POLITECNICO DI TORINO

Master: ICT for Smart Societies

Master's thesis:

Neural Fairness Consensus Protocol

Supervisors:
Prof. Tiziano Bianchi
Dr. Enrico Zanardo

Candidate: Gian Pio Domiziani (S265977)

Abstract

Nowadays, the concept of Immutable Decentralized/Centralized Distributed Ledger Technology, otherwise called *Blockchain*, is rising in many fields of the society. It is easy to hear about it in fields as energy [48], automotive [47], economy, voting, law, insurance, cloud computing, supply chain management, etc. [34].

However, many problems are present. In particular, from a technological point of view, the problem of *scalability*-intended as the capability to maintain the network, composed by a very huge number of nodes, up, with a reasonable operational velocity- should be solved in order to be really suitable in production; as well as the question of maintaining a high level of *security* in terms of data exchanged and identity of members, is an important issue, and has taken central stage thanks to legislation like the European Union's General Data Protection Regulation (GDPR). Finally, it is not always guaranteed that the system is truly *decentralized*. This is known as the *Blockchain Trilemma.* It is a fundamental problem that must be addressed before a global adoption of blockchain ecosystems.

To face such a problem, a remarkable number of solutions have been proposed in recent literature. In particular, several types of consensus' protocols have been conceived, in order to achieve scalability and velocity. In particular the *Proof of Stake* consensus approach was the first to show that a balanced combination of all these three properties may be possible.

Within this context, the main goal of the thesis' job is to design a *Neural Fairness Consensus Protocol (NFCP)* to validate transactions while, at the same time, preserving the privacy of the identity members, as well as the data exchanged into the network and assuring a scalability for the entire system.

The NFCP is a blockchain-based distributed ledger secured using neural networks and machine learning algorithms, enabling a permission-less participation in the process of transition validation while concurrently providing strong assurance about the correct functioning of the entire network.

Contents

Concepts Introduction

Before to describe the protocol and its features, a brief introduction to the Blockchain technology is given, in order to underline fundamental concepts as well as terminologies, for making the rest of work more easy to follow. In particular, the main components of a blockchain system are described, and the *consensus procedure* is discussed in depth allowing to understand its crucial rule in a blockchain system.

1.1 Blockchain

The starting point of the blockchain technology is often indicated with the date of the published paper *"Bitcoin: A peer-to-peer Electronic Cash system* [28], by the pseudonym *Satoshi Nakamoto*[1]. However, similar technologies were already known. In particular, the idea of immutably chaining blocks of information with a cryptographic hash function appears in the 1979 dissertation of Ralph Merkle, in which Merkle explains the concept of *Merkle hash trees*, which are crucial tools for the blockchain, as explained soon.
Arguably, many of the blockchain's elements are present in the David Chaum's 1979 vault system. Chaum describes the design of a distributed computer system that can be established, maintained, and trusted by mutually suspicious groups [27].
However, the work done by the pseudonym *Satoshi Nakamoto (SN)*, was the first to define a digital-currency in which the *double-spending problem* is solved without the need of a trusted authority or central server. This was possible thanks to the mechanism of an ordered list of blocks composed of transactions which are cryptographically secured by the *Proof of Work (PoW)* consensus protocol or *Satoshi consensus* to prevent double-spending in a trustless environment.

The *double-spending problem* arises when two parts want to exchange some stake, using some **digital payments channel**, but there is no possibility to certificate that such stake was already been used. In Finance this is known as *accountability*.
In order to better understand the problem, it may be useful to catch up how transactions are defined in a digital payment.

Transactions in a digital Payment: in a digital payment each transaction should be created, propagated on the network, validated, and finally added to a ledger of transactions. A transaction implies a change of ownership of a particular stake. In figure 1.1 a chain of transactions is shown. Let assume Alice wants to send money to Bob. It means a quantity q of Alice's money must pass from the Alice's balance to the Bob's balance, so a change of money's ownership must happen. Ownership of money in digital payments are defined with the use of the *Digital Signature DS*. A DS is a cryptographic tool used for providing **authenticity**, **integrity**

[1] consequently the *Bitcoin* term is easily jointed to the term *Blockchain*.

as well as **non-repudiation**, in the context of asymmetric cryptography. When Alice wants to send money to Bob, she must compute a message defined as the hash of the Bob's public key and the hash of the transaction containing the quantity of money to be transferred. In order to authenticate the message Alice firms it with its private key, while Bob verifies the message authentication with the Alice's public key. At the end of the story, the ownership of a stake has passed from Alice to Bob.
Until now, there are no possibility to establish if such transaction has been already used many times. Indeed, Alice may be send the same authenticated message to many persons just using the same transaction hash but changing the Owner's public key. Here is where the role of a third trusted entity comes into play. If there is a third person, for which Alice and Bob trust, a such person may take into count the ownership passage and memorizing when such transaction has happen saving it on a ledger. This is, more or less, what a central server, like a central bank, does when Alice sends money to Bob. The Alice's bank sends money from the Alice's balance to the Bob's balance located at the Bob's Bank. Both balance banks are updated by the central banks, that acts as a *Trusted-Third-Party (TTP)*, assuring to avoid the *double-spending problem.*

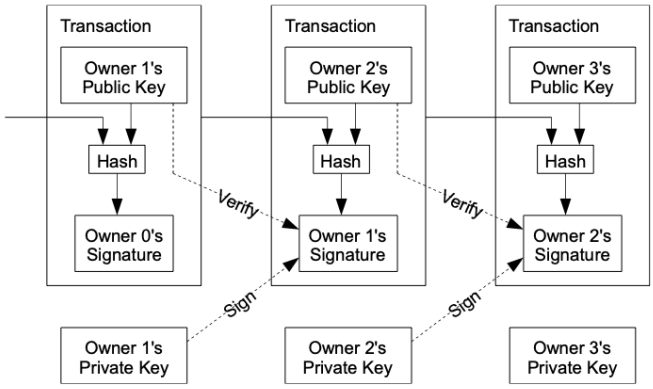


Figure 1.1: *How Digital Payment transactions are defined.*

Satoshi Nakamoto understood that in order to solve the *double-spending problem* each transaction should be seen by all the nodes and all of them must agree on a history describing when such transaction was created. In this way the TTP's role is assigned to nodes of a distributed *peer-to-peer* network. In this context it is the network that assures a validation of transactions, and the distributed ledger where all those transactions are memorized is the blockchain: a chain of hashed blocks containing several transactions that have been validated by the network when a *distributed consensus* is reached.

In figure 1.2 a simple Bitcoin chain, composed of three blocks, is shown.
It is possible to define a Bitcoin block as follows:

- A Tx_root: it is a Merkle root hash representing a bunch of transactions. The cryptographic hash function used assures that if some transaction will change even in a minimum part the Tx_root will change too.[2]
- Prev_hash: this is the hash of the previous block of the chain.
- A timestamp, indicating the time in which such block was created.

[2]this phenomena is known as the *avalanche effect* of cryptographic hash functions [42].

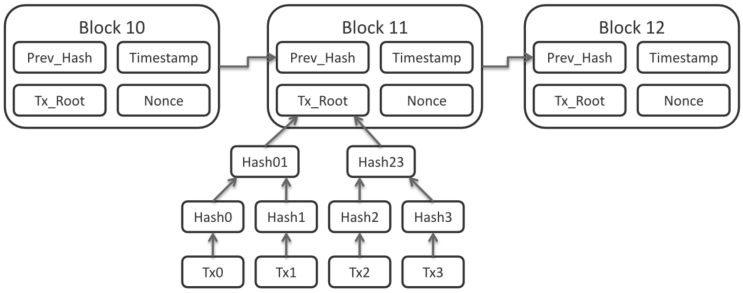


Figure 1.2: *Bitcoin blockchain architecture.*

- a Nonce is a value to be used just once, it is founded by the miner and its use is related to the PoW mechanism: the first peer that founds it becomes the leader and therefore gets the right to propose a block of transactions to be validated.

Once a new Block is created it will be chained with a previous block via its own Hash value. This mechanism allows to know if in the mean time some transaction has changed, because in this case the block hash must changed too, besides offering a way for establishing a *chronological* shared memory. A bunch of transactions are hashed into a unique *Merkle root*, forming the *Tx_ root* hash. The Merkle tree's properties allow to detect even the minimum changing in any transaction. This allows to avoid tampering actions.
Summarize, a blockchain is a sequence of *blocks*, which holds a complete list of *transactions* records which have been verified through a distributed consensus in an untrusted environment. Transactions between peers are executed using *asymmetric cryptographic primitives*. In particular *digital signature* is used to exchange transactions. Each peers has its own pair of private and public keys. The private key is used to *sign* a transaction, the public key is used to *verify* a transaction.
It can be concluded that:

- A blockchain allows to exchange transactions of goods and values, assuring *accountability* without the needed of a *Trusted-Third-Party (TTP)*;
- these transactions are grouped into blocks;
- blocks are chained together, creating a chronological order over the blocks and therefore about the transactions contained within them: each block contains a reference to a previously founded block (the hash block);
- the transactions that must be validated are decided by an elected node: the leader. It is the leader that proposes a block of transactions to be validated, and therefore decides the chronological order of transactions validation. In Bitcoin the leader is selected through the PoW: a cryptographic game, where the winner is the first peer that finds a *nonce* that combined with the *block header* results in a hash $\mathcal{H}_b$ with a given number of leading zero-bits, or *target*.

A blockchain system can be classified in according to the used rules for accessing to the system itself. When *anyone* can get access to the system without any restriction, the blockchain is said to be *permissionless*. Bitcoin and Ethereum [43] are examples of *permissionless Blockchain*. Here, any peer can join and leave the network, as reader and writer at any time. When the access to the network is allowed to only authorized limited set of readers and writers, the system is said to be *permissioned blockchain*. Here, a central entity decides and attributes the right to individual peers to participate to the blockchain operations. The most widely known

permissioned blockchain is *Hyperledger Fabric* [22].
Depending on the particular types, blockchain systems allows to reach some features, [55], [24]:

- *Decentralized:* in a centralized system, each state change must be validated through the central trusted agency acting as a third party. In a blokchain system there is no need of a third party for validating changes of the system, because the *distribute consensus algorithm* allows to maintain data consistency between distributed nodes, and therefore a consensus can be reached in an untrusted environment. A clear advantage of decentralized systems, is that the system is able to manage failure without compromising the entire normal process, because there is no a central entity that produces bottlenecks.
- *Public Verifiability:* it is intended as the possibility to anyone to verify the correctness of the state of the system. In a permissionless blockchain system, even if the state transition is confirmed by a restricted set of participants, anyone, however, can verify the state of the ledger, verifying if or not its state has changed in according to the protocol. In addition, all observers may have eventually the same view of the ledger. In a centralized system, the state transactions are not directly observable by anyone.
- *Integrity*: ensures that information is protected from unauthorized modifications. It is closely related to public verifiability, because if anyone can verify the states changes, it can be verified also the integrity of the information. In a centralized system integrity is provided only if the system is not compromised.

1.2 What problems does a blockchain solve?

Bitcoin and blockchain technologies have shown big advantage in *finance* applications, where the possibility to reach consensus in an untrusted environment without relying on an always on-line TTP gives a clear advantage with respect to a centralized scenario, allowing to decrease cost for managing transactions.
However, a blockchain allows to solve specific problems not *all* problems. It is important to have a big picture of what a blockchain can solve in order to better understand when its utilization can be profitable or not.
In according to the analysis of [24], we analyse some features that clarify when the use of a blockchain system can be profitable or not.

In figure 1.3 a flow chart for helping in a decision making process is shown. In general a blockchain system allows multiple untrusted parties to agree on the same history of events about states changes of a system, without utilizing a TTP.
The flow chart 1.3 explains that a blockchain system is useful when:

1. **Database**: there is something to store;
2. **Writers**: there are multiple writers that do not trust each other;
3. **TTP**: it would like to avoid to rely on a TTP.

when all the above items are present, the analyse continues to distinguish cases:

- if all writers are not known a *permissionless Blockchain* should be preferred. Otherwise, if all the known writers do not trust each other if *public verifiability* is not needed a *private permissioned blockchain* is suggest otherwise a *public permissioned one* is preferred.

When the above items are not present, a blockchain system could be useless, and its use should be avoid.

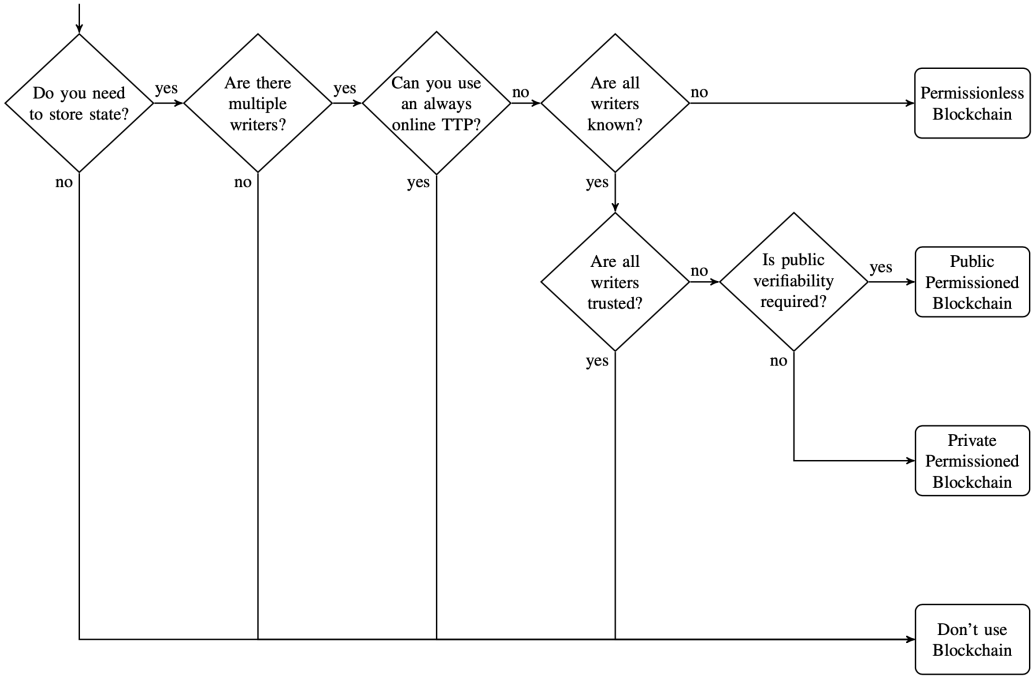


Figure 1.3: *Do you need a blockchain? credit [24].*

Doubtless, a blockchain acts as a distributed ledger, so if there are nothing to store its use is useless. If there is only one writer, intended as an entities with write access in a typical database system, a blockchain should be avoided, because in this case a regular database is enough, both for efficiency and ease of use. However, when multiple writers are present a blockchain could be preferred. But still, if all writers trust each other, a database with shared write access should be used, because *data integrity* would be guaranteed equally.
Finally, the question to ask is if it would be better to remove the intermediary, if there is one, and therefore choosing to use a blockchain? a blockchain utilization allows to reduce transactions cost with respect to the use of a TTP, in addition it implies automatic reconciliation, new regulation and robustness against central attacks, therefore, its use should be preferred.

1.3 Consensus protocols

An important blockchains feature is its capacity to offer *disintermediation*: multiple untrusted parties can *directly* and *transparently* interact with each other without the needed of a trusted intermediary. This makes blockchains well suitable for finance applications, as well as supply chain applications in general. However, its spreading is directly correlated to its scalability and performance. These properties are closely linked to the specific *consensus protocol* a blockchain system uses.
In this section, the *state of art* of consensus protocols are reported, as well as a deeper understanding on what a consensus protocol is and solves in a distributed system, in order to outline the main targets a consensus protocol should follow. A scheme for validating performances of consensus protocol is exposed, in according to [30], [31].

In general, a consensus protocol defines how processes[3] of a system can agree on requests that will produce a change on the current state of the system itself.

The problem of achieving a consensus in a distributed system is not new, indeed -even if in a different environment with respect to the blockchain- several approaches were investigated in the literature, producing interests and studies on *Fault-tolerant* distributed consensus protocol. L. Lamport et al. [32], defined the *Byzantine General Problem*, where possible anomalous system's behaviours that give conflict information to different parts of a system, are sketched abstractly in terms of a group of generals of the Byzantine army camped with their troops around and enemy city. In this context the main goal is to reach a distributed consensus, even if some faults (*Byzantine Faults*) are present. When this is reached, the system is defined as *fault-tolerant*, meaning it is able to tolerate a number of failures in a network continuing to work even in the presence of faults. Faults are solved via *replications*, where several copies of data are distributed across all processes in a distributed system. In particular, *State machine replication (SMR)* is a de facto technique that is used to provide replication services in order to achieve fault tolerance in a distributed system [35], [36].
Process failures can be separated into two types: *crash failure*, when nodes may fail at any time, stopping to process, emit or receive any messages, and *Byzantine failure*, when failed nodes may take arbitrary actions-including sending and receiving crucial messages to defeat property of the consensus protocol. In according to these two types of failures, Fault-tolerant consensus protocols can be classified into two categories:

- **Crash fault-tolerant (CFT):** covers only crash faults, excluding the possibility of faults created by malicious entities[4];
- **Byzantine fault-tolerant (BFT):** deals with types of arbitrary fault, even malicious.

Note that, BFT consensus protocols are able to manage even *crash failures*, so BFTs are a bigger set of consensus protocol than CFTs.

Two main properties a consensus protocol should follow are, [46], [40]:

- *safety/consistency:* if an honest node accepts (reject) a value, other honest nodes must be accepts (reject) the same value.
- *liveness:* requests from correct clients are eventually processed.

[3] a processes can be a machine in a distributed systems or a nodes in a network.
[4] a malicious entity is defined as a processes that does not follow the rules of a protocol.

