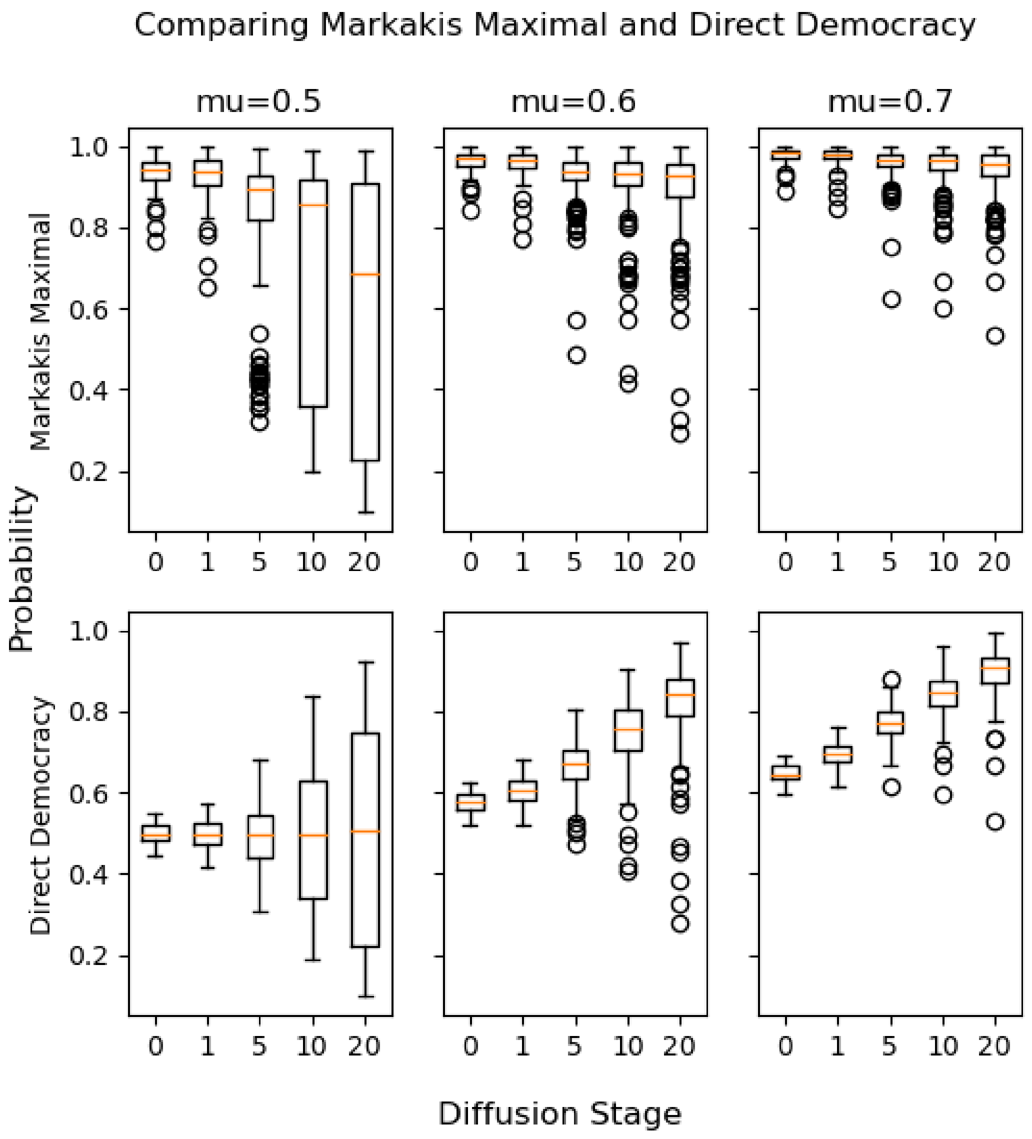
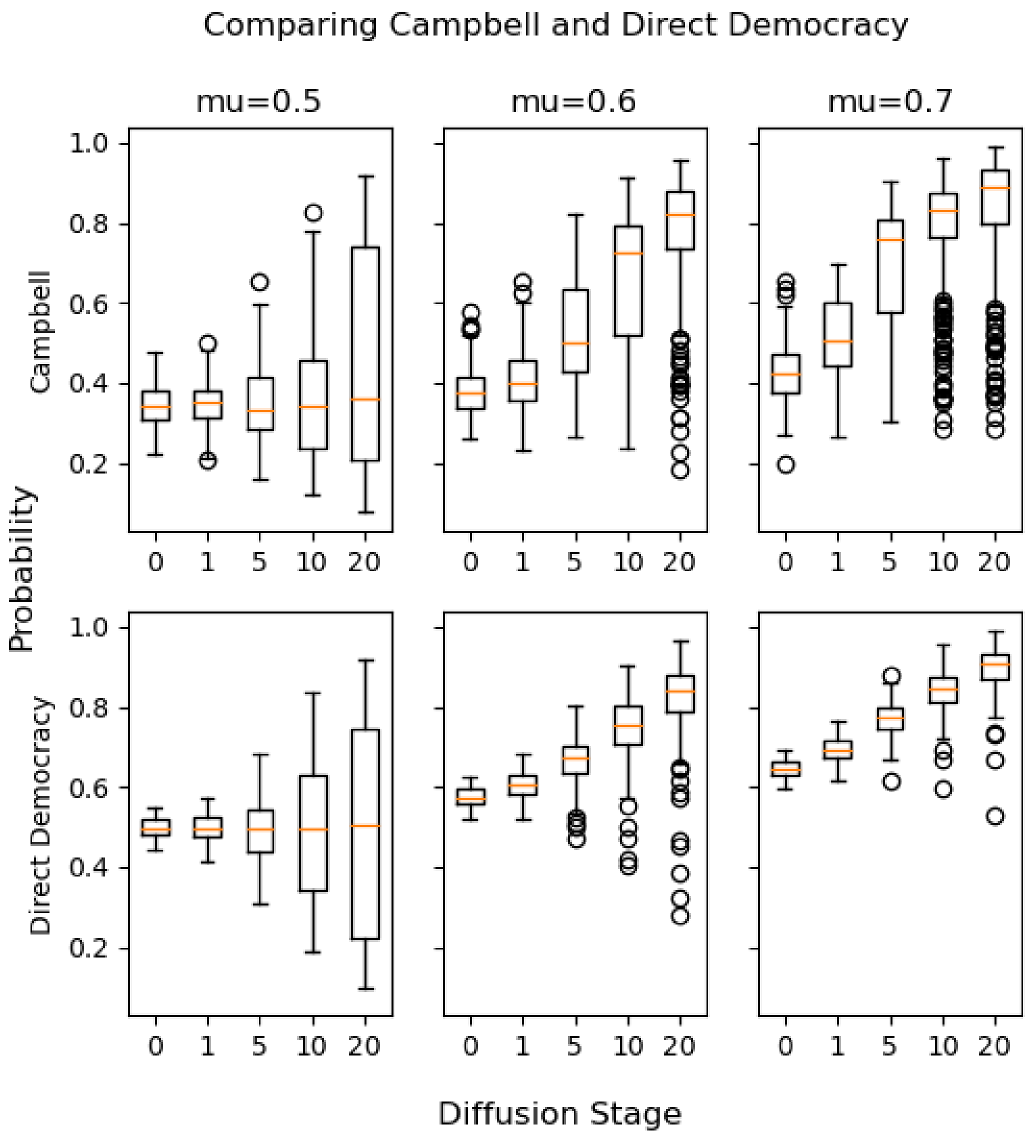

Comparing Markakis Maximal and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Probability
Markakis Maximal
Direct Democracy
1.0
0.8
0.6
0.4
0.2
0
1
5
10
20
Diffusion Stage

Figure 19:



Comparing Campbell and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Campbell
Direct Democracy
Probability
1.0
0.8
0.6
0.4
0.2
0
1
5
10
20
Diffusion Stage

Figure 20:

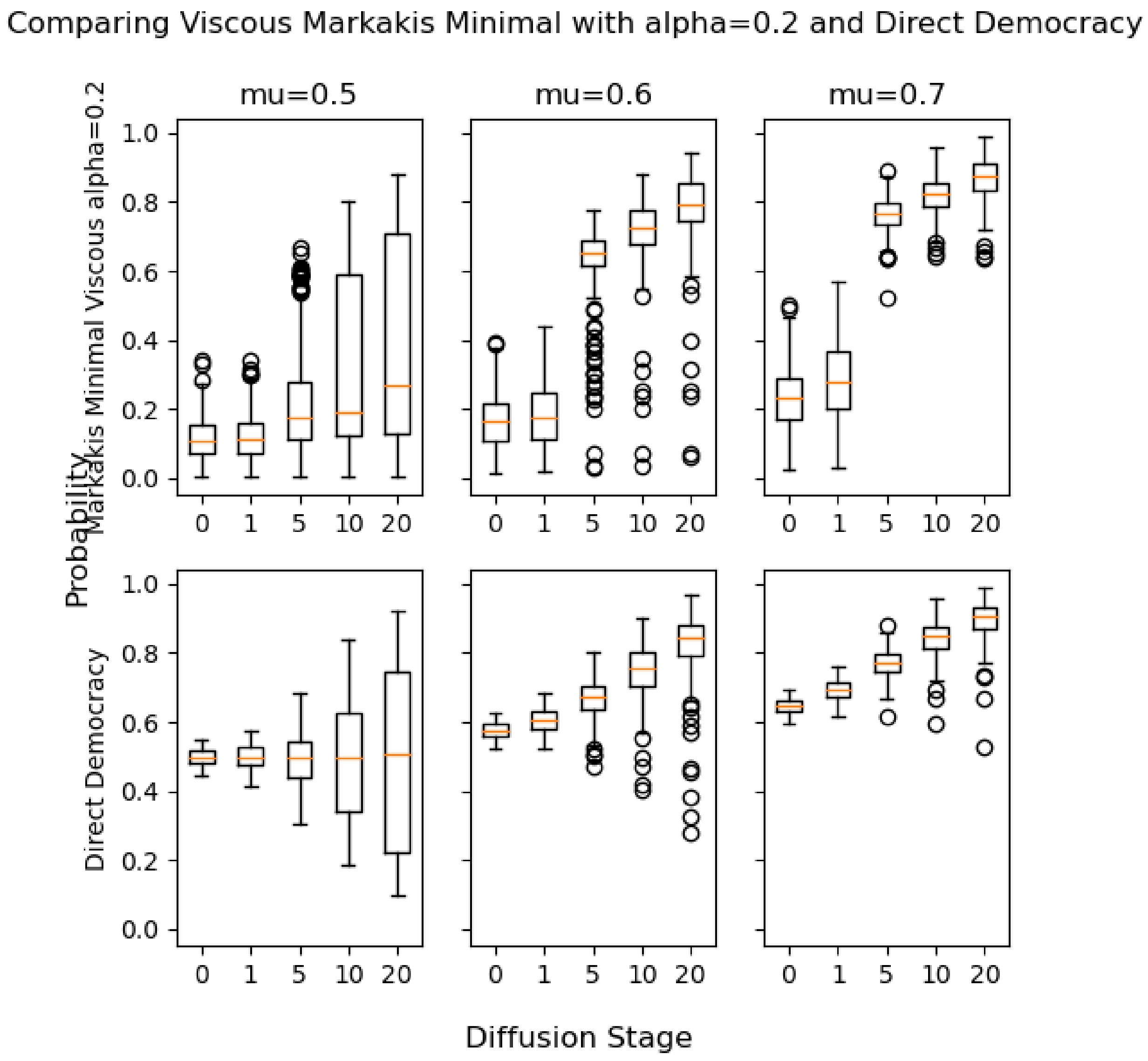

Comparing Viscous Markakis Minimal with alpha=0.2 and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Markakis Minimal Viscous alpha=0.2
Direct Democracy
Probability
Diffusion Stage
0.0
0.2
0.4
0.6
0.8
1.0
0
1
5
10
20

Figure 21:

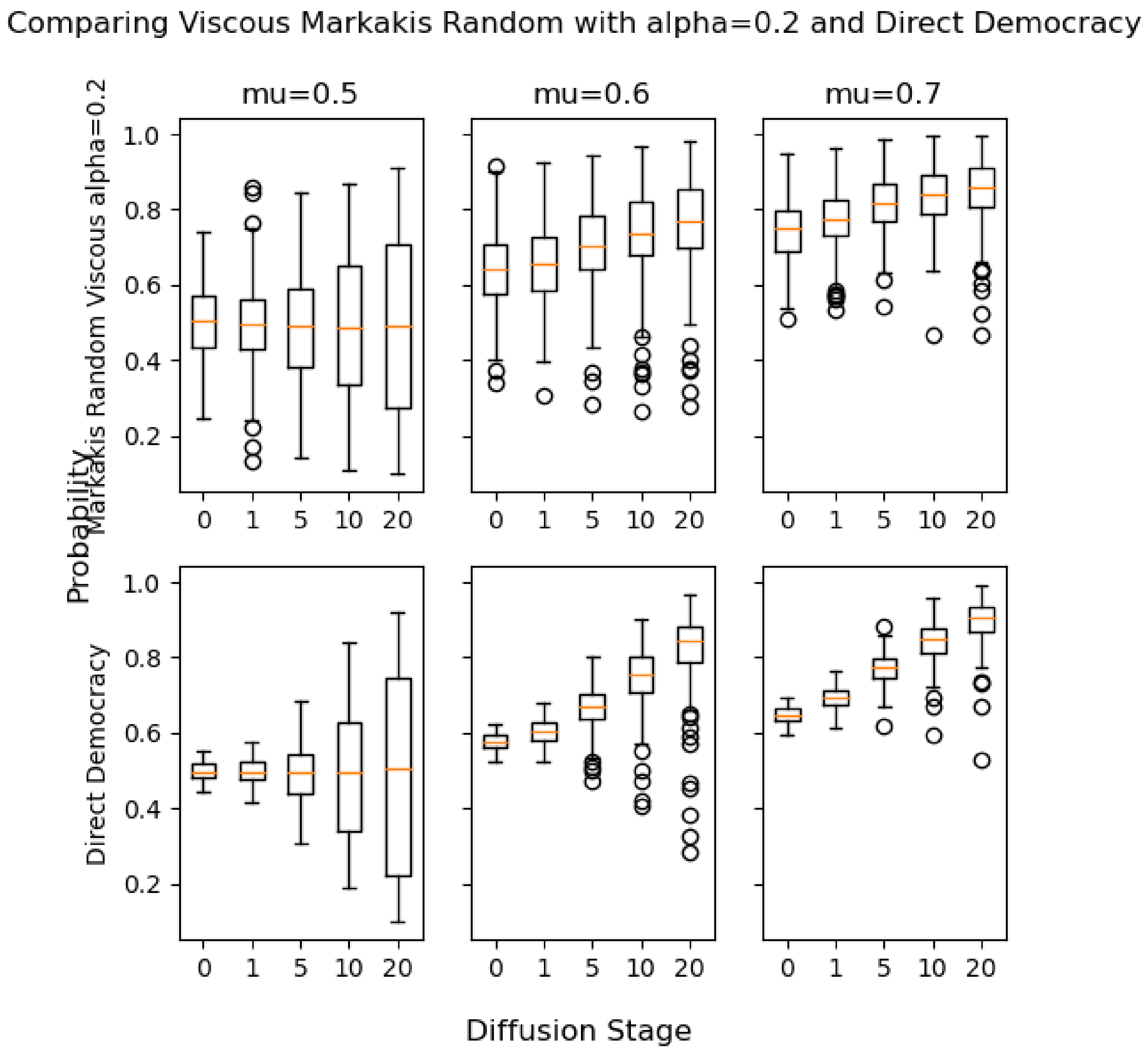

Comparing Viscous Markakis Random with alpha=0.2 and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Probability
Markakis Random Viscous alpha=0.2
Direct Democracy
1.0
0.8
0.6
0.4
0.2
0
1
5
10
20
Diffusion Stage