To sum up, *Safety* property assures that nothing *bad* happens, *Liveness* property assures that something good *eventually* happens.
There exists four accepted requirements a BFT protocol must have [44], [46], which integrate and formalize, the objective of *safety/liveness*:

- *Termination:* every non-faulty process decides an output.
- *Agreement:* every non-fault process eventually decides the same output $\hat{y}$.
- *Validity:* if every process begins with the same input $\hat{x}$, then $\hat{y} = \hat{x}$.
- *Integrity:* every non-fault process' decision and the consensus value $\hat{y}$ must have been proposed by some non-faulty process.

These four requirements provide a general target for distributed consensus protocols. *Termination* and *validity* represent the system's liveness. *Agreement* and *integrity* represent the system's consistency/safety.[5]

The main contribute of these researches is that, for a distributed system, there exist a minimum available resources to be used, in order to reach a fault-tolerant consensus[6]. In particular, in a system where faults are not created by malicious entities (CFT consensus), the minimum number of processes required for consensus is lower bounded by $2f + 1$, where f is the number of faults. In systems where faults can be produced even by malicious entities (BFT consensus), the minimum number of resources is lower bounded by $3f + 1$, [37] [38].

Another crucial distinction in distributed systems is the way nodes of the system are coordinated. This distinction defines the *Network synchrony*. There are three levels of synchrony, name *synchronous*, *partially synchronous* and *asynchronous*, often assumed in the literature [44], [46] :

- In a *synchronous* network, operations of processes are coordinated under a same time reference. In each round all processes perform the same type of operations. This can be achieved by a centralized clock synchronization service and good network connectivity. In practical terms, a network is considered synchronous if message delivery is guaranteed within a fixed delay Δ.
- In a *asynchronous* network, operations of processes are hardly coordinated, due to the absence of a reference time between processes. There is no delay guarantee on a message except for its eventual delivery. And when some type of coordination between processes exists, it is based only on message delivery events.
- Finally, a *partial synchronous* network, operations of processes are loosely coordinated, in a way that message delivery is guaranteed but with uncertain amount of delays.

Fisher, Lynch and Paterson have proven that under asynchronous case, consensus cannot be guaranteed with even a single crash failure [49]. This is known as the FLP impossibility. However, this impossibility can be overpassed, using randomized decisions making and a relaxed termination property, as for example *Nakamoto consensus* does [29]. Moreover, blockchains systems are based on *Peer-to-Peer* networks, typically, formed over a *reliable* transport protocol as TCP, this allows to approximate the environment as at least *partially synchronous*, especially thanks to the TCP's *retransmission mechanism*.

[5] We will see how these four requirements can be generalized even for a blockchain consensus protocol.
[6] Resources can be processes or communication links.

1.3.1 Consensus' procedure steps

The goal of a consensus procedure is to ensure all participating nodes agree on a common transactions chain *history*. In according with the previous discussion, about distribution systems, It is possible to individuate four goals a consensus procedure should aim, in a blockchain context [31]:

- *Termination:* At every honest node, a new transaction is either discarded or accepted into the blockchain, within the content of a block.
- *Agreement:* Every new transaction and its holding block should be valuated in the same way by all honest nodes. An accepted block should be assigned the same sequence number by every honest node.
- *Validity*: If every node receives a same valid transactions/block, it should be accepted into the blockchain.
- *Integrity*: At every honest nodes, all accepted transactions should be consistent with each other (no double-spending). All accepted blocks should be correctly generated and hash-chained in chronological order.

Termination and *Validity* requirements represent the *liveness* property, for which *some good things eventually happen. Agreement* and *integrity* properties assure a chronological agreement is established between honest nodes, in order to have a coherent transactions chain, where the *double-spending* problem is avoided as well as any tampering action, thanks to the hash-chained mechanism. These are the minimum goals a blockchain consensus procedure should aim.

In according to the work of Yang Xiao *et al.* [31], we identify five procedures steps that formalize a blockchains consensus entire procedure. We explain them in the context of a *Committed based Proof of Stake*. In figure 1.4 the five steps forming a consensus procedure are shown. The green boxes contain steps that each committees board must do as an entity, while white boxes contain step the entire system as a collective must do.

We said a consensus procedure aims to assure an agreement, between distributed nodes, on the same transactions chain history. The entire procedure starts from a *Block proposal*, done by a *Leader*, in which an elected Leader node, makes a proposal of some transactions it collected. In order to be validated, the transactions must be analysed by all other committees board. An information propagation step is needed, where transactions inside the proposal block are propagated in the network, together the leader response, in order to reach the committees board. When the under review transactions have reached all committees board, the validation procedure can start. Here, elected nodes start to analyse the transactions, checking their coherence, in terms of *double-spending* and *ownership*. This step is done at the time t, by looking at the already validated blocks constituting the blockchain from t-1 to eventually t=0 (the block genesis). Then, in the *Block finalization* step, the committee has to find an agreement, looking at each committee's board responsum. Here, several rules can be used. In general the agreement is reached, when at least 2/3 of committees board express a responsus equal to the leader's responsus (BZT's rule). Besides the same version of the responsus between committee and the Leader, a new block should be added *if and only if* the block extends the knowing *longest-chain* at the nodes. This is the *longest-chain* rule used in *Bitcoin* blockchain. This rule avoid to incentive *forks* events, for which, at a certain point, different nodes may have different replica of the blockchain.
Finally, the *incentive mechanism* is needed in some configuration. This step is needed in order to incentive honest participation and creating new tokens. This is a mechanism that isn't present in distributed systems, where machines are supposed to be fixed, and the continuous participation

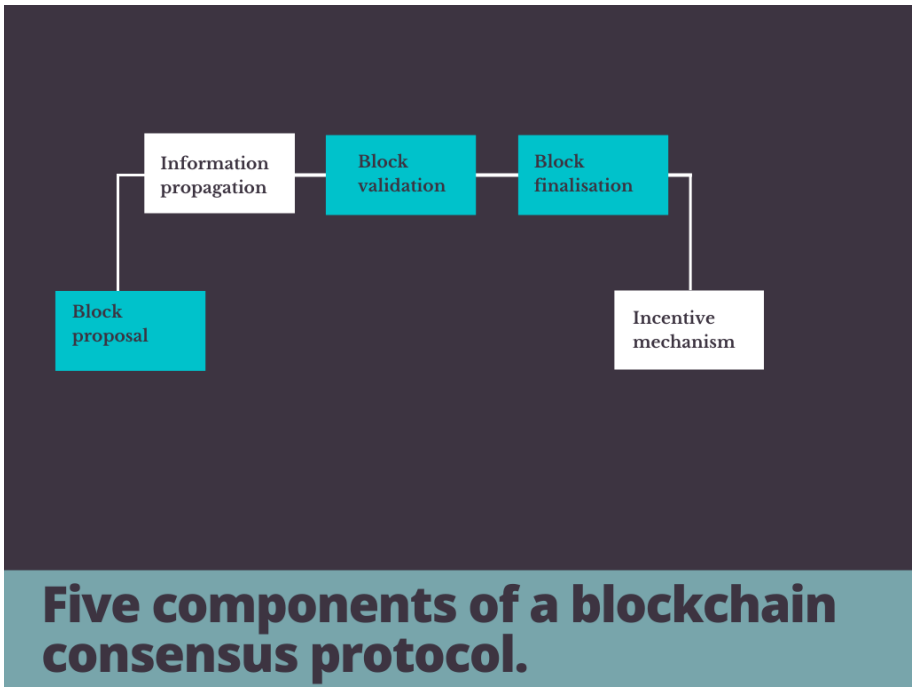


Figure 1.4: *Five components that define a a consensus procedure*

of honest parties is presumed. In a blockchain system, the network is realized by a *peer-to-peer* network formed by distributed nodes around the entire globe, each of them following individual scopes. An incentive mechanism encourages honest participation, by rewarding honest nodes, so that to sustain the system's reliable operation.
These steps are present in every *consensus procedure*, however consensus algorithms may differ in several methodologies. Before to discuss some of the most famous consensus algorithms, an important problem that can affect the validity of a blockchain is described.

1.3.2 Blockchain fork events

A fundamental assumption for assuring a distributed consensus is that there must be a *data consistency* between peers. Transactions validations are tracked in the blockchain, and each peer must keep its own *replica* of the blockchain locally. It is crucial that such replicas are *consistence* between them, because the validity of transactions is verified in according to them. Let's describe when *data consistency* can be lost between peers.
Each blockchain has its own *genesis block*: a root block of the directed tree forming the blockchain. The genesis block is an ancestor of all blocks for definition. The *blockchain* is defined as the *longest-chain* from any block to the genesis block. The distance between a block b and the genesis block is referred to as its *block height* h_b.
The genesis block g has $h_g = 0$, for definition. The block with maximum height, i.e. the block that is furthest away from the genesis, is referred to be the *blockchain head*, with height h_{head}. Let $\mathcal{B}_h$ be the set of blocks with height h. When $|\mathcal{B}_h| > 1$, with $h = h_{\text{head}}$, a *blockchain forks* is happen. During a blockchain fork nodes in the network do not agree of which block is the blockchain head, therefore as blocks are generated they are attached to different blockchains, forming an inconsistency between local replicas.

In figure 1.5 a blockchain fork is shown. In PoW blockchain based, if at a same time, two o more nodes find a valid nonce, they gets the right to propose a new block, at the same height.

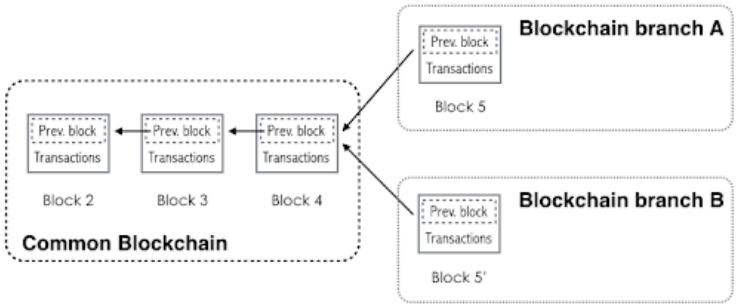


Figure 1.5: *Blockchain fork: the common share blockchain is splitting into two branches, therefore different nodes may have different replica of the blockchain.*

If this happens, different cluster of nodes may receive different blocks to be validated, and since all of the received blocks, have the same height, they both can be considered blocks head, and therefore they will be added to the existed blockchain, forming several blockchains branches. In this situation, nodes have a different local replica of the blockchain, therefore there is not more *data consistency*.
This situation can be very dangerous. Bitcoin never commits a transaction definitely. Every transaction can be in any time un-validated, if it is founded a longest chain started below the block that contains this transaction. Therefore, when a fork event happens, if a single entity could control a majority of the computational power of the network, and thus be able to find nonces in a faster way than other nodes, it could revert any transaction.
Decker and Wattenhofer [39] analysed how information is propagated in the Bitcoin network, considering *transactions* and *blocks* propagation time. They deduce that *propagation delay* is strictly related to the probability of blockchain forks events. In particular, their conclusion is that increasing the block size and decreasing inter block interval increases the probability of fork events. This implies that as Bitcoin utilization increases the propagation delay increases too, because there will be more transactions to be validated, consequently block size will be larger than before. In addition, if the inter-block time will be reduced, for assuring block creation, and therefore validation of transactions, in shorter times than before, the probability of fork events will increase, and therefore Bitcoin network will be more sensible to 51% attacks.

Fork events could happen in *asynchronous environment* as *peer-to-peer* networks are. Such events should be avoided, and several attempts have been done, for mitigating or eliminating them. Now, several consensus algorithms are explained, as well as their attempts to avoid *fork events*.

1.3.3 Proof of Work consensus

Proof-of-Work (PoW) was first proposed by Dwork and Naour in 1993 [30] as a technique for combating spam mail. Sending an email is a free action: it is possible to send multiple emails without spending any work. PoW requires to spend some work in order to get the right to take an action. *Nakamoto consensus* is derived from Hashcash [cite]. It replaces Hashcas's SHA-1 with two successive SHA-2 hashes, and introduce the possibility to defines a minimum value for considering the hash valid, introducing a target integer value t. Thus, the difficulty of the game is controlled by varying the target value: decreasing t increases the number of guesses (and thus work) required to generate a valid hash. The nodes that generate hashes are called *miners* and

the process is referred to as *mining*. The security of Nakamoto consensus relies on economically incentivising miners to validate and mine blocks, by rewarding them with new tokens.

1.3.4 Proof of X consensus

PoW is not sustainable as well as not really decentralized, allowing easy formation of centralization scenario. These limitations motivated a new class of consensus protocol based on Proof-of-X, where wasteful computations are replaced with more sustainable proof. *Proof-of-Stake* [25], [26], is one of the most famous new consensus approach, in the blockchain environment. In PoW, participants vote on new block weighted by their investment such as the amount of the currency held in the blockchain. A common theme in these systems is about select a random peer as Leader among the stakeholders, which then appends a new block to the blockchain. PoS results in two new attacks compared to Nakamoto consensus. The first is called *nothing-at-stake* attack, where miners try to extend any potential fork, for gaining control on the network. The second attack is called *grinding attack* where a miner re-creates a block multiple times until it is likely that the miner can create a second block shortly afterwards. This attack can be avoided by assuring that a miner is not able to influence the next leader election, by using a randomness source as seed of the election.

1.3.5 Hybrid consensus: single committee

In PoS there is a single consensus node suffers from poor performance as well as safety limitations as weak consistency and low fault-tolerance. These problems have generated new consensus procedure based on multiple consensus nodes forming a *committee*. Two of these committee consensus are Algorand [9], and Snow-White [52], where the committee members are selected for each epoch using randomness generated based on previous blocks. The NFCP uses a similar technique for selecting the committee members.

Neural Fairness Consensus Protocol

The *Neural Fairness Consensus Protocol (NFCP)* aims to offer *scalability* as well as *decentralization* in order to achieve a balanced trade-off in according to the trilemma problem [20].
The NFCP allows to achieve a distributed consensus, making use of just a selected sub-set of nodes of the entire distributed network, which will define a board committee that decide about the correctness of a block of transactions.
In particular, it is associated an importance to each node in according to its utility on the network. The grade of importance is represented by an utility factor.
Nodes having a high utility factor have a high probability to be part of the consensus procedure. Since, the utility factor will change over time, in according to the nodes behaviour in the network, the probability each node has to be part of the consensus procedure will change over time too.
In addition, in order to perform an ulterior selection of the board committee for the consensus procedure, another heuristic algorithm is designed. It solves the Conflict of Interest problem, for assuring that not always the same nodes participate to the board committee. This assures to reach *fairness* in the protocol.

The main contribute our protocol gives, is the capacity to assign a level of importance to each node in the network, and therefore managing the consensus phase, by using an *automatic learning* algorithm, identified as a *Reinforcement Learning (RL)* ones.
RL models are known for their capacity to learn by errors in a *unsupervised* way, driven by a *global scope*, that several *agents* have in common.
Actually, there are differences between *unsupervised* and RL models. *Unsupervised* models aims to find similarity between data, searching for hidden structures, without imposing any general rules. A RL model allows several agents to take *actions*, inside an *environment* by following a same *policy*, in order to maximize a reward signal.

A blockchain creates a *community*, where different entities, that are following specific rules, can exchange information, saving it and collaborate for finding distributed agreements. This is possible if and only if the network is composed by a number of honest entities grater than the possible number of dishonest entities.
Training a *learning* model to assign an utility to nodes in the network, in according to outcomes of events internal to the network itself, allows to create an intelligent environment able to not only detect *attacks* but even to predict *behaviours nodes* that are likely to generate *attacks* to the network.

In this chapter, a big picture of the entire system as well as some important definitions as *transactions, blocks*, are given, in order to understand how all the micro-services composing the protocol are dependent from each other.

2.1 Consensus Protocol Architecture

Before to describe the micro-services that compose the consensus protocol, let's have a big picture of the general architecture, in order to define the procedure needed to validate a new proposed block.

Let's define some features that allows to summary the NFCP:

- **Weighted nodes: utility factor**: To prevent Sybil attacks, a neural network model assigns a weight to each node. The protocol is designed to guarantee consensus as long as a weighted fraction (a constant greater than $2/3$) of the nodes are honest. The weight of the nodes is based on the score that the neural network calculate each time that a node is elected to be part of the committee and vote for a set of transactions as well as considering its behaviour inside the network layer.
- **Consensus committee** The system achieves scalability by choosing a committee that is a subset of nodes that are randomly selected by a pseudo random generator lottery, in order to reach a consensus. The first place of the lottery is designed for the leader. The leader is the node that starts the consensus phase. All other nodes observe and broadcast the protocol messages, which allows those that are part of the committee to agreed upon the proposed set of transactions.
- **Cryptographic Random Lottery** A Cryptographic Random Lottery is executed starting from the block hash, for finding random peers to form the consensus.
- **Reinforcement Learning model** The outcome of each committee members, about transactions, is taken by a dedicated RL model. It re-computes the positive scores of the peers, weighted the positive votes with their *voting power*. The nodes composing the system will have the possibility to learn a new more powerful model for predicting an utility to assign to nodes.

Using a RL model implies to take into account an essential phase of *training*, during which the model learns by its own errors. In a blockchain system errors may refer to accept some fraud transactions and this means an economic cost to be paid.
In order to manage the train phase, the entire blockchain will be initialised first of all in a *Test-net* version, where it is possible to train the agents without compromise anything. Indeed, in the *Test-net* version, anyone will be able to join the network and make transactions of *test-coin* as well as trying to attack the network itself. The entire system will be exposed to several attacks and scenarios in order to learn how manage them, and its understanding will be verified by re-proposing the same attacks and looking to the system's response. When the network will show the right balance, being able to predict and detect attacks, besides concluding in a correct way the normal consensus procedure, the network will be switched to a *main-net* phase, where the exchanged tokens will be *real tokens*.

In figure 2.1 a schematic picture of the logical flow the system must following, in the *Test-net* phase, is shown. Each time a new block is added, agents, in an independent way, collect parameters of other agents, in order to assign an utility to them. In this way, the utility factor is computed by other agents for other agents, avoiding cheating. Once utilities are assigned to nodes, the *lottery* can start, for building the committee. The lottery acts in a way to increase the probability of extracting winner tickets belonging to nodes with an utility as high as possible. The elected leader will propose a new block, which is analysed by other committee members, in order to find a distributed consensus. Finally, each agent will receive rewards from the RL model, in according to its taken actions. At the very beginning phase, a *genesis block* is created

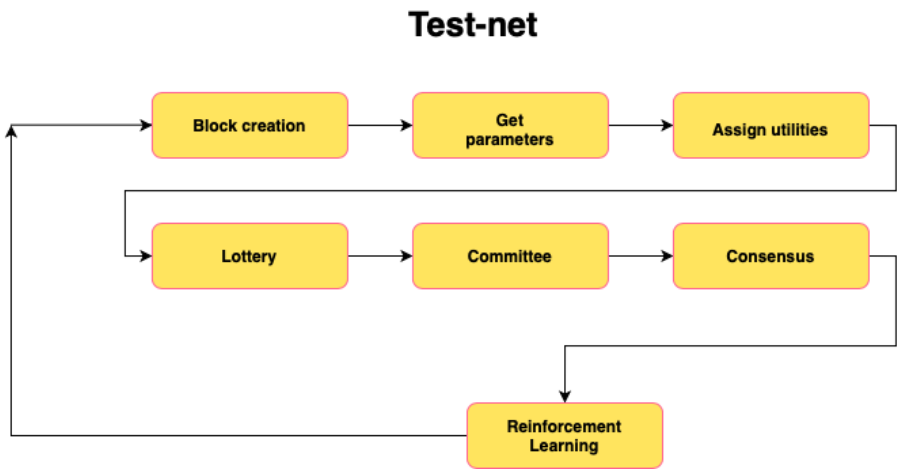


Figure 2.1: *Logical flow for the Test-net phase: the first time the block created is the genesis block, while utilities are initialised to random values.*

and all parameters are initialized in a random way. As the system keeps going to manage new block creations, such parameters will be adjusted in according to the RL model.
In the *Test-net* phase, the system is continually collecting *benchmarks* in order to evaluate how the system is learning. In this phase, a convergence is not assured, therefore there may be the possibility that the network does not reach the right parameters to be passed to the *main-net* phase. In this case, the system will be recomputed by changing the considered features as well as the possible actions an agent can take.

Let us focus on the assigned utility phase. This can be thought as a *learning algorithm* that nodes must running in order to assign an utility to only specific nodes. These selected nodes will have a probability to be part of the committee, and therefore participating to the lottery.
In figure 2.2 is reported a flow chart describing the logical flow to follow for individuating the sub set of nodes having the right to be part of the board committee for establishing the final consensus. It can be thought as a loop the system executes each time a new validated block hash is available.
The first step is about asking if the node's total balance is greater than a specific threshold, if yes it is possible to associate an utility factor to this node, otherwise the node has not the right to be assigned to an utility as well as to participate to the consensus procedure. This step assures a first selection in according to the idea of the *Proof of Stake* consensus procedure [25, 26], where nodes having too few stakes have less interest in the network than nodes having more stakes. Once all stakeholders have their utility, the second step is about being sure of deleting nodes having some conflict between each other nodes. This is a new feature that we are adding in order to achieve a good level of Fairness, assuring that nodes will have the same chance to be part of the block formations phase, avoiding always the same nodes participate to the consensus.
Finally, the subset of node having the right to be part of the consensus procedure is defined. Once this sub-set of nodes (stakeholder) is individuated, the *committee election* can start. This step requires to elect a Leader together others elected members. The Leader will propose a Block to be validated, the other elected members have to generate a responsus about the transactions contained in the Block proposed by the Leader. At the end of the procedure a new block will be added together its validated block hash, so the loop can start again.
From now onwards, we refer to nodes having the right to be associated with an utility factor as

stakeholders, because nodes having an utility factor have the right to be part of the consensus algorithm.

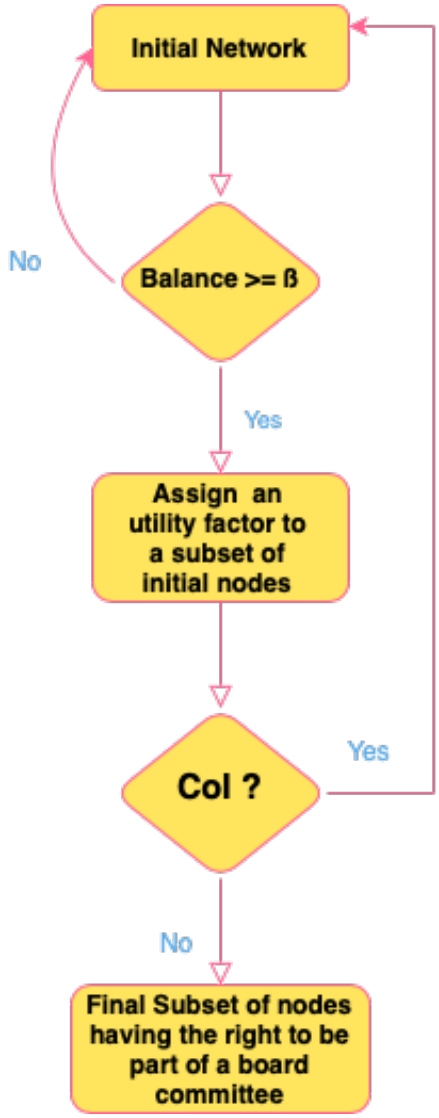


Figure 2.2: *Logical flow of the procedure executed for establishing which nodes have the right to be part of the board committee.*

2.2 Blocks and Transactions definition

Now, first of all, important elements as *blocks and transactions format* must be defined.

The way blocks and transactions are defined in our protocol, makes it different from others. In our idea, the blockchain starts with a predefined number of tokens, belonging to the *genesis address*. Tokens are defined as points of a *Elliptic Cryptographic Curve (ECC)*, this is needed for working with the *Cryptographic Lottery*. A token T_s is defined by a pair $T_s := (x_s, y_s)$, where x_s and y_s are the coordinates of a specific Elliptic curve's point.
Each time, a transaction between peers happens, a specific subsets of tokens must change

ownership, moving from the sender to the receiver. This is done by updating the *linked-list* of these specific tokens.

A transaction Tr_k of the token T_n, from A_s to A_r, is defined as:

- from: A_s; to: A_r; token: T_n; ownership: A_s's DS

where, *from* indicates the starting address, *to* the received address, *token* the specific token (point of the ECC) of this transaction, and *ownership* is a digital signature of the starting address, indicating the *authenticity*, as well as the *non-repudiation* of the transaction.[1] Each time a transaction has reached a node, the node has to check its format consistence as well as the ownership proof, and if and only if these requirements are verified it can flooding the transaction to its neighbours. This step allows to avoid flooding the network with *malicious or incorrect* information.

As already said, each token has its own *linked-list* containing all confirmed ownership (address) changes. This can be thought as the *Token's history*. For a token T_s its own *linked-list* is defined as shown in figure 2.3, where a possible token's history is reported:

Figure 2.3: *T_s's linked list: the token T_s has moved from the address A_j to the address A_k. The final owner is A_k because it is the tail of the linked list.*

In figure 2.3, the T_s linked list is shown.
The T_s history starts, as the history of any tokens, from the *genesis address* indicating with the symbol: (-). The address A_j was the first address to get the token T_s, which was finally transferred to the address A_k. The address A_k is the last owner of the token T_s, because it is the tail block of the linked list.

A linked list of each token can be thought as the entire history of ownership's changes of such token. This approach allows to solve the *double-spending* problem, when transactions are verified by the committee members.

Each time a transaction must be verified in terms of *double-spending*, a check that only the committee members have to do,[2] the verifier has to:

1. find the block containing the token's tail linked list of the transaction;
2. check if the tail box of the linked list has the sender address as node.
3. discarded or admitted the transaction, in according to the previous check.

From the previous items, we can formalise the Blocks content:

- The *genesis* block is a particular block, which contains all the linked list of all possible tokens, where the unique owner is the *genesis address (-)*. This defines the maximum size of a block.

[1] this helps to avoid that any address can send transactions on behalf of others.

[2] this is a different step of the simple action of checking the correctness of transaction format. Indeed, in this step the *double-spending check* is performed.

- A generic block contains all the last box of the linked list of tokens (the tail box) that were present in the last validated transactions as well as the Merkle root of the transactions. In other words, each block contains only the linked list of the tokens for which a change of ownership have been validated in the last consensus procedure. In particular a new block contains:
  - The previous block hash;
  - The current block hash;
  - The timestamp of the block creation;
  - The Merkle Tree of the validated transactions;
  - The tail of the linked list of the considered tokens.

2.2.1 Consensus logical steps

The consensus procedure starts each time a new block hash is present. In this phase the network works in a distributed way to elect a committee as much fair as possible. In a nutshell, the logical steps to follow can be summarized as follows:

1. New block hash? start committee formation.
2. Leader Block proposal.
3. Leader forwards transaction as well as its own responsus.
4. Committees board verification.
5. Leader receives committee's responsus, if there is agreement, firm block send it to all committee.
6. committee receives Leader firmed block, checks it and adds it to the blockchain

The entire stakeholders consensus procedure is executed with an algorithm, whose functioning as *pseudo-code* can be summarized as shown in 1:
The main loop catches up for a new block hash. When it is available, the *CommitteElect()* function is started. It includes, the utility assignments, the *CoI procedure* and finally the *Cryptography Lottery*. All this sub-functions are needed for electing a committee. When the committee is established a leader is individuated together all others committee members. The leader will propose a new block, attending the responsus of other members. The new block proposal is attached to the blockchain if and only if it is validated by at least a portion of 2/3 committee members, and it extends the longest known chain.

Now, a more lower level description is given, where the several micro-services that compose the entire system are described.

Algorithm 1: NFCP procedure algorithm

```
1  Join the network by connecting to know peers;
2  while True do
3  |  Main loop;
4  |  if new block hash then
5  |  |  CommitteeElect();
6  |  |  if Leader then
7  |  |  |  block proposal;
8  |  |  |  attend committee responsus;
9  |  |  |  if positive then
10 |  |  |  |  add block;
11 |  |  |  |  broadcast confirmation;
12 |  |  |  end
13 |  |  end
14 |  |  if received bloc & validated & extendes the longest chain then
15 |  |  |  send positive responsus;
16 |  |  |  attend Leader confirmation;
17 |  |  |  if positive then
18 |  |  |  |  add block;
19 |  |  |  end
20 |  |  end
21 |  end
22 end
```

Services composing the Consensus

In this chapter the micro-services that compose the consensus procedure are described. First of all the utility factor is defined, as well as all the requirements needed for obtaining it. There are two models to be described, a baseline model and a reinforcement learning model. Then, the *Conflict of Interest (CoI)* is discussed with the relative heuristic algorithm solution. Finally, the *Cryptographic Lottery* used for individuating the committee is discussed.

3.1 Utility Factor

The *Utility Factor* is a value that quantifies how much a node is valuable for the entire network and so it establishes how much a user has the right to be part of the committee. A committee member will take part in the consensus procedure, and so it will participate at the new block creation. This is a crucial task that must be done by nodes as honest as possible.

Minimum required balance to be part of the Consensus Procedure

A mandatory property needed to be part of the committee, is a minimum balance $\mathcal{B} \geq \rho$, where ρ is a token quantity to be defined. All nodes having a balance of at least $\mathcal{B}$, have a probability to be part of the Consensus procedure because *if and only if* to each of them is assigned an utility factor. In addition, in order to be considered, the minimum required balance, must be available for a certain time. A minimum balance available from today has an negligible importance with respect to a minimum balance available from 30 days.

3.1.1 Utility Factor Definition

The procedure to assign an utility factor, is related to assign a *value of importance* to a node being part of a network. The considered network is the network formed by the direct graph of transactions, formed by the set $\mathcal{V}$ of vertices represented nodes in the network that exchange transactions, and the set of $\mathcal{E}$ edges; $(i, j) \in \mathcal{E}$ iff at least a transaction from the node A_i to the node A_j is happened.
Transactions information are available to everyone, and the dynamic of validated transactions is controlled by the consensus protocol. This means that, in order to be considered into the transaction graph a transaction must be validated by the network trough a distributed consensus. Therefore, taking the graph transaction as the reference graph to extract the utility of each node assures:

1. everyone can verify if the utility associated to a node is valid, computing by itself the value making use of public open data as transactions.

2. information contained in the transaction graph are not malleable as long as the consensus procedure is secure.
3. the transaction graph evolve in time assuring to take into account always new information related to the behaviour of a node.

There are remarkable algorithms for assigning a value of importance to a node in a graph. A such problem is related to compute a *centrality degree* to a node for understanding its centrality in the network. In the context of the Web, one of the most famous is the *PageRank* algorithm [21], where the rank of a web page is obtained as the output of an iterative linear process applied on a Markov chain transition probability matrix.
The PoI is inspired by the NEM project [8] , in particular, the following *outlink matrix*, is part of the PoI protocol of NEM.

The outlink matrix

Suppose the PoI calculation is done at height h. The protocol considers, for each node with a balance $b \geq \mathcal{B}$, all its transactions with the following properties:

- Transferred an amount of at least ϕ tokens
- Happened within the last 43k blocks (approximately 30 days)

For each such transaction T_k that transferred an amount θ from the account A_i to the account A_j and happened at height h_{ijk}, a weight is associated to this transaction as follows:

$$w_{ijk} = \theta \cdot \exp[\log(0.9)[\frac{h - h_{ijk}}{1440}]]$$

where [x] denotes the floor function. The weight is obtained as the output of an exponential function depending on the time, so that transaction happened today has an higher associated weight than transaction happened the last 30 days.
Summing up all transaction

$$\hat{w}_{ij} = \sum_k w_{ijk}$$

and, setting

$$\hat{o}_{ij} = \begin{cases} \hat{w}_{ij} - \hat{w}_{ji} & \text{if } \hat{w}_{ij} - \hat{w}_{ji} > 0, \\ 0 & \text{otherwise .} \end{cases}$$

finally, the outlink matrix $\mathbf{O}$ is obtained as follows:

$$o_{ij} = \begin{cases} \frac{\hat{o}_{ij}}{\sum_i o_{ij}} & \text{if } \sum_i o_{ij} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

The outlink matrix element o_{ij} gives the **weighted net flow** of tokens from A_i to A_j during the (approximately) last 30 days. Summary, the outlink matrix takes into account only positive net transactions flow between nodes.

Degree and Betweenness Centrality

To catch up how much a node is important for the network, other two quantities are used: the **degree centrality** and the **betweenness centrality**.
The degree centrality is defined as the total number of edges a node has. Since, the transactions graph is a directed one, it is possible to define a in and out degree for a node. From the 3.1

matrix is possible to extract the number of degree of a node, counting the number of cells with a value different from zero.
The betweenness centrality is a measure to quantify how many times a node is present in the short paths of a network. A node having a high betweenness centrality may act as a bridge between several clusters in the network.
For a node i, its betweenness centrality $\mathcal{BC}$ is defined as:

$$\mathcal{BC}(i) = \sum_{s \neq i \neq t \in \mathcal{V}} \frac{\sigma_{st}(i)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(i)$ is the number of those paths that pass through i.

Finally, for a node i, whose has the right, its utility value is defines as:

$$\mathcal{U}_i = \rho_i \cdot \alpha + \sum_j o_{ij} + \mathcal{D}(i) + \mathcal{BC}(i) \tag{3.2}$$

where ρ_i is its balance, α a number to be defined, which takes into account how much the total balance should be considered important for the utility, $\sum_j o_{ij}$ is the total weight net flow, $\mathcal{D}(i)$ and $\mathcal{BC}(i)$, its degree and betweenness centrality respectively.

3.1.2 Utility formalization

Besides taking into account stakeholders behaviour inside of the transaction graph, the protocol aims to classify stakeholders behaviour in according to their actions in the *network layer*. A final score, to be added to the utility as defined until now, is given in according to specific features related to the network layer.
The main objective is to individuate stakeholders with a high activity in the network and with high *round-trips-time* as well as high *throughputs*. Imposing a higher utility factor to stakeholders showing a higher reliability in the context of the network layer, allows to get near to the assumption of *synchronous* network, where it is possible to define an upper bound Δ to message delays. This assumption allows to operate in a *real-time* scenario, besides offers several theoretical advantages for the consensus. Moreover, imposing the *committee* to be established by stakeholders which appear *available* as much as possible, allows to well approximate the assumption of having a prefix number of stakeholders on-line in the committee, when the consensus must be reached.

In the table 3.1 the considered features are shown. All of them are collected for a specific period of time, called *epoch*. For some of them history is collected, while other are refreshed at each *epoch*.

The features described above form the definition of the utility, meaning that a particular configuration of such features individuates the stakeholder for which the best utility must be assigned.

Now, two algorithms for computing the utility are described. In particular, the first one is thought as a *baseline* scenario, for investigating the problem without using a computational expensive approach, while the second one is a more powerful *learning* algorithm, that can reach better results, however requiring more computational resources.

Features	Values
Total number of TCP connection	integer
sum of all uploaded bytes	float
sum of all downloaded bytes	float
sum of the number of retransmitted bytes from peer	float
average round trip time between peers	float
average time for processing and returning the first segment with payload since the first flow	float
ping	float

Table 3.1: *Stakeholders features used to estimate the utility.*

3.1.3 Baseline Utility Algorithm

The best and the worst stakeholders features configuration that individuate respectively, the maximum and the minimum utility are selected by the system. A *Self-Organizing-Map (SOM)* is trained for individuating the *Best Unit Match (BUM)* neurons on the map, of these two selected features, which we will call *targets utility stakeholders (TUSs)*. Subsequently, in the *mapping phase*, all stakeholders samples are mapped to the trained grid, where each of them is assigned to a BUM neuron. An euclidean distance between the weights associated to the BUM's TUSs ($w_{\text{target}_1}, w_{\text{target}_2}$) and the BUM's weight w assigned to a stakeholder is computed, and according to these two distances an utility is assigned to the stakeholders, as follows:

$$U = \begin{cases} 1 + \exp(-d_2) & \text{if } d_1 > d_2 \\ 1 - \exp(-d_1) & \text{if } d_2 > d_1 \end{cases} \tag{3.3}$$

The pseudocode of the baseline utility algorithm is reported in the algorithm 2:

Algorithm 2: Baseline Utility algorithm pseudo-code

input : Two targets nodes & a subset of samples nodes.
output : a list of utility values to be assigned to the sample nodes

```
1  Train the SOM with the two target samples nodes;
2  get targets weights ;
3  foreach node ∈ 𝒩 do
4  |  predict the BMU;
5  |  get weights w associated to the predicted BMU;
6  |  compute distance d_1 between w and w_target_1;
7  |  compute distance d_2 between w and w_target_2;
8  |  if d_1 > d_2 then
9  |  |  u = 1 + exp(−d_2);
10 |  else
11 |  |  u = 1 − exp(−d_1);
12 |  end
13 end
```

In figure 3.1 the utility function's 3.3 shape is shown. The utility is a value in the range: $u \in [0, 2] \subset \mathbb{R}$. If a stakeholder is associated to the worst target neuron, its utility is in the range $[0, 1]$, otherwise, if it is associated to the best target neuron, its utility is in the range $[1, 2]$, in according with the distance between its weights and the target neuron's weights.