Figure 22:

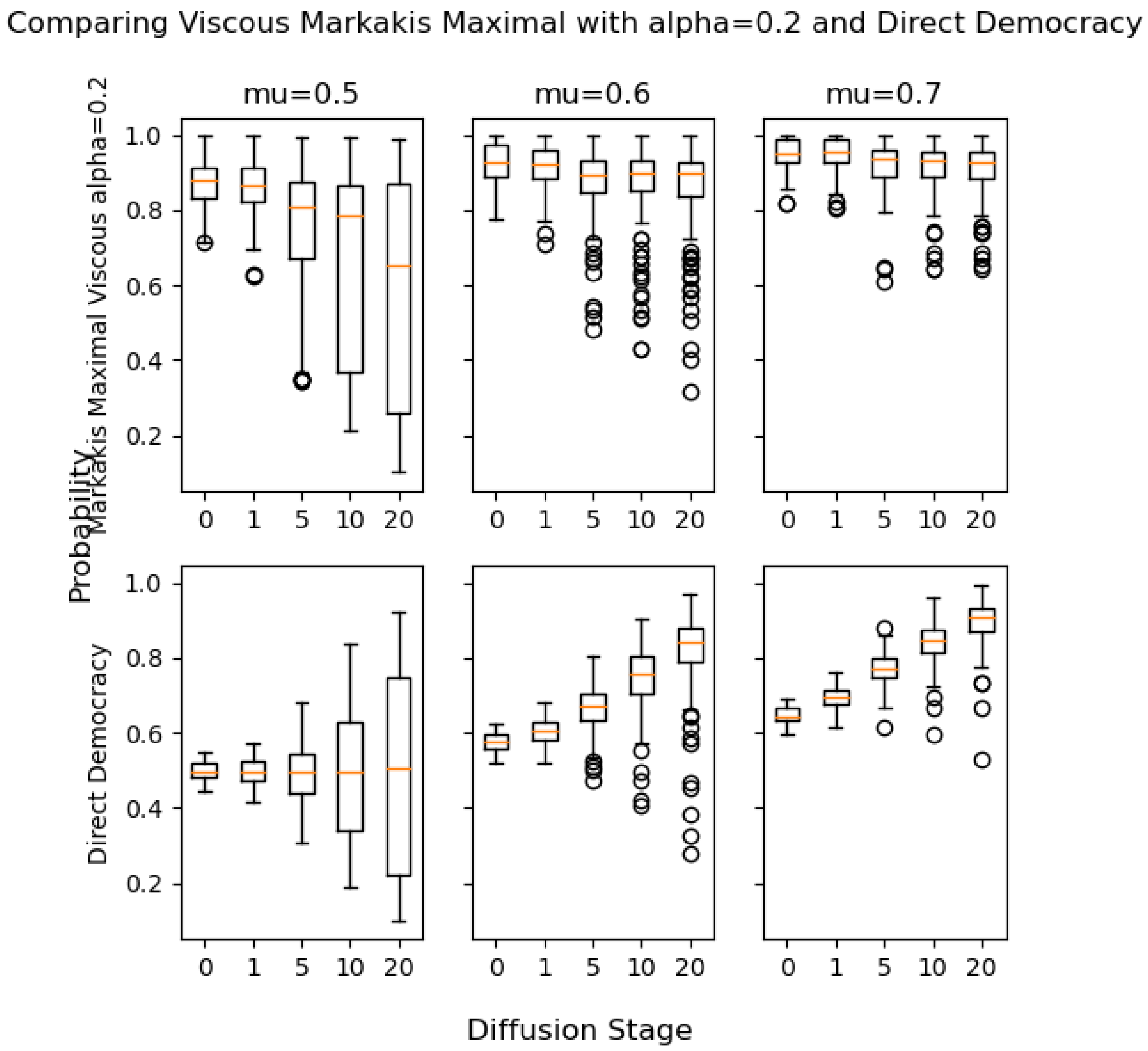

Comparing Viscous Markakis Maximal with alpha=0.2 and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Probability
Markakis Maximal Viscous alpha=0.2
Direct Democracy
Diffusion Stage
0
1
5
10
20
0.2
0.4
0.6
0.8
1.0

Figure 23:

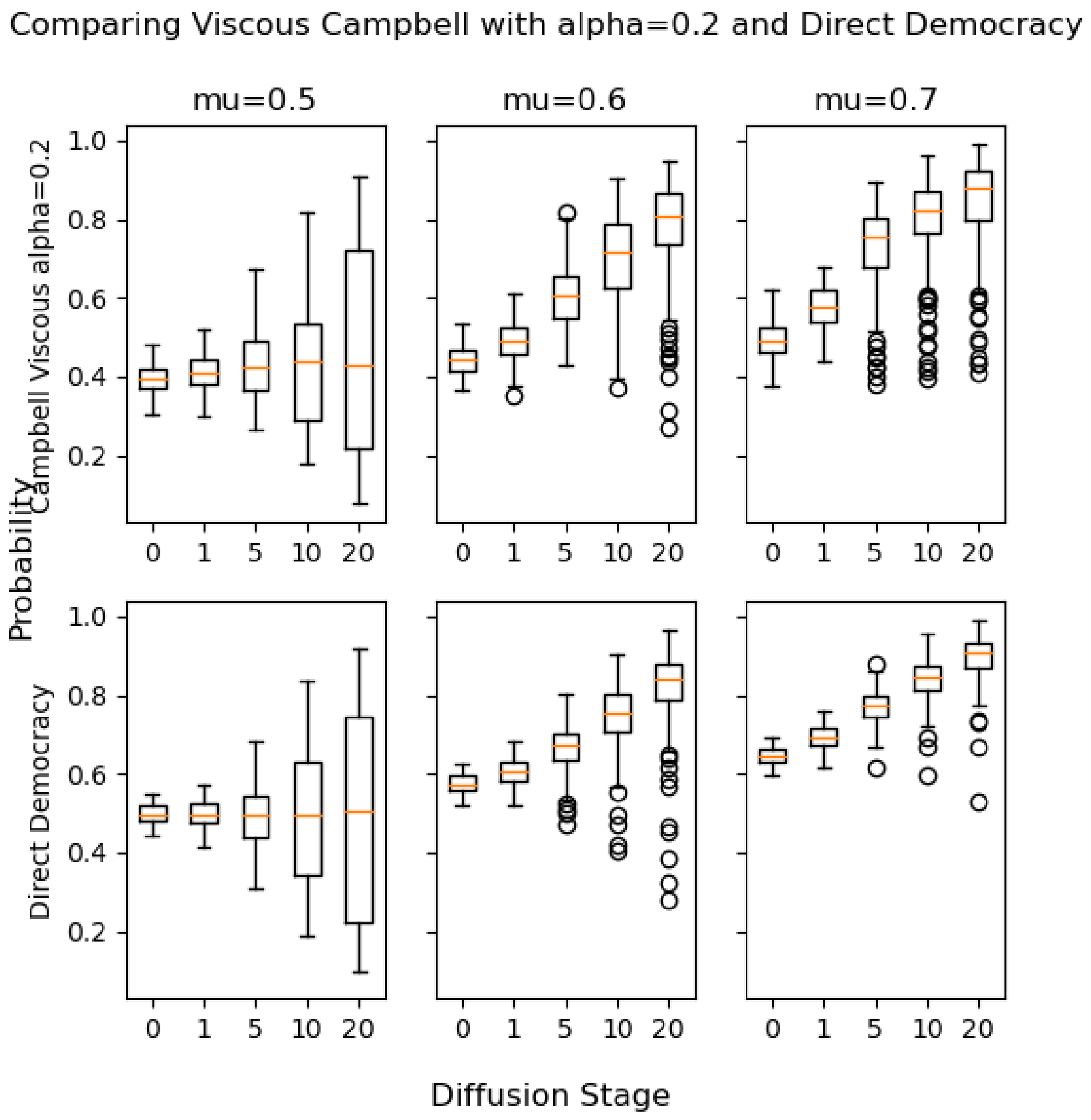

Comparing Viscous Campbell with alpha=0.2 and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Probability
Campbell Viscous alpha=0.2
Direct Democracy
1.0
0.8
0.6
0.4
0.2
0
1
5
10
20
Diffusion Stage