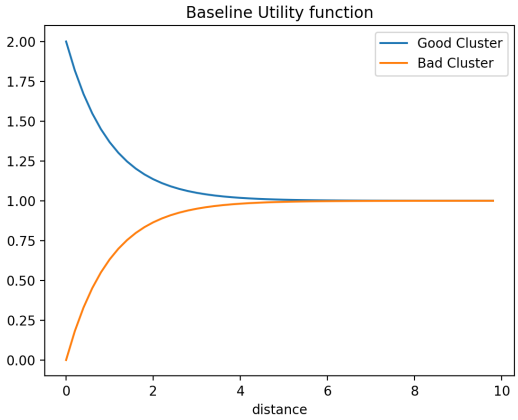


Figure 3.1: *Utility function's shape.*

3.2 Reinforcement Learning Utility algorithm

In this section the *Deep-Q-Network* model used for learning nodes to assign an utility to other nodes, is described. Before to explain the details, a brief introduction about the *Reinforcement Learning field* is given. In particular the Q-learning method is described, and how it is possible to use the well established *supervised Deep Neural Network* theory for solving a Reinforcement Learning problem.

3.2.1 Markov Decision Process (MDP)

Reinforcement Learning is strictly related to MDP. MDP is a discrete-time stochastic control process. It provides a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker. The main elements of MDP are:

- Environment
- Agent: a decision-maker interacting with the environment performing subsequent actions.
- States: Representations of the environment under certain conditions.
- Action: Performed by the agent with respect to the state.
- Reward: Consequence of the action given to the agent.

Given a finite set of states $\mathbf{S}$, a finite set of actions $\mathbf{A}$, and a finite set of rewards $\mathbf{R}$, at each time step $t = 0, 1, 2, ...$ the agent receives some representation of the environment's state $s_t \in \mathbf{S}$. Based on this state, the agent selects an action $a_t \in \mathbf{A}$ resulting in the state-action pair (s_t, a_t). Time is then incremented to the next time step $t+1$, and the environment transits to a new state s_{t+1}. At this time, the agent receives a numerical reward $r_{t+1} \in \mathbf{R}$ for the action a_t performed in the state s_t. The reward assignment is represented by a function $f(s_t, a_t) = r_{t+1}$.

3.2.2 Discount Return

The importance of the reward relies on the fact that the goal of the agent is to choose the action that maximizes the cumulative rewards. These rewards can be assessed to through the

concept of expected return defined as:

$$G_t = \sum_{j=1}^{T} r_{t+j}$$

where T is the final time step. In this way, the expected return is the sum of the future rewards. However, it may happens that the considered environment cannot evolve in a finite number of time steps, making $T = \infty$. Because of this, the concept of the expected return can be translated to the discounted return one. This means that the agent will try to perform an action by maximizing the cumulative rewards, but focusing more on the immediate reward over future rewards. The discounted return can be defined as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}$$

For every time point, the return is calculated as as um of subsequent rewards, but more distance rewards are multiplied by the discount factor raised to the power of the number of steps we are away from the starting point at t.
To know how likely it is for an agent to take any given action from any given state, the *policies* are used.
Formally, a policy π is defined as the probability distribution over actions for every possible state:

$$\pi(a|s) = Pr\{a_t = a|s_t = s\}$$

Since the main objective of the agent in RL is to gather as much return as possible, and different policies can give the agent different amounts of return, the main goal for a RL algorithm is to find the best policy in terms of returned rewards.

3.2.3 Q-Value

By recalling that the rewards an agent expects to receive are dependent on what actions the agent takes in given states, it is possible to define the Q-value as a measure of a state-action pair goodness in terms of expected/discounted returns. In this way, the state-action Q-value for policy π, referred as Q_π, tells how good is for the agent to take any given action from a given state while following policy π. And so, $Q_\pi(s, a)$ is the expected return from starting from the state s, taking the action a under the policy π:

$$Q_\pi(s, a) = E_\pi[G_t|s_t = s, a_t = s] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_t = s, a_t = a \right]$$

Note that, it is possible to define the Q-value in a recursive way, as follows:

$$Q(s, a) = r(s, a) + \gamma \max_{a' \in A} Q(s', a')$$

In the preceding formula, it is assumed that some reward is given to the agent immediately after executing a particular action a. But if reward is provided for reaching some state, s', via action a', the formula imposes that the Q-value for the action a given the state s, given the new state s', will take the optimal action a' that gives the maximum Q-value. This formula is used in *Deep-Q-Networks* models, where the deep network learns the optimal policy the agent has to follow, as described soon. The Bellman Optimality Equation gets the optimal Q-value using the recursive formula described above.

3.2.4 The Bellman Optimality Equation

The goal of reinforcement learning algorithms is to find a policy that will yield a lot of rewards for the agent if the agent indeed follows that policy. Specifically, reinforcement learning algorithms seek to find a policy that will yield more return to the agent than all other policies. For this reason, in terms of return, a policy π' is considered better with respect to another π if

$$\pi' \geq \pi \Leftrightarrow Q'_{\pi}(s,a) \geq Q_{\pi} \quad \forall s \in \mathbf{S} \quad \forall a \in \mathbf{A}$$

Therefore, the optimal policy has an optimal Q-value Q^* defined as:

$$Q^*(s,a) = \max_{\pi} Q_{\pi}(s,a)$$

If Q^* is optimal, then the Bellman equation is satisfied:

$$Q^*(s_t,a_t) = E\left[r_{t+1} + \gamma \max_{\pi} Q_{\pi}(s_{t+1},a_{t+1})\right]$$

This means that the optimal Q-value referred to an (s,a) pair at time t is the expected return r_{t+1} which can be obtained by choosing the action a from the state s plus the maximum expected discounted return that can be achieved from any possible next state-action pairs. Once Q^* is found, then it is possible to determine the optimal policy because, with Q^*, for any state s, a reinforcement learning algorithm can find the action that maximizes $Q^*(s,a)$.

3.2.5 Q-Learning

Q-learning is a technique that can solve for the optimal policy in an MDP. The objective of Q-learning is to find a policy that is optimal in the sense that the expected value of the total reward over all successive steps is the maximum achievable. So, in other words, the goal of Q-learning is to find the optimal policy by learning the optimal Q-values for each state-action pair obtained by interaction with the environment.
The main idea is to iteratively update the Q-values for each state-action pair getting from the interaction with the environment, without iterating the entire state,action spaces, using the Bellman equation until the Q-function converges to the optimal Q-function. This approach is called value iteration or tabular Q-learning.

The algorithm is obtained as follows:

1. Start with an empty table, mapping states to values of actions.
2. By interacting with the environment, obtain the tuple s, a, r, s' (state, action, reward, and the new state). In this step, you need to decide which action to take, and there is no single proper way to make this decision.
3. Update the $Q(s,a)$ value using the Bellman approximation:

$$Q(s,a) \leftarrow r + \gamma \max_{a' \in A} Q(s',a')$$

4. Repeat from step 2.

The end condition could be some threshold of the update, or we could perform test episodes to estimate the expected reward from the policy [10].

Actually, the update rule as reported in step 3 is not used in practice. In general, it is a bad idea to just assign new values on top of existing values, as training can become unstable. What is usually done in practice is updating the $Q(s, a)$ with approximations using a *blending* technique, which is just averaging between old and nerw values of Q using learning rate $\alpha \in [0, 1]$:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a' \in A} Q(s', a'))$$

This allows values of Q to converge smoothly.

3.2.6 Deep Q-Learning (DQL)

The Q-learning method reduces iteration over the full set of states, but still can struggle with situations when the count of the observable set of states is very large. Indeed, storing too many pair (state, action) in the Q-table could become unsustainable.

As a solution to this problem, DQL translates the problem to a *regression* ones. The idea is to build a deep neural network in order to learn a non-linear function that maps both the state and action onto a value. The method requires to interact with the environment in order to get data to train on. One method of acting is the *epsilon-greedy method*. It consists to switch between random and Q policy in according to an hyperparameter ϵ. By varying ϵ we can select the ratio of random actions. The usual practice is to start with $\epsilon = 1.0$ (100% random actions) and slowly decrease it to small values. In this way, the algorithm explores the environment in the beginning and stick to good policy at the end of the training.
Another problem to be solved is that *supervised models* require that the samples are *independent and identically distributed (i.i.d)*, this feature is reached by using the *replay buffer* technique.
Replay buffer relies on the concept of experience which can be considered as what the agent learned from the previous state of the environment. More specifically the experience acquired at time t can be defined as:

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

this means that it is a collection of the current environment state, the performed action, the resulting reward and the next state in which the agent is after having performed the action. The current experience of each step is stored in a memory with finite capacity N. This memory randomly sampled to remove the correlation between the subsequent steps is used as the training dataset of a neural network. Another issue is related to the natural form of the Bellman approximation. It provides us with the value of $Q(s, a)$ and $Q(a', s')$, where both states s and s' are distanced by only one step. This makes them very similar, and it is hard for a NN to distinguish between them. To make training more stable there is a trick, called *target network*. It consists to keep a copy of the used network and using it for getting the value of $Q(s', a')$ to be used in the Bellman equation. This network is synchronized with our main network only periodically, by defining a number N of training steps N, after which network's weights are shared between the two models.

Epsilon-greedy, replay buffer and target network are some tricks used to efficiently training a DQN. These methods were used in the DeepMind project for training a DQN on a set of 49 Atari games, with substantial advantages [12].
The algorithm for DQN from the preceding papers has the following steps:

1. Initialize the parameters for $Q(s, a)$ and $\hat{Q}(s, a)$ with random weights, $\epsilon \leftarrow 1.0$, and empty the replay buffer.

2. With probability ϵ, select a random action a; otherwise $a = \arg\max aQ(s, a)$.
3. Execute action a and observe the reward r, and the next state s'.
4. Store transaction (s, a, r, s') in the replay buffer.
5. Sample a random mini-batch of transactions from the replay buffer.
6. For every transaction in the buffer, calculate target $y = r$ if the episode has ended at this step, or $y = r + \gamma \max_{a' \in A} \hat{Q}(s', a')$ otherwise.
7. Calculate the loss: $\mathcal{L} = (Q(s, a) - y)^2$.
8. Update Q(s, a) using the SGD algorithm by minimizing the loss in respect to the model parameters.
9. Every N steps, copy weights from Q to $\hat{Q}$.
10. Repeat from step 2 until converged.

Now, we can describe the DQN model that interacting with a *Blockchain* environment learns how to assign a value to nodes describing their utility to the network.

3.2.7 DQN utility model

In this section the DQN utility model is described. First of all states, actions and rewards are defined. Then, the Deep-Q-Network architecture is described, together its hyperparameters and metrics. Finally, a simulation with its results are described.

In this scenario, nodes (agents) have to select an action for each state. States are samples reporting some features about the behaviour of each node. Actions indicate how much a node has the right to participate to the final consensus. Some general consideration are needed.
A *Reinforcement Learning (RF)* model requires a fundamental training phase, in which *agents* learn the best way to take actions in order to maximise a *reward signal function*. In an Atari game, errors mean nothing else than losing a game, in a blockchain system, they could mean loss of stakes. For this reason, it has been decided to train the network in a *Test-net* phase, where users can join the network, making transactions by using *TestNet Coins*. In this phase, users have to select one of the possible behaviours scenarios, at the beginning of their connection:

1. *Honest Player (HP)*: the user joins the network with the intention of behaving as an honest player, exchanging transactions with other users, without never acting in a malicious way.
2. *Bad Player (BD)*: the user joins the network for making attacks to the network.

In the TestNet phase, each node (*agent*) collect in a non-interactive way, features about other nodes (*agents*). These features compose the *states* of the *environment*. Features, could change, in a Test Net phase, if the network does not converge to a fixed threshold, described later on. At the beginning the selected features are:

- *UpTime:* the total time a node results to be connected to the network.
- *OutLink Matrix:* as described in 3.1.1.
- *Vested Balance:* it is the node's total balance in times, defined as follows:

$$\rho = \beta \cdot \exp(T),$$

  where β is the current balance, T is the total time such balance is present.
- *Total number of TCP connection (TTCP)*

From the features, a state is defined as a 1D tensor of dimension 5, as follows:

$$\mathbf{s} = [\mathit{UpTime}, \mathit{Outlink\ Matrix}, \mathit{Vested\ balance}, \mathit{TTCP}, \mathit{Behaviour}]$$

where *Behaviour*, is the selected behaviour scenario each node must choose at the beginning: 1 for HP, 0 for BP.

Agents must choose a particular *action*, from the available ones. While, *states* can change, actions are always the same for all TestNet phase. They are as follows:

1. participate with 100% tickets: 0;
2. participate with 75% tickets: 1;
3. participate with 50% tickets: 2;
4. participate with 25% tickets: 3;
5. participate with 0% tickets: 4

Rewards: agents get rewards for their actions. At the beginning, the system select two states, representing the best behaviour ($\mathbf{s}_{\text{Best}}$) and the worst behaviour ($\mathbf{s}_{\text{Worst}}$) a node can show. These states are used for calculating the rewards. All possible scenarios are as follows: for each given state a agent must select an action

- If the current state belongs to a HP (*Behaviour == 1*), an agent should choose an action in the set [0,1,2]. In particular, an *Euclidean distance* is computed between the given state and the $\mathbf{s}_{\text{Best}}$. If the distance is grater than 2, and the agent has chosen the action 1 or 2, it receives a reward of +1, otherwise it receives 0 rewards. If the distance is less than or equal to 2, if the agent has chosen the action 0, it receives a reward of +1, otherwise 0 rewards.
- If the current state belongs to a BP (*Behaviour == 0*), an agent should choose an action in the set [3,4]. In particular, an *Euclidean distance* is computed between the given state and the $\mathbf{s}_{\text{Worst}}$. If the distance is grater than 2, and the agent has chosen the action 3, it receives a reward of +1, otherwise it receives 0 rewards. If the distance is less than or equal to 2, if the agent has chosen the action 4, it receives a reward of +1, otherwise 0 rewards.

The *Behaviour* feature is used only in the TestNet phase, and if the system will converge, and a MainNet phase can start, such feature will not be more present, but the network has learnt how to choose the right best action given the states represented by the selected features. In that phase, rewards will be not more necessary, since the network has already been trained, and nodes having the DQN's weights will be able to predict the correct action, given a state.

3.2.8 DQN Architecture

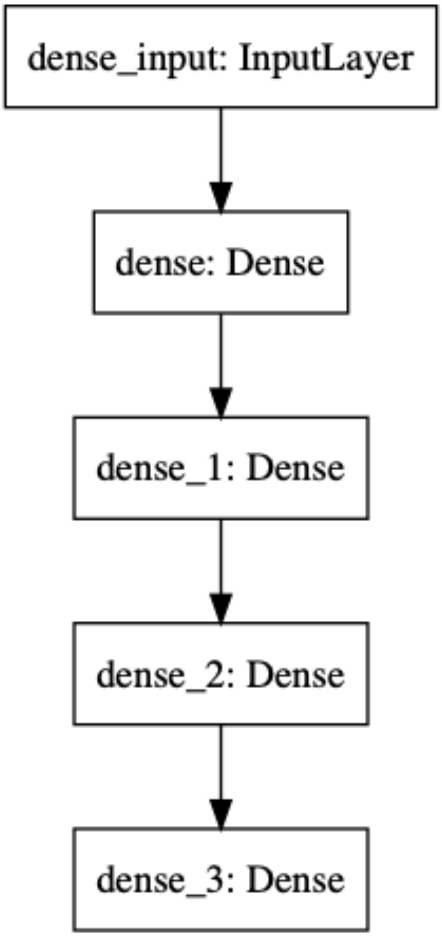
The DQN model has been conceived looking at the nature of the states. Since, the states are 1D tensor, and the problem to be solved is a *regression* one, its has been decided to build a *Fully-Connected-Neural Network*. The architecture is shown in figure 3.2b: it is composed by three hidden layer and one final layer with a *linear* activation function, for a total of 929 trainable parameters. All the three hidden layers have a *ReLu* function as activation function, for reducing likely problem related to the *vanishing gradient* problem [14]. The input layer takes as input vectors $\mathbf{s} \in \mathbb{R}^5$, while the final output layer outputs vectors $\mathbf{s} \in \mathbb{R}^5$, with a dimension equal to the dimension of the possible actions. A second architecture is used, where there is the addition of Batch Normalization layers, as shown in figure 3.2d. For the model 3.2c, the total trainable parameters are 1037.