Figure 24:

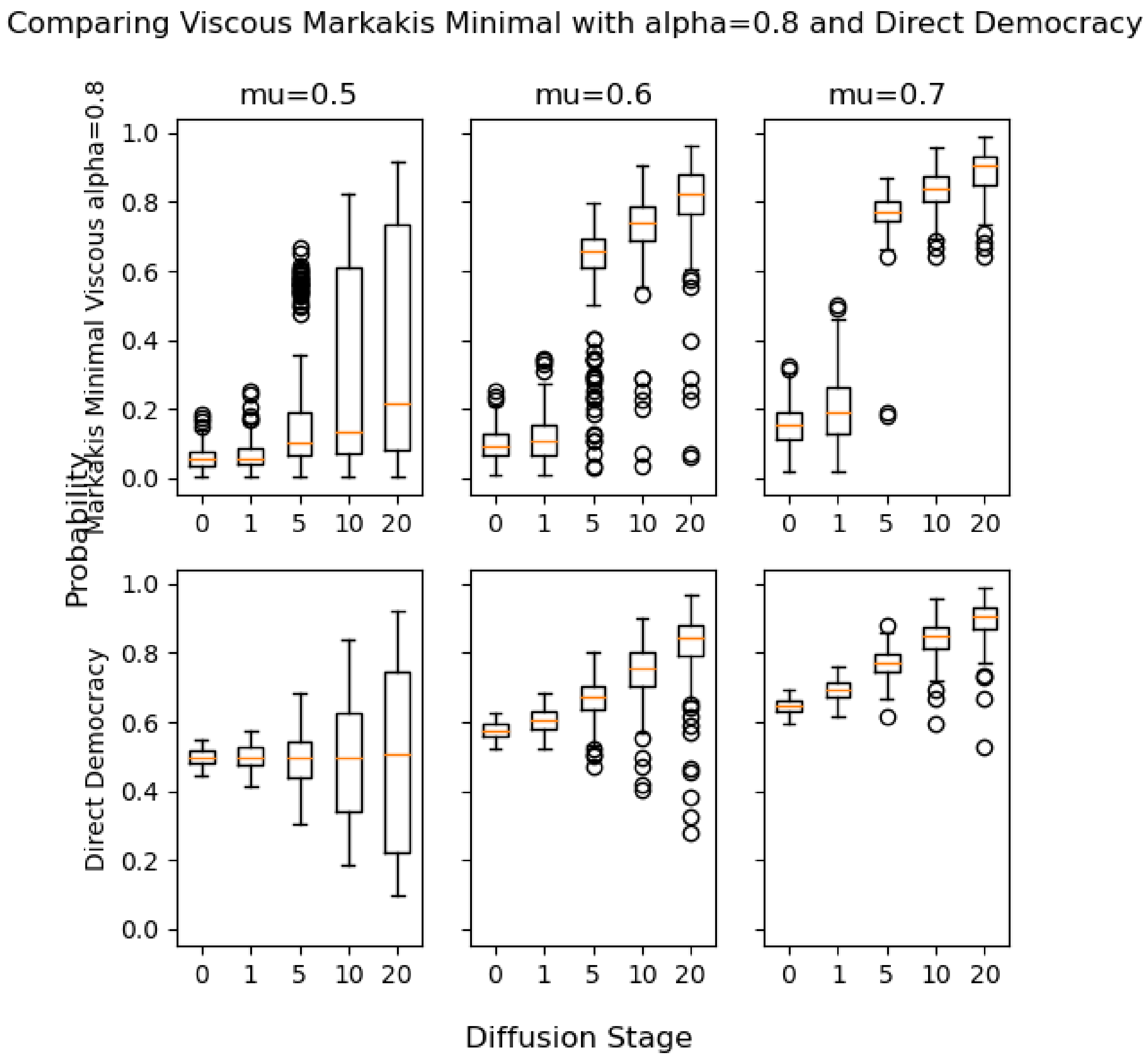
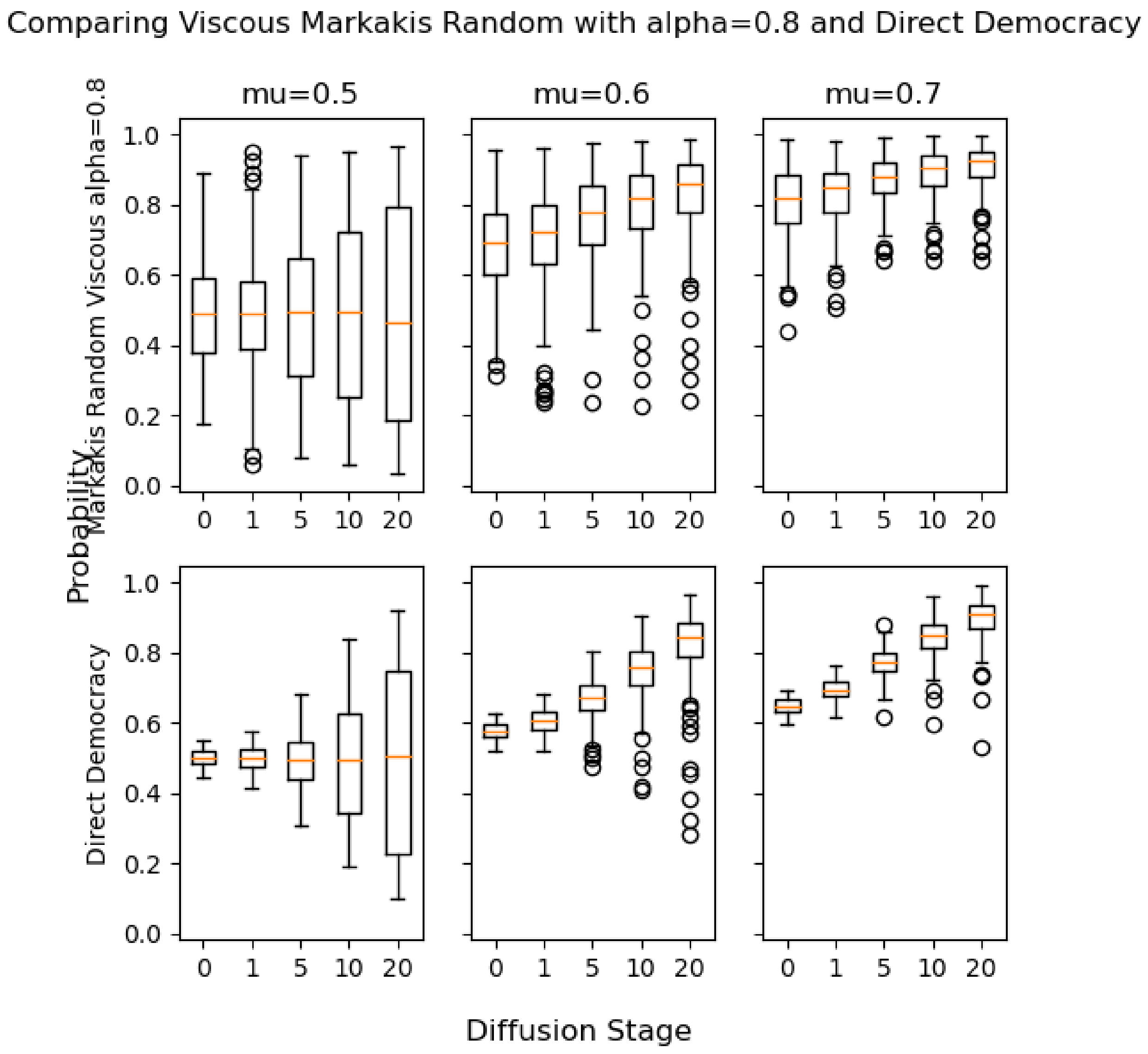