Figure 3.2: *Models architectures and summaries.*

(a) *Model 1 Summary: L1 and L2 regularization only.*

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 14)                84
_________________________________________________________________
dense_1 (Dense)              (None, 24)                360
_________________________________________________________________
dense_2 (Dense)              (None, 16)                400
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 85
=================================================================
Total params: 929
Trainable params: 929
Non-trainable params: 0
```

(b) *Model 1 architecture: L1 and L2 regularization only.*



(c) *Model 2 Summary: L1, L2 and Batch Normalization regularizations.*

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 14)                84
_________________________________________________________________
batch_normalization (BatchNo (None, 14)                56
_________________________________________________________________
dense_1 (Dense)              (None, 24)                360
_________________________________________________________________
batch_normalization_1 (Batch (None, 24)                96
_________________________________________________________________
dense_2 (Dense)              (None, 16)                400
_________________________________________________________________
batch_normalization_2 (Batch (None, 16)                64
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 85
=================================================================
Total params: 1,145
Trainable params: 1,037
Non-trainable params: 108
```

(d) *Model 2 architecture: L1, L2 and Batch Normalization regularizations.*

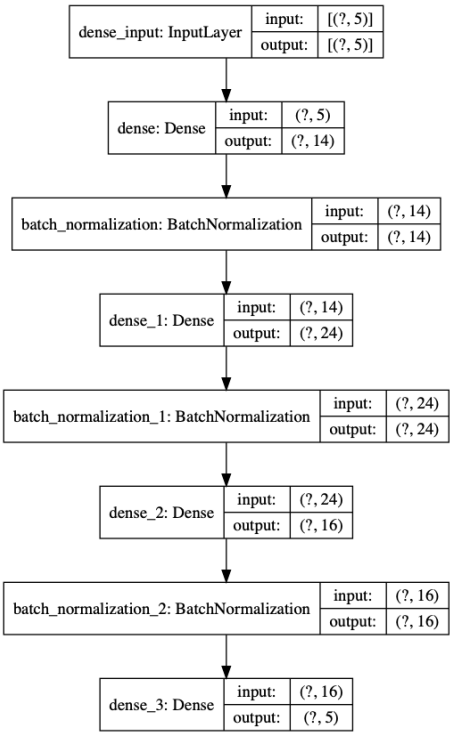


Table 3.2: *Hyperparameters of the learning model.*

Type	Description	Starting value
learning rate	its value define the learning rate of the deep neural network	0.001
discount rate	its value define how rewards of future states are important	0.99
exploration rate	its value define the trade-off in the *Epsilon-greedy* technique	1
decay exploration rate	rate of decay for the exploration rate	0.92
minimum exploration rate	it defines the minimum possible value for the exploration rate	0.001
NShareWeights	it gives the number of steps after which sharing weights between the regular model and the target model.	SIM_TIME*0.56
memory dequeue	buffer size for saving samples batch to be used in the *Replay Buffer* technique	2000

3.2.9 Hyperparameters and Metrics

The *learning* model requires six hyperparameters, as shown in the table 3.2. The learning rate is related to the DQN, while all the others are related to the *Reinforcement Learning* field. *NShareWeights* defines the number of steps after which the target model's weights are updated with the weights of the primary model. SIM_TIME is referred to the simulation time.

Huber-Function: as *loss function* the *Huber-loss* is used. It graph is shown in figure 3.3. It is defined as follows:

$$\mathcal{L}_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{3.4}$$

where, $a = y - \hat{y}$ is the residual, or error. The δ is called *clip delta*, it defines the trade-off between a quadratic behave or a linear ones. The Huber loss, acts as a quadratic one for small error (less than δ), while it acts as a linear function for errors smaller than the clip delta. In particular, residuals larger than delta are minimized with L1 distance or Mean Absolute Error (MAE) (which is less sensitive to large outliers), while residuals smaller than delta are minimized "appropriately" with L2 distance or Mean Squared Error (MSE). This loss has shown significantly advantage used in the context of *Reinforcement Learning* [13].

In RL models another significant metric is the *total rewards* gained for episode. We focus on average total rewards for episode, for evaluating the correctness of the DQN to choose the *optimal Q value*.

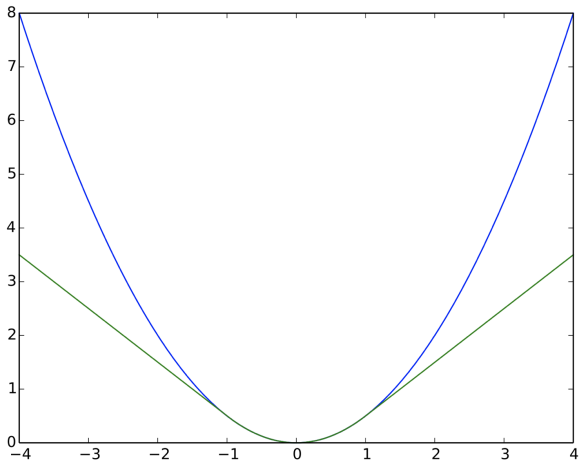


Figure 3.3: *Huber Loss: it behaves as a quadratic function for small residual, and as a linear function for big residual.*

3.2.10 Regularization

The model is trained using four types of *regularizations*. First, L1 and L2 regularizations are used, in kernel, bias and activity layers. These layers expose 3 keyword arguments:

- kernel_regularizer: Regularizer to apply a penalty on the layer's kernel
- bias_regularizer: Regularizer to apply a penalty on the layer's bias
- activity_regularizer: Regularizer to apply a penalty on the layer's output

In particular, in *kernel_ regularizer* it is used both L1 and L2, with L1 and L2 penalties equal to $1 \exp -4$, while for both bias_regularizer and activity_regularizer it is used only L2 with penalty equal to $1 \exp -4$. The architecture model for this configuration is shown in figure 3.2b.
The second model uses Batch normalizations layers, for avoiding *covariance shifting*, [15]. Finally, a combination of *Dropout* and *L2 regularization* are used, following the recent good results obtained in [18], while it has been avoid to use both Batch normalization and Dropout together for their inconsistency showing in *learning algorithm*, as reported in [16].

3.2.11 Train Main Loop

The Train loop is equal to the one described in 3.2.6. The loop is an iterative one, over *EPISODES* and *TIME*. The global variable EPISODES defines the number of episodes, while TIME define the maximum simulation time each episode can be long. For each episode, after $N = TIME * 0.64$ the target model update its weights with those of the primary model, therefore the episode stops, and a new episode is started. Agents take action following the ϵ-greedy strategy, for which an agent chooses an action in a random way with probability ϵ, and in a predicted way, with probability $1 - \epsilon$. The *exploration rate* ϵ is reduced at each step, by multiplying itself with the $\epsilon_{\text{decay}} = 0.92$. If a minimum loss, equal to the LOSS_TH global constant, is reached, the loop is break, and weights are shared.

3.2.12 Simulations

In order to evaluate the DQN model, a simulation environment is created. The two BEST and WORST states, are selected, and from them, a method generates a *dataset*, simulating

nodes' states, in a random way, such that to create a equal distributed dataset, contained 50% of states near to the BEST configuration and 50% near to the WORST configuration. Such dataset is split in a training, validation and test one. In particular, 80% is used for training (of which 20% for validation), the remained 20% is used for the test.
The obtained results are described later one.

3.3 Conflict of Interest (CoI)

In this section, an additional micro-services is described. It could be useful for increasing *Fairness* in the network, avoiding to have always the same nodes forming the consensus committee. Actually, its utilization is quite general and easily adaptable for several situations rising in a distributed consensus mechanism.

We want to reach a *consensus* as fair as possible, avoiding centralization scenario, in which only few nodes have a consensus power equal or greater of the 51 %.
The Fairness in the consensus is reached when all honest nodes have the same probability to be part of the final board. However, not all nodes, even if honest, have the same utility in the network. For instance, not all nodes have the capability to cache all the block history in memory, or not all nodes want to be on-line 24h per day[1]. Therefore, it rises the problem of individuating a sub-set of nodes with specific features, able to assure as fast as possible block creation. But, focusing only on a such utility may take an unbalance to the boards created for validating blocks. This is solved by introducing the concept of Conflict Graph.

Conflict Graph (CG): relying only on the utility factor could lead to an unbalance situation in which always the same nodes will be part of the final Board. In order to avoid this, and therefore guaranteeing a level of fairness as high as possible, a *Conflict Graph* is used. A CG $\mathcal{G}$ is defined as the set of $\{V, E\}$ where V is the set of nodes and E the set of edges. An edges $e_{i,j}$ between the node i and the node j, exists if these nodes may have a *Conflict of Interest (CoI)*. A CoI exists when two nodes could have either some interests in common or they have experienced some event together[2].

With this in mind the mathematical model that will output which nodes could be part of the next board[3] is as follows:

$$\max. \quad \sum_{i=1}^{N} U(a_i) \cdot x_i \tag{3.5}$$

$$s.t \quad x_i + x_j \leq 1 \quad if \quad (i,j) \in CG \tag{3.6}$$

where

- $U(a_i)$ is the *utility* associated to the node a_i
- $x_i \in \{0,1\}$, $i = 1, \cdots, N$, where N is the total number of nodes for which an utility exists, are the binary decision variables
- CG is the *Conflict Graph*

[1] these are all crucial assumptions needed for assuring a block creation.
[2] for example they have been both members of previous board
[3] and so can participate to the lottery.

Such problem is known in literature as the *Maximum Weight Independent set* [1]. This is a well-known NP-hard problem and several possible solutions and algorithms were proposed. Rennes [4] proposed an intuitive and straightforward greedy algorithm, which at each iterations adds the minimum degree vertex to the solution and deletes the neighbours from the solution space. In this paper, a similar approach is discussed considering different weights of the vertices. Goldberg and Spencer [6] gave a parallel algorithm for MIS which finds an independent set of size at least $\frac{n}{d_G+1}$ [6]. If a maximum total weight is considered, the problem reduces to the disjunctively constrained knapsack problem which can be solved with the heuristic proposed by Hifi and Otmani [7].

Now, two different heuristics to solve the MWIS problem are presented.

MWIS - Dynamic programming (MWIS-DP)

Here, an heuristic for solving the MWIS problem is described. The algorithm exploits the *dynamic programming* approach, in which the entire macro-problem is solved by solving several smaller sub-problems.

The main algorithm's idea is that, for each *sub-graph* defined as the conflict graph that a node defines with its conflict nodes, the node with the maximum weight is selected as solution while all others are deleted.

The algorithm explores all the graph node by node. For a given node i, It computes the i's conflict nodes, in order to compute a list with the weights of each of them, plus the weight of the node i. From this list the maximum value is selected, and the node with this weight value is chosen as solution, while all others are deleted from the initial graph. The algorithm allows to solve the MWIS in a *polynomial time* instead of an *exponential time*, however it obtains an accuracy lower than the optimal case, of more or less 40% of the optimal solution, as it will be described later.

The *PSEUDO-CODE* of the algorithm is as follows:

Algorithm 3: MWIS pseudo-code

input : A weight graph G.
output : A maximum independent subset of G

```
1 initialize;
2 conflictNodes = ∅;
3 foreach n ∈ G do
4 |   conflictNodes ← N_G(i);
5 |   totalUtility ← w(N_G(i)) + w(i);
6 |   select the maximum weight ;
7 |   select the corresponding node which has as weight the maximum value;
8 |   delete from G all other nodes;
9 end
```

Recursive approach:

Here, another heuristic for solving the MWIS is described. The algorithm is a recursive one. The approach is more or less similar to the previous one, but in order to improve the accuracy with respect to the previous one, one additional condition is imposed.

While, in 3.3, for each sub-conflict graph, the optimal node is selected as the node with the maximum weight, here a sort of utility metric is used. For each selected node i and the corresponding sub-conflict graph $N_G(i)$, the node i is selected as the optimal one if:

$$w(i) > \frac{U_T}{n}, \tag{3.7}$$

where $U_T = \sum_{j=1}^{n} w(j) \quad j \in N_G(i)$ is the total sum utility obtained as the sum of the weights of the nodes in conflict with the node i, n is the number of conflicts and $\mu \in [0, 1]$ is an hyper-parameter to be chosen. Otherwise, only it is deleted from the started conflict graph G and the algorithm restarts in a recursive way.

This solution allows to reach a better accuracy with respect to the 3.3 algorithm, because the used *criterio* for choosing the optimal solution is more restrictive, and at each step the maximum number of nodes deleted may be: 1 if the query node does not respect the condition; n-1 otherwise. Therefore a better solution is obtained, however more time is needed in order to solve the problem.

The *CoI* procedure is a service that can be modified in according to the particular *environment* in which the blockchain system is used. The concept of *Conflict of Interest* can depend on the context of the specific application. For example, in an application where *Smart Contracts* are exchanged, a CoI can arise when in the committee there are members associated with the organization that is proposing this same smart contract. In this case the evaluation of the proposing smart contract can be influenced by a clear personal interest, avoiding to have an impersonal evaluation.

3.4 Board Election Algorithm: Cryptography Lottery

At the end of the CoI procedure, a particular sub-set of the entire network is selected. Nodes inside this sub-set have the following features:

1. each of them has a balance of at least $\mathcal{B}$;
2. each of them has an utility factor associated;
3. each of them has not conflict of interest between each other.

At this stage, nodes being part of this particular sub-set , have a probability to be part of the committee board, and so be part of the consensus procedure, in according to their utility.
In order to select the boards committee in a uniform random way, a *Cryptography Lottery CL* is executed. It is based on the mechanism of linking an extracted integer to a point of an Elliptic curve. The lottery outputs are integers, these integers indicate how many operations (additions) must be done on the chosen Elliptic Cryptography Curve in order to individuate a point on the curve associated with a specific wallet: the owner of that specific token. The steps needed to perform such task are as follows:

1. **Environment:**

A particular Elliptic Cryptography Curve is selected: ECC of type Secp256k1:

$$y^2 = x^3 + 7$$
$$P(x1, y1);$$
$$Q(x2, y2);$$

The Secp256k1 is a 256-bit Elliptic Curve defined over a finite prime field $\mathbb{Z}_p$.
Its recommended parameters are specified by the sextuple $T = (p, a, b, G, n, h)$ [5], where:

- $$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$
$$= \texttt{FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC2F}$$
- The base point, in compressed form is:
$$G = \texttt{0279BE667EF9DCBBAC59F2815B16F81798}$$
- The *order* n of G is:
$$n = \texttt{FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141}$$

The ECC fixes a mathematical space where executing some operations in order to find the winner tickets. The algorithm gives some integers as output, these integers are interpreted as the extracted tickets giving the number of *operations* to do on the ECC, starting from the known points P and Q. In particular, points addition are the operations to be executed.

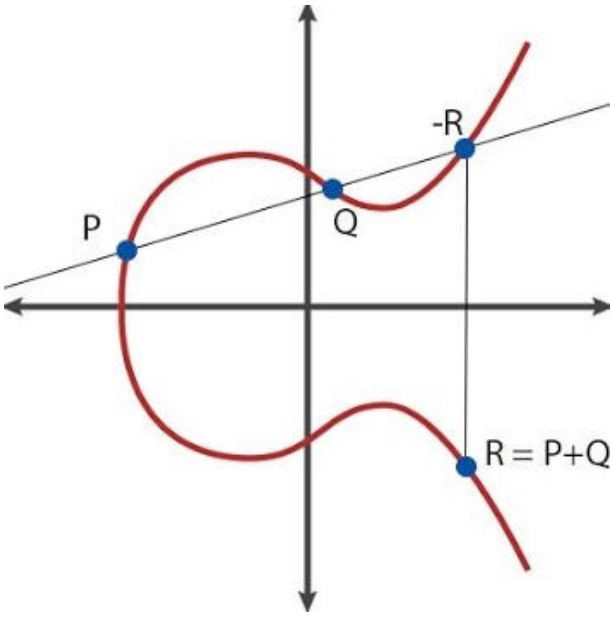


Figure 3.4: *Points addition over elliptic curves.*

2. Ticket Generator
The maximum number of tickets [1...N], to be extracted, must be chosen by the systems. This is a parameter that will change in time, depending on the level of *inflation*.
The algorithm described in 3.4, will be executed, in a independent way, by each node having an

utility. The number of possible tickets, each node can extract, depends on its utility: more high is the utility more tickets can be extracted.
Each extracted ticket is an integer that represents the number of operations to be done on the ECC to find a point on the ECC representing a specific token:

$$\begin{aligned} P + Q = -R = R \quad &[\text{Ticket 1}] \\ P + R = -E = E \quad &[\text{Ticket 2}] \\ P + E = -F = F \quad &[\text{Ticket 3}] \\ \vdots = \vdots = \vdots \\ P + \text{ticket}_{N-1} = -\text{ticket}_N = \text{ticket}_N \quad &[\text{Ticket N}] \end{aligned}$$

If the extracted ticket is 1, the known parameters P and Q must be added once. In general if the extracted number is n the winner token is the point of the ECC individuated as $nP + Q$. Thanks to the nature of the ECC those points (tokens) are all well separated on the plane.

3. Select the Leader: Now, the core algorithm is described. It can be defined as a *deterministic random algorithm*, because given the same input its output are always the same, but the possible output are uniform distributed between all the possible cases, as it will be shown later on.

Starting for the hash of the last block that is written into the Blockchain do the following steps:

1. Starting hash: 4ceb86317d0d4dac6853663589ef02ccb67134cee75bb886a4410b7aedd0e109
2. Split the hash in 4 parts:
   (a) 4ceb86317d0d4dac
   (b) 6853663589ef02cc
   (c) b67134cee75bb886
   (d) a4410b7aedd0e109
3. Transform each part into binary
   (a) 0100110011101011100001100011000101111101000011010100110110101100
   (b) 0110100001010011011001100011010110001001111011111000000101100 1100
   (c) 1011011001110001001101001100111011100111010110111011100010000110
   (d) 1010010001000001000010110111101011101101110100001110000100001001
4. Compute the XOR between [part 1 / part 2] and [part 3 / part 4]
   - Partial Result 1:
     00100100101110001110000000000100111101001110001001001111011000 00
   - Partial Result 2:
     00010010001100000011111111011010000001010100010110101100110001111
5. Compute the XOR between [partial result 1 and partial result 2]
   - Final Result :
     11011010001000110111111011000011111110011010010001011011011 01111
6. Transform the Binary number into a decimal number and compute the modulo N function
   - Winner (Leader): 919

Ticket n. 919 is the extracted number. The winner token is the ECC's point reachable by computing the operation: $919P + Q$. The elected leader is the address having the ownership of the individuated token.