Comparing Viscous Markakis Minimal with alpha=0.8 and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Markakis Minimal Viscous alpha=0.8
Direct Democracy
Probability
Diffusion Stage
0.0
0.2
0.4
0.6
0.8
1.0
0
1
5
10
20

Figure 25:



Comparing Viscous Markakis Random with alpha=0.8 and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Markakis Random Viscous alpha=0.8
Direct Democracy
Probability
Diffusion Stage
0.0
0.2
0.4
0.6
0.8
1.0
0
1
5
10
20

Figure 26:

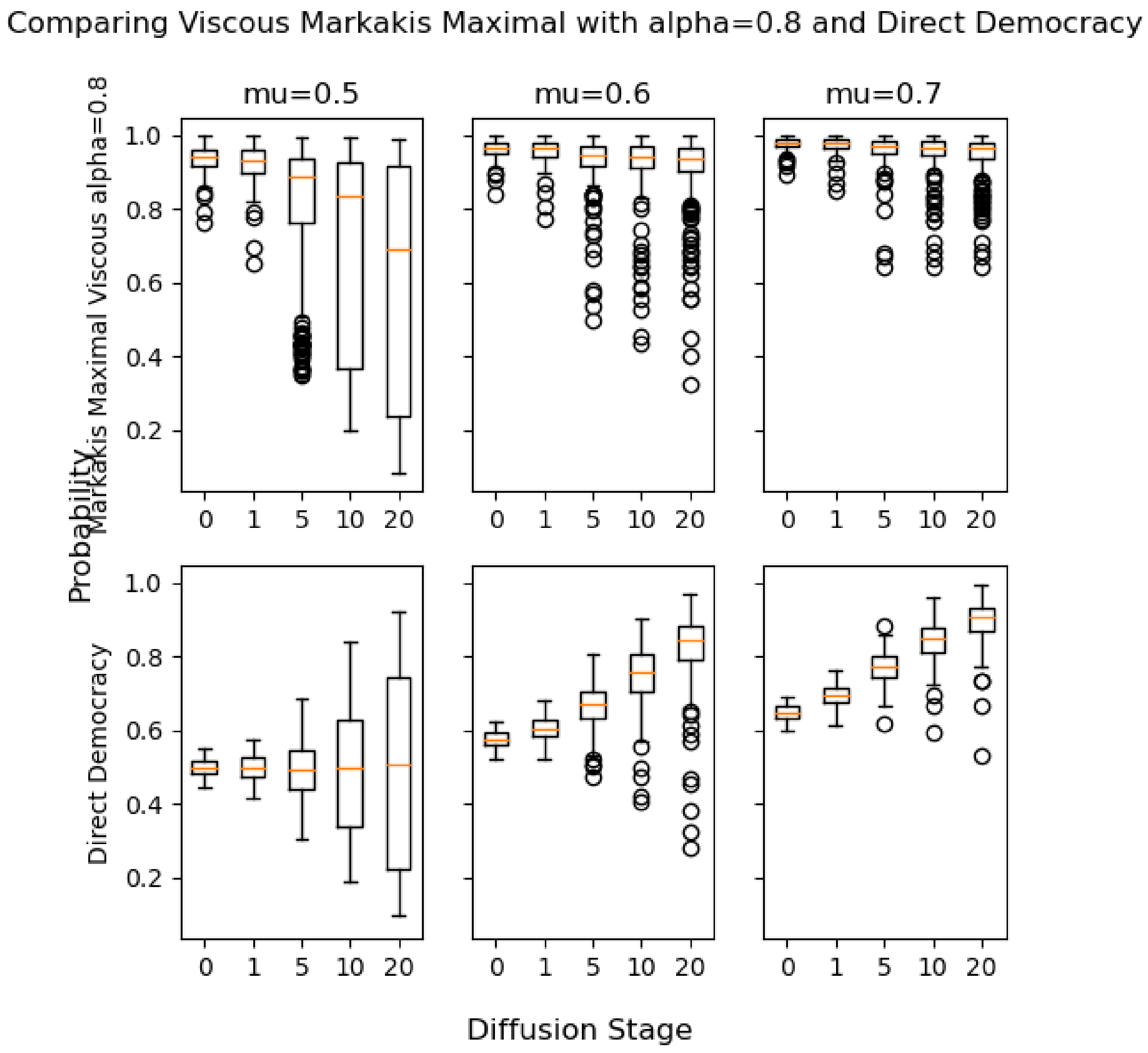

Comparing Viscous Markakis Maximal with alpha=0.8 and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Markakis Maximal Viscous alpha=0.8
Direct Democracy
Probability
1.0
0.8
0.6
0.4
0.2
0
1
5
10
20
Diffusion Stage

Figure 27:

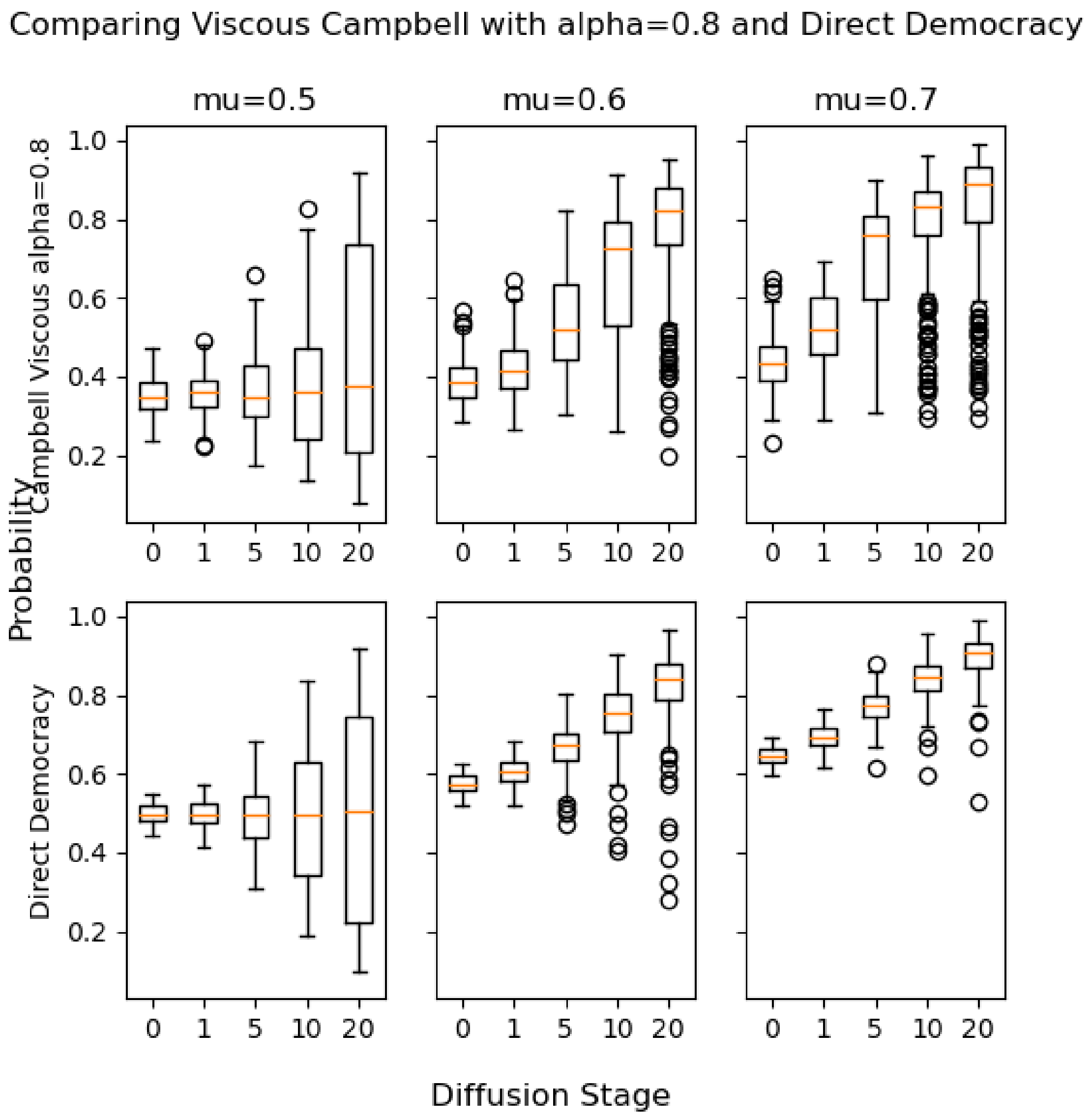

Comparing Viscous Campbell with alpha=0.8 and Direct Democracy
mu=0.5
mu=0.6
mu=0.7
Probability
Campbell Viscous alpha=0.8
Direct Democracy
1.0
0.8
0.6
0.4
0.2
0
1
5
10
20
Diffusion Stage

Figure 28:

## 9.3 C: Code

Only the code for the relevant functions and one version of the aggregate framework are shared here.

- KnowVar:

```
def KnowVar(n,mu,sigma,seed):           #The initial knowledge states of voters
lower =0                  #picking normally distributed values between 0 and 1
upper =1
Knowledge = sp.stats.truncnorm.rvs(
      (lower-mu)/sigma,(upper-mu)/sigma,mu,sigma,n,random_state=seed)
return Knowledge
```

- Adjacency:

```
def Adjacency(n,seed):                   #Adjacency Matrix to describe graph structure
    rng=np.random.default_rng(seed)  #For reproducability
    A=rng.integers(0,2,(n,n))        #Creating a random nxn matrix consisting of 0 and 1
    for i in range(0,n):             #Making it symmetric
        for j in range(i,n):
            A[i][j]=A[j][i]
        A[i][i]=1                    #Diagonal can't have a 0
    return A
```

- Confidence_bound

```
def Confidence_bound(Knowledge_array): #finding the confidence bounds
#print("Confidence Bound Start")
length=len(Knowledge_array)
bound=np.array([])
for i in range(0,length):
    bound=np.append(bound,0.5-abs(0.5-Knowledge_array[i]))
#print("Confidence Bound Done")
return bound
```

- Confidence_Matrix

```
def Confidence_Matrix(Adjacency,Knowledge,Bound):
#print("Confidence Matrix Start")
n=len(Knowledge)
A=np.zeros((n,n))
for i in range(0,n):
    Influence=np.array([])
    for j in range(0,n):
        if Adjacency[i][j]==1 and abs(Knowledge[i]-Knowledge[j])<=Bound[i]:
            Influence=np.append(Influence,j)
    Size=len(Influence)
    for j in range(0,n):
        if j in Influence:
            A[i][j]=1/Size
#print("Confidence Matrix Done")
return A
```

- direct

```
    #DIRECT DEMOCRACY
def Direct(Knowledge):                  #Takes knowledge state of voters
    EDir=np.sum(Knowledge)              #Expectation
    PDir=EDir/len(Knowledge)            #Probability
    return PDir
```

- Identifier

```
    #PATH IDENTIFIER FOR CYCLES
def Identifier(Size, nbar, Dnew): #Takes the transpose of the delegation graph Dnew.
    z=0
    i=nbar[z]
    j=0
    Parr=np.array([i],dtype=int)
    Length=len(Parr)
    while Dnew[i][i]==0 and j<=Size:  #First time round we do not
        if Length==0:
            for j in range(0,Size):
                if Dnew[j][i]!=0:
                    p=i
                    Parr=np.append(Parr,p)
                    Length=len(Parr)
                    if Length>=8:
                        break
        if Length>=8:
                        break
        else:
            while j <=Size:
                if Dnew[j][i]!=0:
                    p=j
                    i=p
                    j=0
                    Parr=np.append(Parr,p)
                    Dnew[p]=Dnew[p]+Dnew[Parr[z]]
                    z=len(Parr)-1
                    if Dnew[i][i]!=0:
                        j=Size+1
                    if  z!=0:
                        #Extra condition in case of a jellyfish cycle.
                        for l in range(0,z):
                            if Parr[z]==Parr[l]:
                                j=Size+1
                else:
                    j+=1
            if z!=0 and Parr[z-1]==Parr[0]:
                j=Size+1
    return Dnew, Parr
```

- IdentifierTwo

```
    #PATH IDENTIFIER FOR CYCLELESS DELEGATION GRAPHS
def IdentifierTwo(Size, nbar, Dnew):
    z=0
    i=nbar[z]
    j=0
    Parr=np.array([i],dtype=int)
    Length=len(Parr)
    while Dnew[i][i]==0 and j<=Size:
        if Length==0:
            for j in range(0,Size):
                if Dnew[j][i]!=0:
                    p=i
                    Parr=np.append(Parr,p)
                    Length=len(Parr)
                    if Length>=8:
                        break
        if Length>=8:
                        break
        else:
            while j <=Size:
                if Dnew[j][i]!=0:
                    p=j
                    i=p
                    j=0
                    Parr=np.append(Parr,p)
                    Dnew[p]=Dnew[p]+Dnew[Parr[z]]
                    z=len(Parr)-1
                    if Dnew[i][i]!=0:
                        j=Size+1
                    if  z!=0 and Parr[z]==Parr[0]:
                        j=Size+1
                else:
                    j+=1
            if z!=0 and Parr[z-1]==Parr[0]:
                j=Size+1
    return Dnew, Parr
```

- Cyclefinder

```
def Cyclefinder(D):
Dnew=np.transpose(D)
Size=len(D[0])
nbar=np.arange(Size,dtype=int)
Array=[]
while len(nbar)!=0:
    Dnew, Arr=Identifier(Size,nbar,Dnew)
    length=len(Arr)
    if len(Arr)>=2:
     #Extra condition in case of jellyfish cycles.
        for l in range(0,length-1):
            if Arr[l]==Arr[len(Arr)-1]:
                Array=[*Array, *Arr[l:]]
    i=0
    for i in range(0,len(Arr)):
        nbar=np.delete(nbar,np.argwhere(nbar==Arr[i]))
return Array
```