4. Board Members

We could have several approaches to define the board members. For instance we could run the functions described into point 3 in a deterministic way (like splitting the hash in different parts) or more easily starting from the Leader ticket simple count another m (number of board members) consecutive tickets (ex. 919, 920, ... m-1, m). The fact that they are consecutive tickets do not matter because they are representing points that are far away one to each other.

The number of committees members is in the range $[4, 1000]$. In particular, at the begging, the system needs at least 4 nodes to start, therefore the maximum number of members is fixed to 4. As the number of nodes increases, the number Nc of committee members increases too, as follows:

$$Nc = \gamma \cdot n^2$$

where, n is the current number of nodes, γ is a fixed parameter used to constrain the number of members to be at maximum 1000. The γ's value will change in time. Until the number of nodes is less or equal to 10000, γ will be equal to $1e - 5$. As the number of nodes increases γ is adjusted in order to limit the maximum number of the committee's member to 1000.

Regarding the number of tickets N_T each lottery has to distribute, they are obtained following an exponential function, as follows:

$$N_T = \delta \cdot \exp(n)$$

where n is the current number of nodes, δ is a fixed constant used to have control on the shape of the exponential function. At the beginning, $\delta = 1$, as the number of nodes increases, its value is decreased, in order to limit the creation of tokens.

3.4.1 Is the Cryptography Lottery algorithm a Pseudorandom Generator?

In order to verify if the Board Election algorithm is a Pseudorandom Generator (PRG), some simulations were carried out. In particular, 10 simulations were made, each of whom is composed as follows: there are 10 independent experiments (lottery extractions) and in each of them the same number of tickets have been extracted, varying the percentage for each simulation. In particular, only the 10 % of the 1000 tickets, for each experiment, in the first simulation, has been used ; 20%, for each experiment, in the second simulation; 30% for each experiment, in the third simulation , and so on, until arrive at 100% in the last simulation. Therefore, there are a total of 10 simulation, and 10 experiment for each of them, for a total of 100 experiments.

From these simulations all data have been collected, in order to build some statistical graphs. In particular, it has been chosen to use a **Density curve Plot** instead of an histogram, to understand the underlying probability distribution of the data. A density curve attempts to visualize the underlying probability distribution of the data by drawing an appropriate continuous curve. This curve needs to be estimated from the data, and it has been used, one of the most frequent method, called *kernel density estimation* to do that. In kernel density estimation, it is drawn a continuous curve (the kernel) with a small width (it is controlled by

a parameter called *bandwidth*) at the location of each data point, and then it is added up all these curves to obtain the final density estimate. Density curves are usually scaled such that the area under the curve equals one. This means that on the Y axis there is not a probability but a density probability function for the kernel density estimation.
Regarding the bins to be used in the density plot graphs, it has been chosen to use the Freedman-Diaconis rule (FD rule). The FD rule is designed to minimize the difference between the area under the empirical probability distribution and the area under the theoretical probability distribution. Each bin width is computed as follows:

$$Bin_{width} = 1\frac{IQR(x)}{\sqrt[3]{n}}$$

where IQR(x) is the *interquartile range* of the data and n is the number of observations in the sample x.

Moreover, the **Empirical Comulative Density Function (ECDF)** has been obtained. It is simply an empirical realization of a Comulative Density Function. The terms empirical is referred to the fact that the function is made up by using empirical measure of a sample. The empirical distribution function is an estimate of the cumulative distribution function that generated the points in the sample. It is defined as follows:
Let $\{X_1, \cdots, X_n\}$ be real random variables i.i.d[4] with common Comulative Distribution Function F(t). Then the ECDF is defined as

$$\hat{F}_n(t) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{X_{i\leq t}},$$

where $\mathbb{1}_A$ is the indicator of event A. It can be proved that the ECDF is an unbiased estimator for F(t).

All the resulted plots are shown in the appendix 5.2.1. The algorithm outputs seems to follow an uniform distribution, as expected from a PRG.

[4]independent, identically distributed.

Obtained results

In this chapter the NFCP system is shown, at its actual state, where the logical skeleton composing the blockchain is done, as well as the *Reinforcement Learning* model.
An environment for performing simulations has been created. The objective is to show the entire working flow of the blockchain. Simulations results are showed as well as the evaluations of the *Deep-Q-Network* model. Regarding the blockchain environment, a novel technique for managing the entire storage is used. Each new validated block is stored in a *IPFS* cluster [45], allowing to maintain the entire status of the blockchain distributed around the world, in order to allow even mobile nodes to be participated in the blockchain ecosystem without the needed of storaging big amount of data on their own memory.
IPFS is a peer-to-peer hypermedia protocol with the aims to overpass the actual limitations of the HTTP. The main feature IPFS offers is a way to retrieve contents from the web in a distributed way, overpassing the problematic situations happening in a *central-server* environment. In a Nutshell, IPFS can be thought as a distributed file system, where each updated content is individuating by a unique code of identification, its own *fingerprint*: *Content Identifiers (CIDs)*. In the HTTP web, each content is obtainable via its own URL, that indicates *where* the content is and nothing about its nature. For example, if today an URL gives a content, tomorrow the same URL could give a different content. This implies that, a same content can be retrieved using different URL, producing duplications. IPFS uses content addressing to identify content by what's in it, rather than by where it's located. In IPFS each content has its own fingerprint: its own CID. The CID is not a URL for reaching a certain content, but it is the unique code that identifies a certain content, in a unique way. Thus, if two same contents are updated on IPFS, from two different places, they will have the same CID, avoiding duplicates. Continuing with the example, these two same contents, will be stored in a distributed way on the peer-to-peer network. In particular, the content is split in several blocks, each block has its own CID, and is saved on a IPFS peers. When a peer wants to get the content, the list of peers that have the blocks forming the content are fund, using a *Distributed Hash Table*, and the entire content is restored, because a *Direct Acyclic Graph* has been used, for linking the content's blocks together.

Now, let's describe the logical steps to follow for initializing the blockchain.
In order to start the blockchain system, the following steps must be done:

- *Settings:* a config file of the Blockchain is edited, and stored on the IPFS cluster. It contains the hash of the *Block Genesis*its CID, number of tokens to generate at the next mining process. The number of token to be distributed with the *mining* process, is established, following rules reported in 3.4.
- *Start a new Blockchain:* The network will start with a number $n = 4$ of nodes with initial 0 tokens, that will start to communicate. In particular, nodes start to collect Features (states) from peers, to be used with the DQN model.
- *Genesis:*

1. Get the number of token to distribute, from the config file: Es. 40
2. Run the lottery for extracting 40 tokens using the *Genesis Hash* as retrieved from the config file. These 40 tokens will be pre-distributed to the nodes, in a random way. Nodes receiving these tokens do not have yet their ownership. The order of extractions decides the *committee*. In particular the node that receives the first extracted token becomes the Leader, the other members are naturally individuating as the nodes receiving the subsequent tokens. At this step, nodes have received tokens in a random way, and some node may have received more tokens than other.

- *Consensus Cycle*:
  1. The Leader, with the data that it knows (collected features) in that specific moment, run the DQN model. The outputs are actions to assign to other peers. These actions define the real *power of voting* of each peers. In particular, now each peer have just a percentage of the pre-received tokens, as established from the action it received. If for example, node A has pre-received 20 tokens, but the DQN model has assigned a *power of voting* equal to 25%, its real number of tokens will be 5, consequently its *voting power* used in the *Soft Vote* is of just 5.
  2. Now, the leader proposes the transactions it collected. Each committee peer validates such transactions, by checking *double-spending* and *ownership*, and sends its vote to the leader. The Leader verifies all the votes are coming from committee members, by checking the signature, and weights the received votes with the *voting power* of each nodes. Thus, if a node has sent 10 positive nodes, but its voting power is 0%, such positive votes are excluded from the final consensus, therefore this *bad node* is automatically excluded from the final consensus thanks to the DQN model. In other words, the network is robust against *Byzantine Faults* because it is able to detect *malicious nodes* and excluding them from the final consensus. Finally, the Leader computes the final consensus as

     $$\text{consensus: Boolean} = \text{Positive votes/Total votes} > \text{CONSENSUS\_RATIO}$$

     . If consensuns = $True$, the block is signed by the leader and sent to all other peers, otherwise the blocks is rejected.
  3. If the consensus is true, Peers verify the signature of the received block (*Verify Hard Vote step*), for assuring the proposed block is coming from the Leader, and the block is added to the blockchain, stored on the IPFS.

- *Future Steps:*
  1. From the new generated block hash run the lottery and execute the steps as reported in the *Consensus Cycle*.

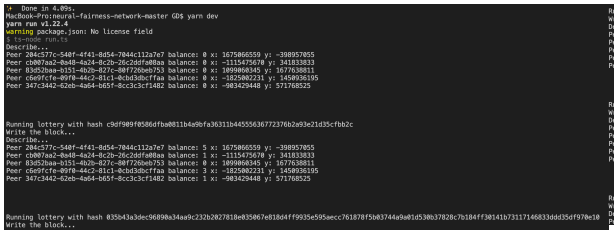
4.0.1 Blockchain Application

In figures [4.1a, 4.1d] the Blockchain simulation as described above are shown. The Blockchain starts with 5 peers, all with a 0 initial balance, and their public key is reported. Starting from the *genesis* hash the lottery is run, and some number of tokens are distributed between peers. The Leader is individuating and the block creation is executed, where some times the process can reject some block, as happens in figure 4.1d, meaning the consensus has not been reached. In figures 4.2a the client of the IPFS cluster is shown. Here, it is possible to visualise the cluster status, checking the coming and outgoing traffic. The cluster manages the storage of the blockchain. In the folder *neural-fairnes-network*, there are the config file and all the validated

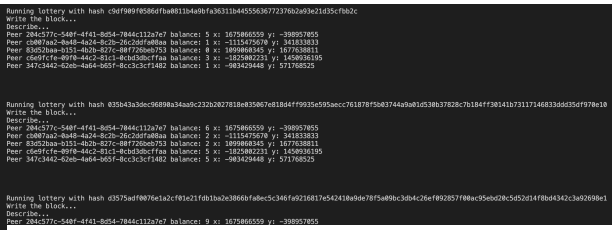
blocks.

Figure 4.1: *Simulation Blockchain.*

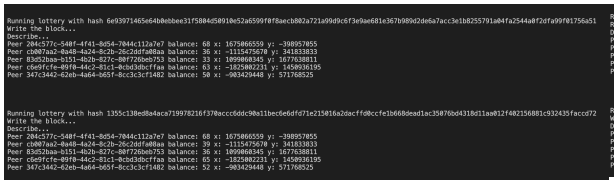
(a) *Experiment 1: the Blockchain starts*



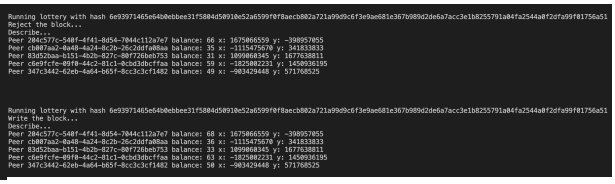
(b) *Experiment 2: nodes start to mine blocks*



(c) *Experiment 3*



(d) *Experiment 4*



While the NFCP is at its *alpha* state, where it is possible to run simulations and evaluating its DQN model, as well as verifying how the IPFS storage is working, an initial web application, is done. At this state, from the web application is possible to create a wallet, make real transaction (relying on the NEM blockchain), storage some encrypted stake on the IPFS cluster.

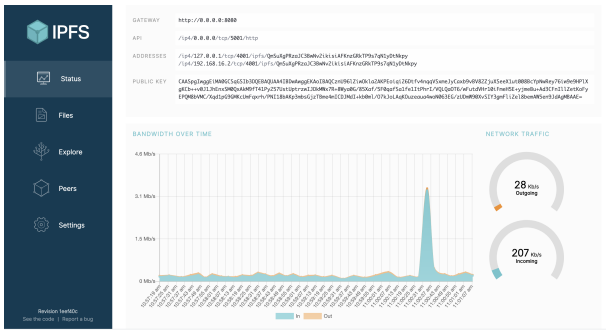
4.0.2 DQN evaluation

Here the evaluation of the DQN model is shown. Several experiments have been done, and results about two types of configurations are shown. The two models differ due to the selected *regularization* used. In particular, three configurations have been tested: only Batch normalization, Batch normalization and L2 regularization, Dropout and L2 regularization. The showed metrics are, as described in 3.2.9, the Huber Loss and the total gained rewards together the total rewards score defined as the ration between the total rewards and the number of executed steps. The total number of episodes is 3, while the maximum time an episode can long is 2000.
In figures In figure 4.4a the result of the model using only Batch normalization (BN model), are shown, while in figures [4.5a, 4.5e] the results of the model using both Batch normalization and L2 regularization (BNL2 model), are shown. Finally, the results about the model using both Dropout and L2 regularization (DL2 model) are shown in figures [4.6a, 4.6e].

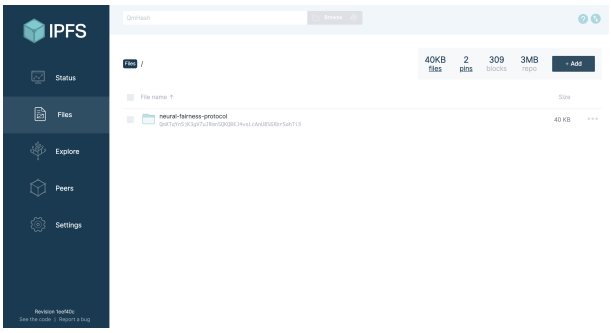
The best model, in according to the selected metrics, is the DL2 model, showing a total rewards score of more than 87%, in the third episode, showing a constant increasing of the score between episodes. The worst model is individuating with the BN model, where it has been decided to stop the simulation at the first episode since there were not improvements. The intermediate model, is the one using both Batch normalization and L2 regularization (BNL2 model), reaching a total rewards score of no more than 48%.
Regarding the model's losses, already in the first episodes both DL2 and BNL2 reach a minimum average loss of more or less 0.04, saturating around this value. For this reason the losses of the other episodes are not reported. The loss results shown that likely the architecture model needs to be reconsidered, in order to reach lower values, or the number of steps after which updating the weights of the *target model* must be tuned.

Figure 4.2: *Simulation Blockchain: IPFS.*

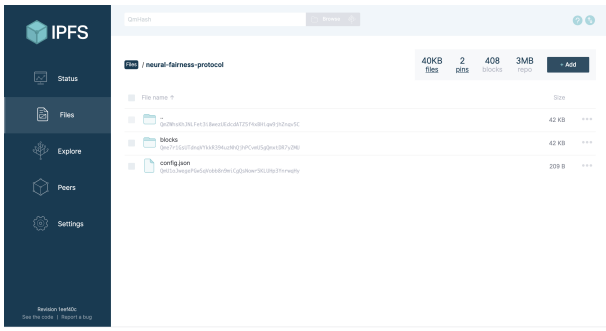
(a) *IPFS Cluster Status*



(b) *neural-fairness-network directory*



(c) *contents of the neural-fairness-network*



(d) *All validated blocks of the blockchain: each of them has its own CID*

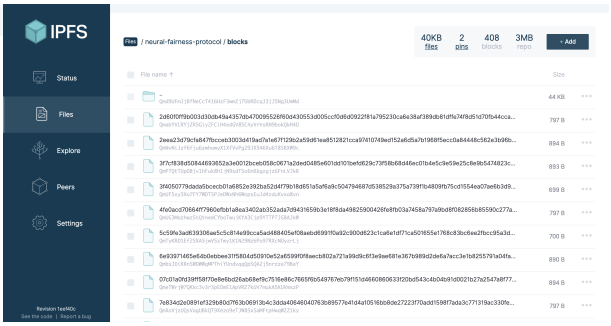
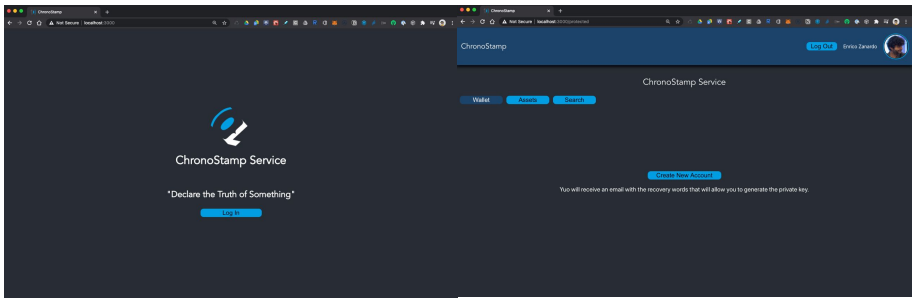


Figure 4.3: *Front-End of the Applications:*

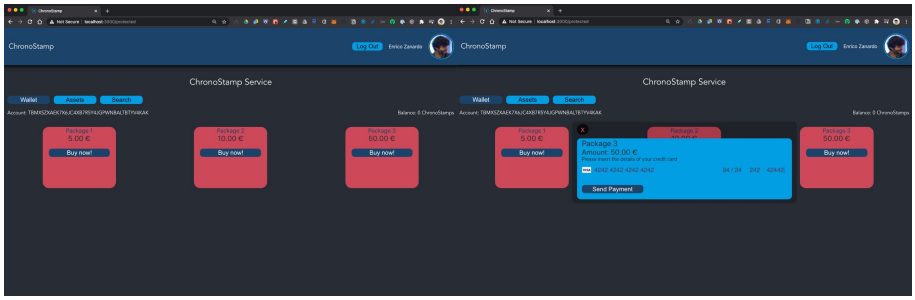
(a) *Login*

(b) *HomePage*



(c) *Wallet*

(d) *Making a real transaction on the NEM blockchain*



(e) *Transaction done.*

(f) *Store an encrypted asset on the IPFS cluster.*

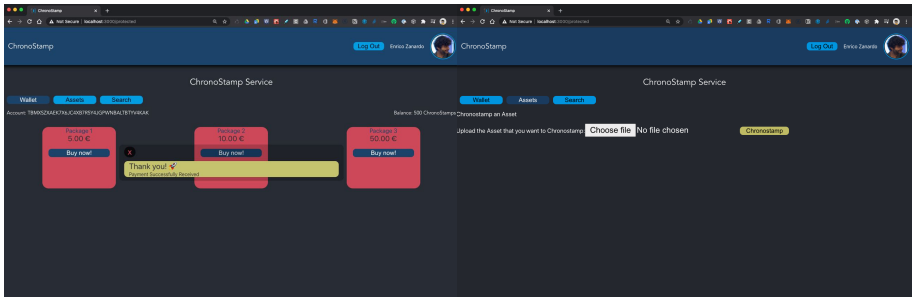


Figure 4.4: *Batch normalization only: the worst model, there is a problem of vanish exploding, and the total rewards does not reach neither the 1% of score.*