- Pathfinder

```
def Pathfinder(D,Epsilon): #Only if Cyclefinder gives an empty list.
Check=Cyclefinder(D)
if len(Check)!=0:       #Check if there are still cycles in the delegation matrix
    return print("The graph contains cycles")
else:
#Finding all paths to gurus and the corresponding weighted delegation matrix
    Dnew=np.transpose(D)
    Size=len(D[0])
    nbar=np.arange(Size,dtype=int)
    Array=[]
    while len(nbar)!=0:
        Dnew, Arr=IdentifierTwo(Size,nbar,Dnew)
        Array=Array+[Arr.tolist()]
        i=0
        for i in range(0,len(Arr)):
            nbar=np.delete(nbar,np.argwhere(nbar==Arr[i]))
for i in range(len(Array)-1,-1,-1):           #Deleting subpaths of longer paths.
    for j in range(len(Array)-1,-1,-1):
        if i!=j and set(Array[i])<=set(Array[j]):
            del Array[i:i+1]
return Array, Dnew
```

- Buster

```
def Buster(D,Epsilon,cycles,seed,t): #Inputs: Delegation matrix, Confidence matrix
,cycles, seed, time
Copycycle=copy.deepcopy(cycles)
rng=np.random.default_rng(seed)
Entries=cycles
Epsilon=Epsilon
Length=len(Epsilon[0])
corrected=np.array(())
k=0
z=0
i=0
m=0
numbers=[]
Numbers=[]
for i in Entries:
    for j in range(0, len(Epsilon[i])):
        if Epsilon[i][j]!=0:
            numbers+=[j]
    Numbers+=[numbers]
    numbers=[]
i=0
j=0
while i <len(Entries):
    p=Entries[i]
    if Entries[i] in corrected:
        i+=1
    else:
        if Entries[i]==p and m!=0:
            for l in range(z,i):
                k+=len(Numbers[l])
            if t>=20:
                D[Entries[i]]=np.zeros((Length))
                D[Entries[i]][Entries[i]]=1
            if k==i-z and t!=20:
                A=rng.integers(0,k)
                D[Entries[A+z]]=np.zeros((Length))
                D[Entries[A+z]][Entries[A+z]]=1
            else:
                for l in range(z,i):
                    A=rng.integers(0,len(Numbers[l]))
                    D[Entries[l]]=np.zeros((Length))
                    D[Entries[l]][Numbers[l][A]]=1
            z+=i
            corrected=np.append(corrected,Entries[i])
            i+=1
        else:
            i+=1
            m+=1
return D
```

- Campbell

```
def Campbell(D,knowledge,seed):
rng=np.random.default_rng(seed)
n=len(knowledge)
Approv=np.zeros((n,n))
for i in range(0,n):   #creating a matrix of confidence values for connected voters
    for j in range(0,n):
        if D[i][j] != 0:
            Approv[i][j]=knowledge[j]
for i in range(0,n): #Creating approval matrix of only approved connected voters.
    if knowledge[i]<=0.25:
        A=rng.integers(0,2)
        if A==0:  #case vote for antiexpert
            for j in range(0,n):
                if Approv[i][j]>0.25:
                    Approv[i][j]=0
            if max(Approv[i])==0:   #case if there are no approved delegates.
                Approv[i][i]=knowledge[i]
        else:     #case vote for nonexpert
            for j in range(0,n):
                if Approv[i][j]<=0.25 or Approv[i][j]>=0.75:
                    Approv[i][j]=0
            if max(Approv[i])==0:   #case if there are no approved delegates.
                Approv[i][i]=knowledge[i]
    if knowledge[i]>=0.75:
        A=rng.integers(0,2)
        if A==0:  #case vote for expert
            for j in range(0,n):
                if Approv[i][j]<0.75:
                    Approv[i][j]=0
            if max(Approv[i])==0:   #case if there are no approved delegates.
                Approv[i][i]=knowledge[i]
        else:     #case vote for nonexpert
            for j in range(0,n):
                if Approv[i][j]<=0.25 or Approv[i][j]>=0.75:
                    Approv[i][j]=0
            if max(Approv[i])==0:   #case if there are no approved delegates.
                Approv[i][i]=knowledge[i]
    else:
            A=rng.integers(0,3)
            if A==0:  #case vote for expert
                for j in range(0,n):
                    if Approv[i][j]<0.75:
                        Approv[i][j]=0
                if max(Approv[i])==0:   #case if there are no approved delegates.
                    Approv[i][i]=knowledge[i]
            if A==1:     #case vote for nonexpert
                for j in range(0,n):
                    if Approv[i][j]<=0.25 or Approv[i][j]>=0.75:
                        Approv[i][j]=0
                if max(Approv[i])==0:   #case if there are no approved delegates.
                    Approv[i][i]=knowledge[i]
            else:     #case vote for antiexpert
                for j in range(0,n):
                    if Approv[i][j]>0.25:
                        Approv[i][j]=0
                if max(Approv[i])==0:   #case if there are no approved delegates.
                    Approv[i][i]=knowledge[i]
    Epsilon=copy.deepcopy(Approv)
    for i in range(0,n):
        cols=np.array((),dtype=int)
        for j in range(0,n):
            if Approv[i][j]!=0:
                cols=np.append(cols,j)
        A=rng.integers(0,len(cols))
        for j in cols:
            if j!=cols[A]:
                Approv[i][j]=0
            else:
                Approv[i][j]=1
    return Epsilon, Approv
```

- confcheck

```
def confcheck(Paths,Epsilon,Dnew,A):
i=0
while i <len(Paths) and i<=20000:
    j=1
    while j <len(Paths[i]):
        k=0
        while k <len(Paths[i]):
            if Epsilon[Paths[i][j]][Paths[i][k]]==0 and A[Paths[i][j]][Paths[i][k]]!=0
            and j!=k and len(Paths[i])>1:
                placeholder=Paths[i][j::]
                Paths[i]=Paths[i][0:j]
                Paths[i+1:i+1]=[placeholder]
                Dnew[Paths[i][len(Paths[i])-1]][Paths[i+1][len(Paths[i+1])-1]]=0
                j=1
                k+=1
            else:
                k+=1
        j+=1
    i+=1
return Paths, Dnew
```

- counter

```
def counter(Dnew):
length=len(Dnew[0])
Array=np.zeros((length))
for i in range(0,length):
    for j in range(0,length):
        if Dnew[i][j]>1:
            Dnew[i][j]=1
for i in range(0,length):
    if Dnew[i][i]!=0:
        Array[i]=sum(Dnew[i])
return Array
```

- Markakisrndm

```
    #Markakis
def markakisrndm(A,xini,conf,seed): #Adjacancy matrix, initial knowledge
, confidence bound, random seed
    Length=len(xini)
    rng=np.random.default_rng(seed)
    Epsilon=np.zeros((Length,Length))
    for i in range(0,n):
        for j in range(0,n):
            if A[i][j]==1 and abs(xini[i]-xini[j])<=conf[i]:
                Epsilon[i][j]=xini[j]
    Approv=np.zeros((Length,Length))
    Numbro=np.array((),dtype=int)
    for i in range(0,Length):
        for j in range(0,Length):
            if Epsilon[i][j]!=0:
                Numbro=np.append(Numbro,j)
        rndm=rng.integers(0,len(Numbro),dtype=int)
        Approv[i][Numbro[rndm]]=1
        Numbro=np.array((),dtype=int)
    return Epsilon, Approv
```

- markakisdetmax

```
def markakisdetmax(A,xini,conf,seed): #Adjacancy matrix, initial knowledge
, confidence bound, random seed
Length=len(xini)
Epsilon=np.zeros((Length,Length))
for i in range(0,n):
    for j in range(0,n):
        if A[i][j]==1 and abs(xini[i]-xini[j])<=conf[i]:
            Epsilon[i][j]=xini[j]
    if max(Epsilon[i])==0:
        Epsilon[i][i]=xini[i]
Approv=np.zeros((Length,Length))
for i in range(0,Length):
    j=0
    k=0
    while j<Length:
        if j==0:
            k=j
            j+=1
        else:
            if Epsilon[i][j]>Epsilon[i][k]:
                k=j
            j+=1
    Approv[i][k]=1
return Epsilon, Approv
```