(a) *Metrics episode 1*

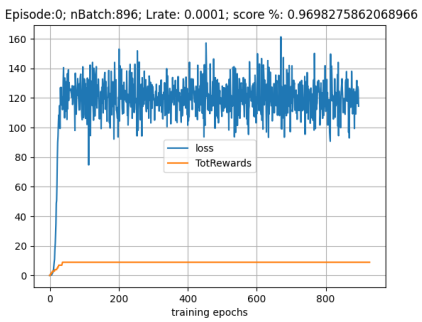
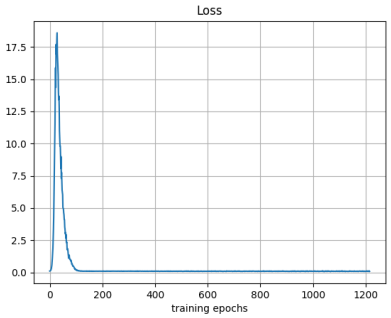
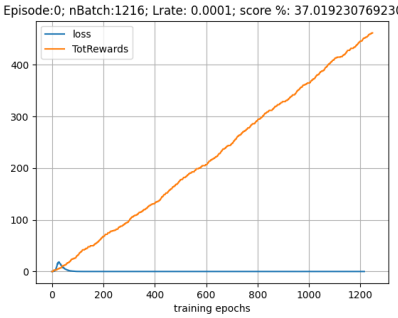


Figure 4.5: *Batch normalization and L2 regularization: The maximum score reached at episode 3 is 47.92%.*

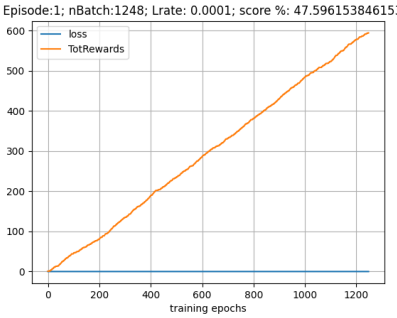
(a) *Loss episode 1*



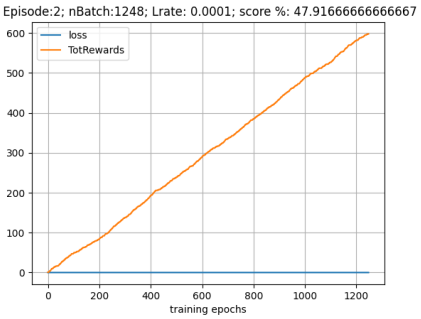
(b) *Metrics episode 1*



(c) *Metrics episode 2*



(d) *Metrics episode 3*



(e) *Total Rewards episode 3*

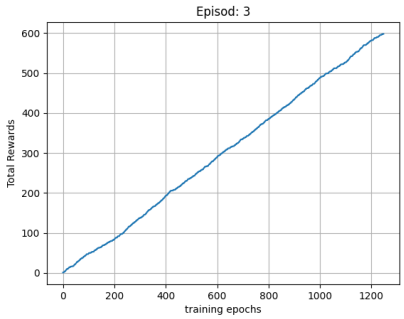
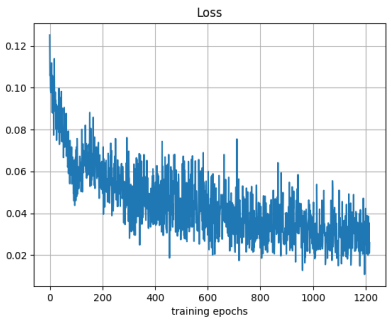
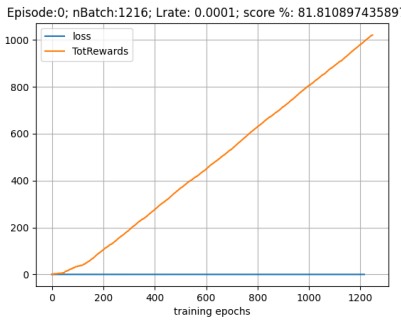


Figure 4.6: *Dropout and L2 regularization: The maximum score reached at episode 3 is*

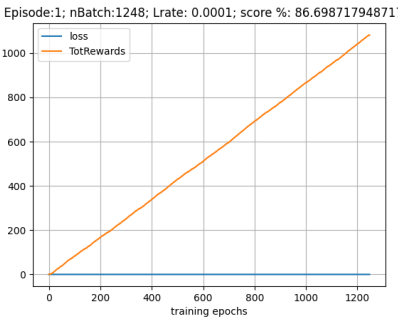
(a) *Loss episode 1*



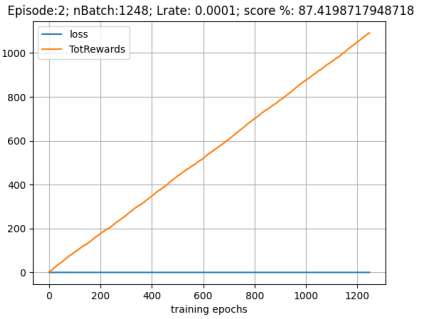
(b) *Metrics episode 1*



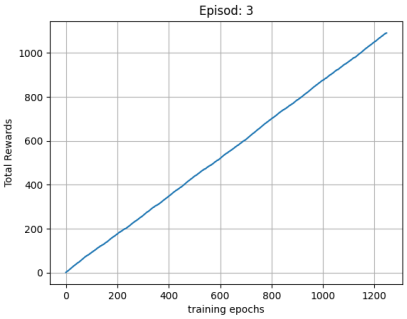
(c) *Metrics episode 2*



(d) *Metrics episode 3*



(e) *Total Rewards episode 3*



Conclusion

The job of this thesis gave the starting point of an ambitious project: developing a *smart automatic learning blockchain* system. The skeleton of the blockchain is defined and a valid *learning model* has been founded. In particular, the DQN model has shown good performance reached a total rewards score more than 87% percentage, with the Dropout and L2 configuration. The nature of machine learning algorithm assures to get always a better model, as soon as new data are available, thus the system is bound to improve with time. The possibility to store the blockchain on a distributed file system as IPFS could allow to increase the distribution of the service even for mobile nodes. All these assumption forms a stable ground on which to work for reaching a *scalable, efficient* and *fairness* blockchain.

Surely, the project will keep to grow up. The first step will be founding the best features configuration for the model, in order to reach a 100% of total rewards score, and switching to a *Main Net* phase. Again, several tests must be done for assuring the stability of the IPFS cluster storage solution. In parallel, the developing of the mobile app will keep to grow by adding other services.

5.1 Acknowledgements

The author is grateful to Dr. Enrico Zanardo, for his support in the entire realization, as well as his fundamental intuition and idea about the nature of the protocol. The author is grateful to the Prof. Tiziano Bianchi for his comments and suggestions during the thesis job. Finally, the author would to like to tell thank you to the Politecnico di Torino, representing by all the teachers of the Master *ICT for Smart Societies*, for the important experience the two years of master has allowed.

Bibliography

[1] Maximum weighted independent set problem - https://en.wikipedia.org/wiki/Independent_set_(graph_theory)

[2] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).

[3] Klobučar, A.; Manger, R. Solving Robust Variants of the Maximum Weighted Independent Set Problem on Trees. Mathematics 2020, 8, 285.

[4] Rennes, ENS. "Study of greedy algorithm for solving Maximum Independent Set problem." (2017).

[5] http://www.secg.org/sec2-v2.pdf

[6] M. Goldberg and T. Spencer, An efficient parallel algorithm that finds independent sets of guaranteed size, SIAM J. Disc. Math., Vol. 6, (1993), pp.443-459.

[7] Hifi, M., & Otmani, N. (2012). An algorithm for the disjunctively constrained knapsack problem. International Journal of Operational Research, 13(1), 22.

[8] https://nemplatform.com

[9] Gilad, Yossi, et al. "Algorand: Scaling byzantine agreements for cryptocurrencies." Proceedings of the 26th Symposium on Operating Systems Principles. 2017.

[10] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[11] ISBN: 9781788834247

[12] Playing Atari with Deep Reinforcement Learning, 1312.5602v1, Mnih and others

[13] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

[14] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." Proceedings of the fourteenth international conference on artificial intelligence and statistics. 2011.

[15] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

[16] Li, Xiang, et al. "Understanding the disharmony between dropout and batch normalization by variance shift." Proceedings of the IEEE conference on computer vision and pattern recognition. 2019.

[17] Cobbe, Karl, et al. "Quantifying generalization in reinforcement learning." International Conference on Machine Learning. PMLR, 2019.

[18] Farebrother, Jesse, Marlos C. Machado, and Michael Bowling. "Generalization and regularization in DQN." arXiv preprint arXiv:1810.00123 (2018).

[19] Zheng, Zibin, et al. "An overview on smart contracts: Challenges, advances and platforms." Future Generation Computer Systems 105 (2020): 475-491.

[20] The Scalability Trilemma in Blockchain, Sep. 2019, [online] Available: https://medium.com/@aakash_13214/the-scalability-trilemma-in-blockchain-75fb57f646df.

[21] Langville, Amy N., and Carl D. Meyer. Google's PageRank and beyond: The science of search engine rankings. Princeton university press, 2011

[22] Androulaki, Elli, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." Proceedings of the thirteenth EuroSys conference. 2018.

[23] K. You, R. Tempo and L. Qiu, "Distributed Algorithms for Computation of Centrality Measures in Complex Networks," in IEEE Transactions on Automatic Control, vol. 62, no. 5, pp. 2080-2094, May 2017, doi: 10.1109/TAC.2016.2604373.

[24] Wüst, Karl, and Arthur Gervais. "Do you need a blockchain?." 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). IEEE, 2018.

[25] David, Bernardo, et al. "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Cham, 2018.

[26] Kiayias, Aggelos, et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol." Annual International Cryptology Conference. Springer, Cham, 2017.

[27] Sherman, Alan T., et al. "On the origins and variations of blockchain technologies." IEEE Security & Privacy 17.1 (2019): 72-77.

[28] Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system. Manubot, 2019.

[29] Garay, Juan, Aggelos Kiayias, and Nikos Leonardos. "The bitcoin backbone protocol: Analysis and applications." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2015.

[30] Bano, Shehar, et al. "Consensus in the age of blockchains." arXiv preprint arXiv:1711.03936 (2017).

[31] Xiao, Yang, et al. "A survey of distributed consensus protocols for blockchain networks." IEEE Communications Surveys & Tutorials 22.2 (2020): 1432-1465.

[32] Lamport, Leslie, Robert Shostak, and Marshall Pease. "The Byzantine generals problem." Concurrency: the Works of Leslie Lamport. 2019. 203-226.

[33] Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance." OSDI. Vol. 99. No. 1999. 1999.

[34] Conoscenti, Marco, Antonio Vetro, and Juan Carlos De Martin. "Blockchain for the Internet of Things: A systematic literature review." 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA). IEEE, 2016.

[35] Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system." Concurrency: the Works of Leslie Lamport. 2019. 179-196.

[36] Schneider, Fred B. "Implementing fault-tolerant services using the state machine approach: A tutorial." ACM Computing Surveys (CSUR) 22.4 (1990): 299-319.

[37] Pease, Marshall, Robert Shostak, and Leslie Lamport. "Reaching agreement in the presence of faults." Journal of the ACM (JACM) 27.2 (1980): 228-234.

[38] Fischer, Michael J., Nancy A. Lynch, and Michael Merritt. "Easy impossibility proofs for distributed consensus problems." Distributed Computing 1.1 (1986): 26-39.

[39] Decker, Christian, and Roger Wattenhofer. "Information propagation in the bitcoin network." IEEE P2P 2013 Proceedings. IEEE, 2013.

[40] Cachin, Christian, and Marko Vukolić. "Blockchain consensus protocols in the wild." arXiv preprint arXiv:1707.01873 (2017).

[41] Attiya, Hagit, Amotz Bar-Noy, and Danny Dolev. "Sharing memory robustly in message-passing systems." Journal of the ACM (JACM) 42.1 (1995): 124-142.

[42] Eisenbarth, Thomas, et al. "A survey of lightweight-cryptography implementations." IEEE Design & Test of Computers 24.6 (2007): 522-533.

[43] Zheng, Zibin, et al. "Blockchain challenges and opportunities: A survey." International Journal of Web and Grid Services 14.4 (2018): 352-375.

[44] Xiao, Yang, et al. "Distributed consensus protocols and algorithms." Blockchain for Distributed Systems Security 25 (2019).

[45] https://ipfs.io

[46] Attiya, Hagit, and Jennifer Welch. Distributed computing: fundamentals, simulations, and advanced topics. Vol. 19. John Wiley & Sons, 2004.

[47] Dorri, Ali, et al. "Blockchain: A distributed solution to automotive security and privacy." IEEE Communications Magazine 55.12 (2017): 119-125.

[48] Andoni, Merlinda, et al. "Blockchain technology in the energy sector: A systematic review of challenges and opportunities." Renewable and Sustainable Energy Reviews 100 (2019): 143-174.

[49] Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. "Impossibility of distributed consensus with one faulty process." Journal of the ACM (JACM) 32.2 (1985): 374-382.

[50] Micali, Silvio, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In Proceedings of the 40th Annual Symposium on the Foundations of Computer Science (FOCS '99), 120-130. New York: IEEE Computer Society Press.

[51] Al-Rubaie, Mohammad, and J. Morris Chang. "Privacy-preserving machine learning: Threats and solutions." IEEE Security & Privacy 17.2 (2019): 49-58.

[52] Bentov, Iddo, Rafael Pass, and Elaine Shi. "Snow White: Provably Secure Proofs of Stake." IACR Cryptol. ePrint Arch. 2016 (2016): 919.

[53] Yang, Qiang, et al. "Federated machine learning: Concept and applications." ACM Transactions on Intelligent Systems and Technology (TIST) 10.2 (2019): 1-19.

[54] Kairouz, Peter, et al. "Advances and open problems in federated learning." arXiv preprint arXiv:1912.04977 (2019).

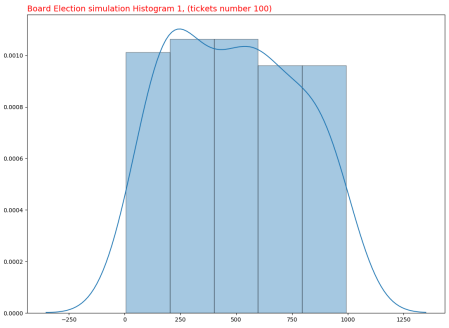
[55] Zheng, Zibin, et al. "An overview of blockchain technology: Architecture, consensus, and future trends." 2017 IEEE international congress on big data (BigData congress). IEEE, 2017.

5.2 Appendix

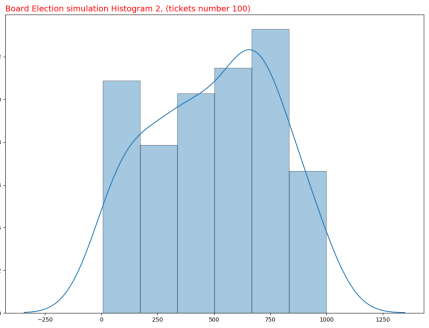
5.2.1 Lottery Analysis

Figure 5.1: *Simulation 1: 100 tickets extracted.*

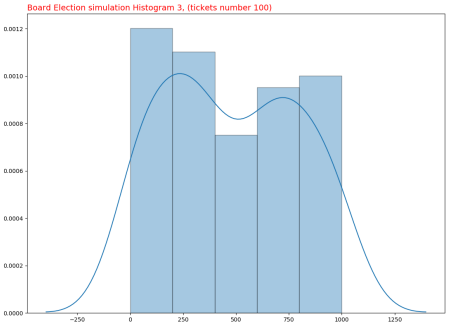
(a) *Experiment 1*



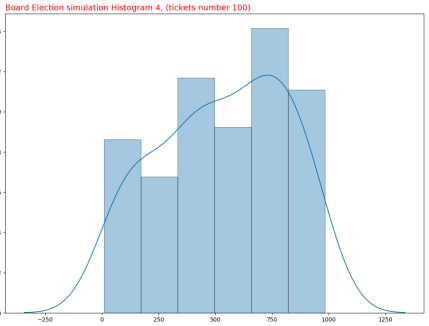
(b) *Experiment 2*



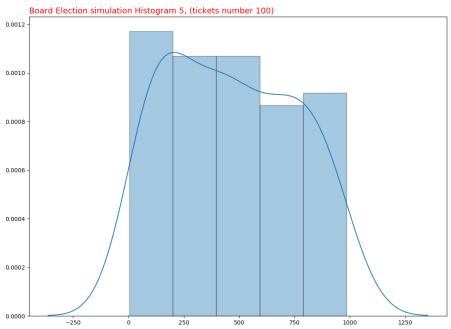
(c) *Experiment 3*



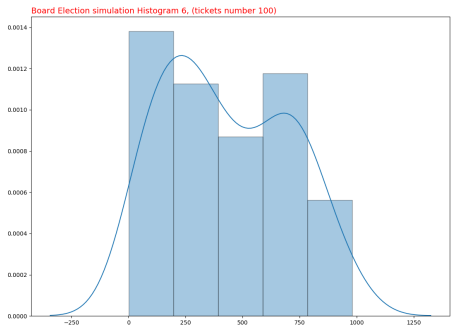
(d) *Experiment 4*



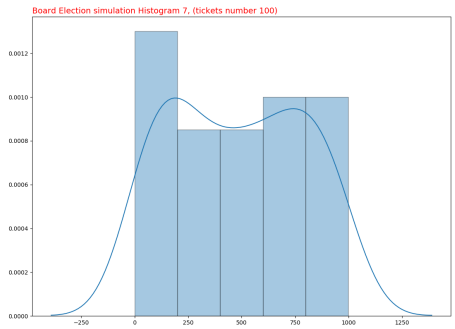
(e) *Experiment 5*



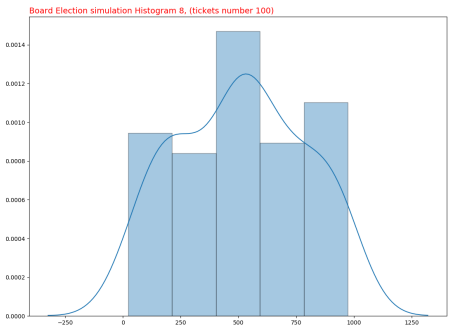
(f) *Experiment 6*



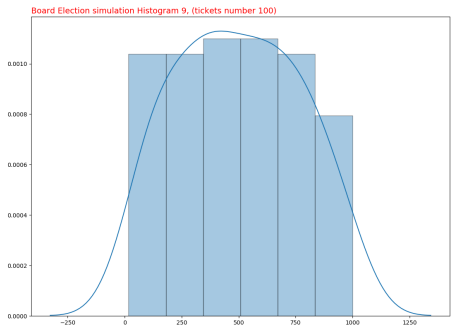
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