- markakisdetmin

```
def markakisdetmin(A,xini,conf,seed): #Adjacancy matrix, initial knowledge
, confidence bound, random seed
Length=len(xini)
Epsilon=np.zeros((Length,Length))
for i in range(0,n):
    for j in range(0,n):
        if A[i][j]==1 and abs(xini[i]-xini[j])<=conf[i]:
            Epsilon[i][j]=xini[j]
    if max(Epsilon[i])==0:
        Epsilon[i][i]=xini[i]
Approv=np.zeros((Length,Length))
for i in range(0,Length):
    j=0
    k=0
    while j<Length:
        #print("j=",j)
        if Epsilon[i][k]==0 and Epsilon[i][j]==0:
            j+=1
        if Epsilon[i][k]==0 and Epsilon[i][j]!=0:
            k=j
            j+=1
        else:
            if Epsilon[i][j]!=0 and Epsilon[i][j]<Epsilon[i][k]:
                k=j
            j+=1
    Approv[i][k]=1
return Epsilon, Approv
```

- viscous

```
    #Viscous Democracy
def viscous(Dnew,Paths,alpha):
    Length=len(Dnew[0])
    nbar=np.arange(0,Length)
    Array=np.zeros(Length).tolist()
    while len(nbar)>0:
        zeroth=nbar[0]
        j=0
        while j < len(Paths):
            plen=len(Paths[j])
            guru=Paths[j][len(Paths[j])-1]
            k=0
            while k < plen:
                if Paths[j][k]==zeroth:
                    Array[guru]+=alpha**(plen-k-1)
                    nbar=np.delete(nbar,0)
                    k=plen
                    j+=len(Paths)
                else:
                    k+=1
            j+=1
    Weigth=sum(Array)
    return Array, Weigth
```

- `Aggreg_xxxx`
- This code block summarizes all the aggregate functions for all delegation mechanisms. The only difference will be the delegation function chosen, and if it uses either the approval method or viscous democracy. In case of viscous democracy the lines:

  ```
          PathsY, DnewY=confcheck(Paths,EpsilonY,DnewY,Ab)
  ```

  and

  ```
          CountY=counter(DnewY)
          ProbabilityY=np.dot(CountY,xini)/XLength
  ```

  get replaced by:

  ```
          ArrayY, WeigthY=viscous(DnewY,Paths,alpha)
  ```

  and

  ```
          ProbabilityY=np.dot(ArrayY,xini)/WeigthY
  ```

  where Y represents the diffusion stage.

```
def Aggreg_xxxx(A,xini,seeding):
Ab=copy.deepcopy(A)
xini=copy.deepcopy(xini)
t=0
while t<20:
    if t==0:
        bound=Confidence_bound(xini)
        conf0=copy.deepcopy(bound)
        A=Confidence_Matrix(Ab,xini,bound)
        xnew=np.matmul(A,xini)
        x1ni=copy.deepcopy(xnew)
        t+=1
    if t==1:
        bound=Confidence_bound(xnew)
        conf1=copy.deepcopy(bound)
        A=Confidence_Matrix(Ab,xnew,bound)
        xnew=np.matmul(A,xini)
        x1ni=copy.deepcopy(xnew)
        t+=1
    if t==4:
        bound=Confidence_bound(xnew)
        conf5=copy.deepcopy(bound)
        A=Confidence_Matrix(Ab,xnew,bound)
        xnew=np.matmul(A,xnew)
        x5ni=copy.deepcopy(xnew)
        t+=1
    if t==9:
        bound=Confidence_bound(xnew)
        conf10=copy.deepcopy(bound)
        A=Confidence_Matrix(Ab,xnew,bound)
        xnew=np.matmul(A,xnew)
        x10ni=copy.deepcopy(xnew)
        t+=1
    if t==19:
        bound=Confidence_bound(xnew)
        conf20=copy.deepcopy(bound)
        A=Confidence_Matrix(Ab,xnew,bound)
        xnew=np.matmul(A,xnew)
        x20ni=copy.deepcopy(xnew)
        t+=1
    else:
        bound=Confidence_bound(xnew)
        A=Confidence_Matrix(Ab,xnew,bound)
        xnew=np.matmul(A,xnew)
        t+=1
Epsilon0,Approv0=xxxx(Ab,xini,conf0,seeding)
```

```
#inputs depend on delegation mechanism xxxx
Epsilon1,Approv1=xxx(Ab,x1ni,conf1,seeding)
Epsilon5,Approv5=xxxx(Ab,x5ni,conf5,seeding)
Epsilon10,Approv10=xxxx(Ab,x10ni,conf10,seeding)
Epsilon20,Approv20=xxxx(Ab,x20ni,conf20,seeding)
Cycles=Cyclefinder(Approv0)
t=0
while len(Cycles)!=0 or t<=20:
    Matrix=Buster(Approv0,Epsilon0,Cycles,seeding,t)
    Cycles=Cyclefinder(Matrix)
    if np.all(Matrix-Approv0==0):
        t+=1
Paths, Dnew0=Pathfinder(Approv0,Epsilon0)
Paths0, Dnew0=confcheck(Paths,Epsilon0,Dnew0,Ab)

Cycles=Cyclefinder(Approv1)
t=0
while len(Cycles)!=0 or t<=20:
    Matrix=Buster(Approv1,Epsilon1,Cycles,seeding,t)
    if np.all(Matrix-Approv1==0):
        t+=1
    Cycles=Cyclefinder(Matrix)
Paths, Dnew1=Pathfinder(Approv1,Epsilon1)
Paths1, Dnew1=confcheck(Paths,Epsilon1,Dnew1,Ab)

Cycles=Cyclefinder(Approv5)
t=0
while len(Cycles)!=0 or t<=20:
    Matrix=Buster(Approv5,Epsilon5,Cycles,seeding,t)
    if np.all(Matrix-Approv5==0):
        t+=1
    Cycles=Cyclefinder(Matrix)
Paths, Dnew5=Pathfinder(Approv5,Epsilon5)
Paths5, Dnew5=confcheck(Paths,Epsilon5,Dnew5,Ab)

Cycles=Cyclefinder(Approv10)
t=0
while len(Cycles)!=0 or t<=20:
    Matrix=Buster(Approv10,Epsilon10,Cycles,seeding,t)
    if np.all(Matrix-Approv10==0):
        t+=1
    Cycles=Cyclefinder(Matrix)
Paths, Dnew10=Pathfinder(Approv10,Epsilon10)
Paths10, Dnew10=confcheck(Paths,Epsilon10,Dnew10,Ab)

Cycles=Cyclefinder(Approv20)
```

```
t=0
while len(Cycles)!=0 or t<=20:
    Matrix=Buster(Approv20,Epsilon20,Cycles,seeding,t)
    if np.all(Matrix-Approv20==0):
        t+=1
    Cycles=Cyclefinder(Matrix)
Paths, Dnew20=Pathfinder(Approv20,Epsilon20)
Paths20, Dnew20=confcheck(Paths,Epsilon20,Dnew20,Ab)

XLength=len(xini)

Count0=counter(Dnew0)
Probability0=np.dot(Count0,xini)/XLength
Count1=counter(Dnew1)
Probability1=np.dot(Count1,x1ni)/XLength
Count5=counter(Dnew5)
Probability5=np.dot(Count5,x5ni)/XLength
Count10=counter(Dnew10)
Probability10=np.dot(Count10,x10ni)/XLength
Count20=counter(Dnew20)
Probability20=np.dot(Count20,x20ni)/XLength
PArr=np.array([Probability0,Probability1,Probability5
,Probability10,Probability20]).tolist()
return PArr
```