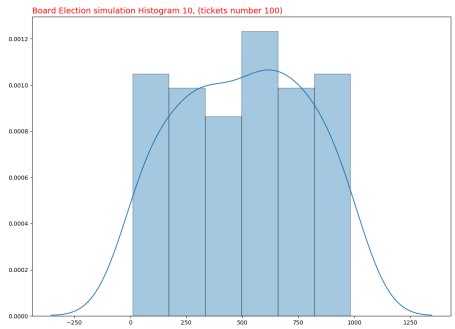
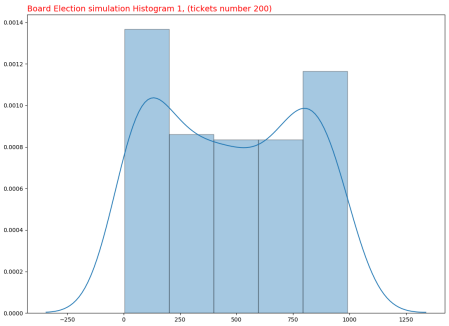
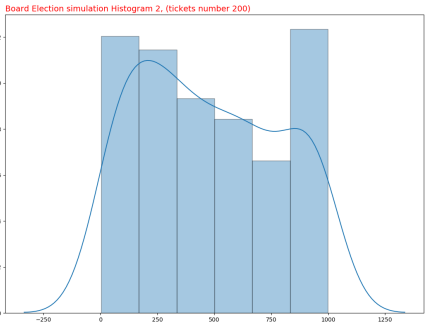


Figure 5.2: *Simulation 2: 200 tickets extracted.*

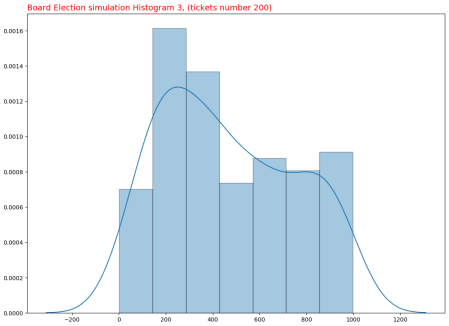
(a) *Experiment 1*



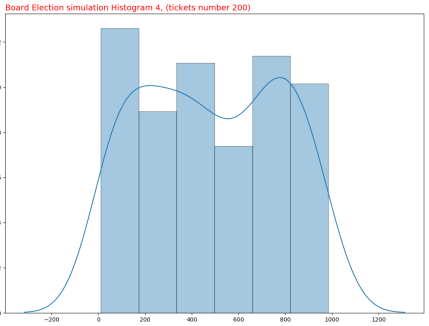
(b) *Experiment 2*



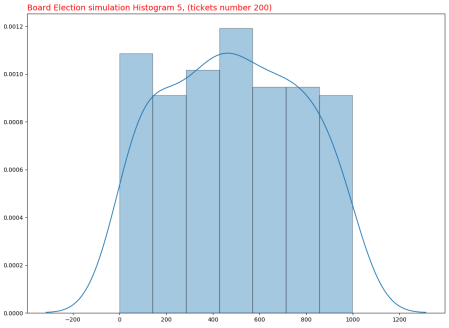
(c) *Experiment 3*



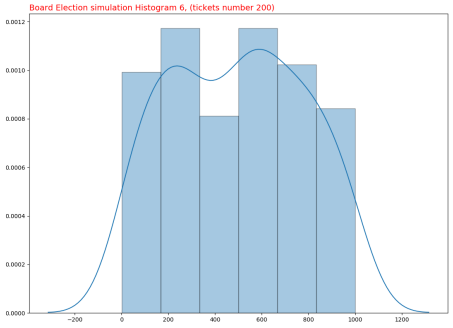
(d) *Experiment 4*



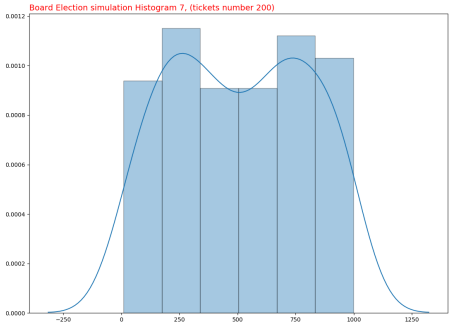
(e) *Experiment 5*



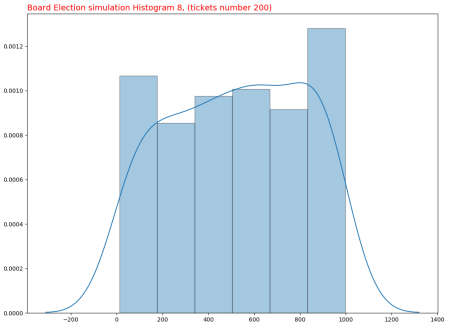
(f) *Experiment 6*



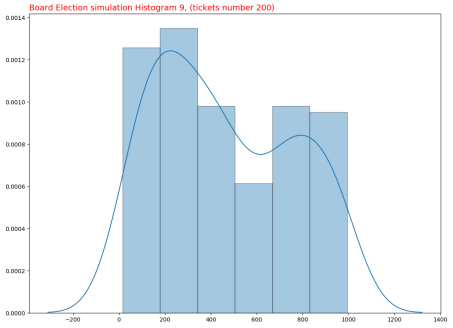
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

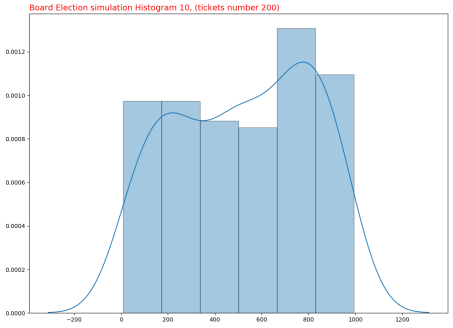
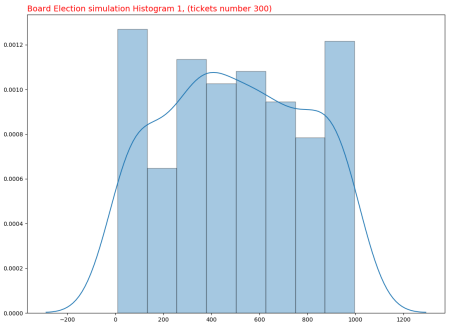
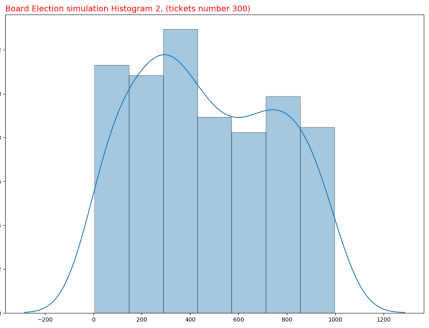


Figure 5.3: *Simulation 3: 300 tickets extracted.*

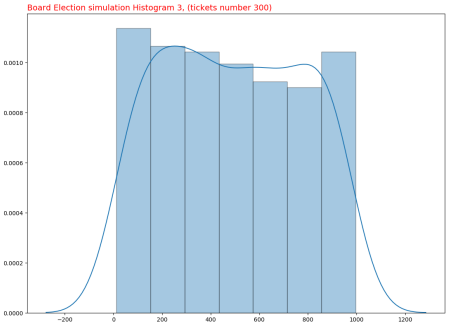
(a) *Experiment 1*



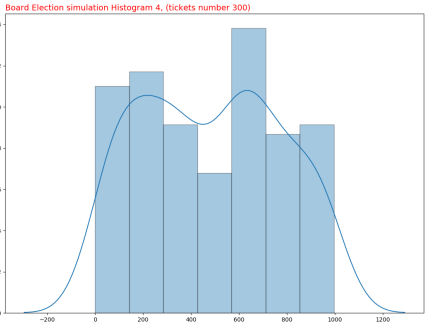
(b) *Experiment 2*



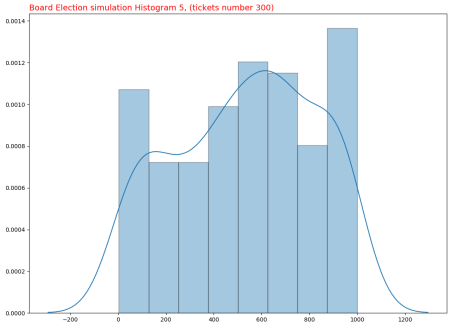
(c) *Experiment 3*



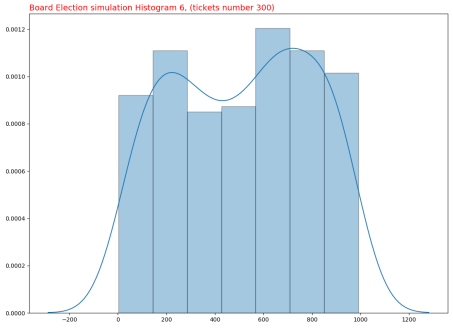
(d) *Experiment 4*



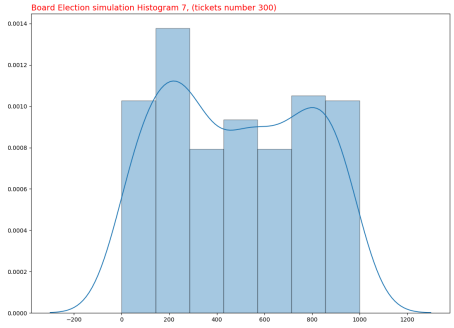
(e) *Experiment 5*



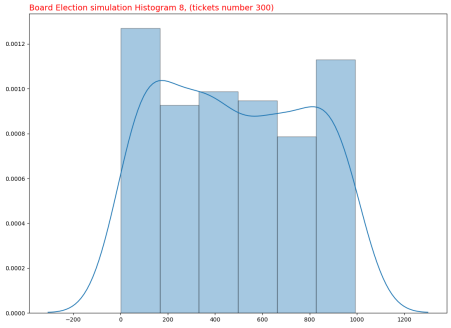
(f) *Experiment 6*



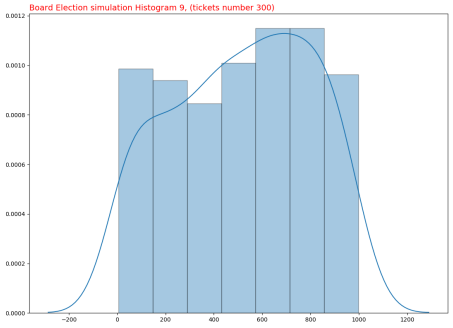
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

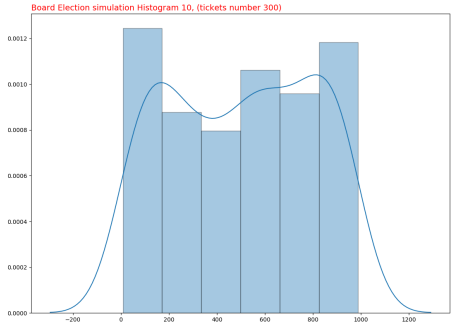
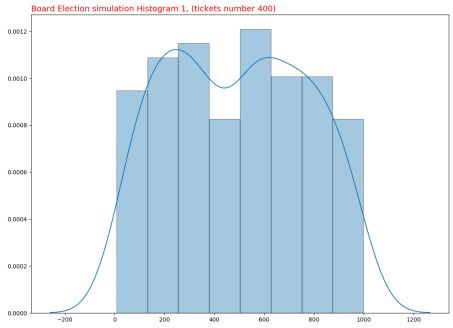
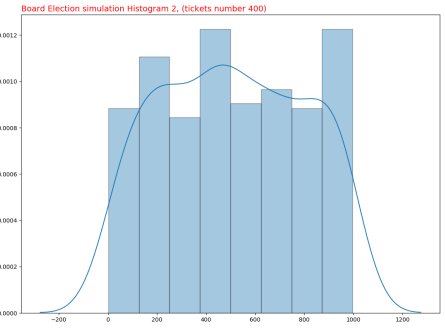


Figure 5.4: *Simulation 4: 400 tickets extracted.*

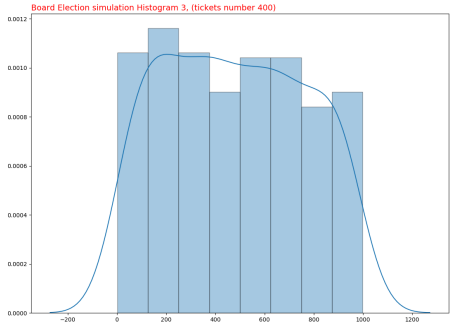
(a) *Experiment 1*



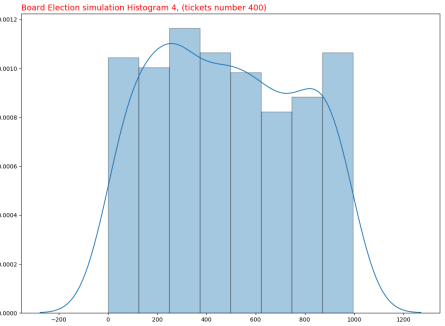
(b) *Experiment 2*



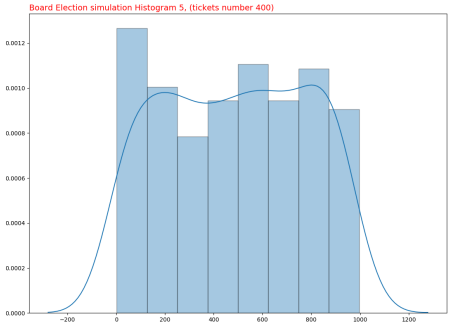
(c) *Experiment 3*



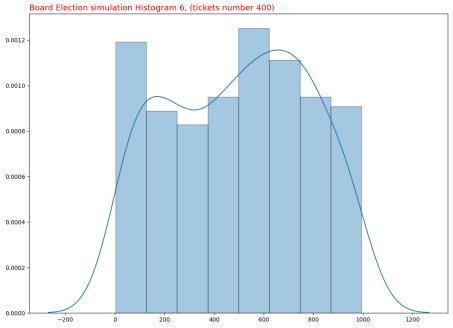
(d) *Experiment 4*



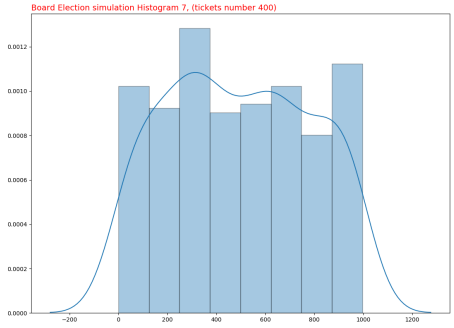
(e) *Experiment 5*



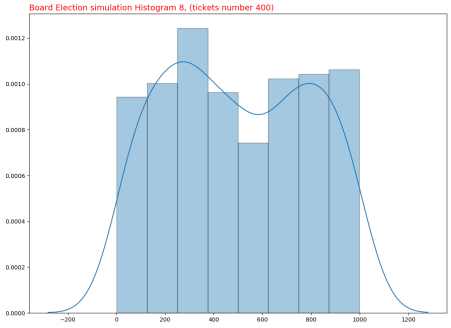
(f) *Experiment 6*



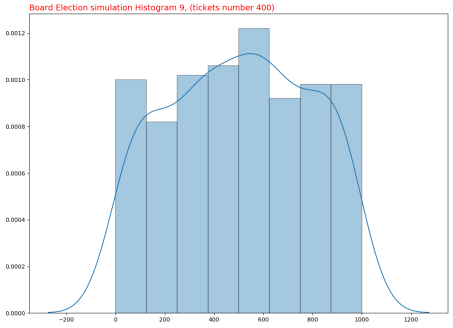
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

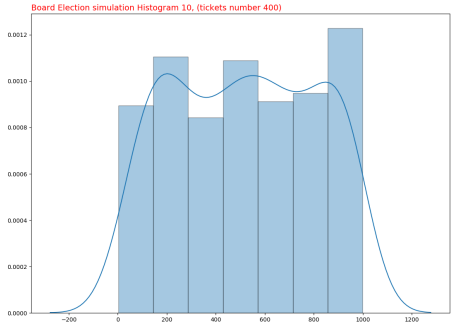
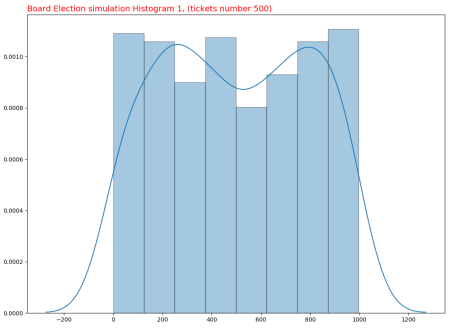
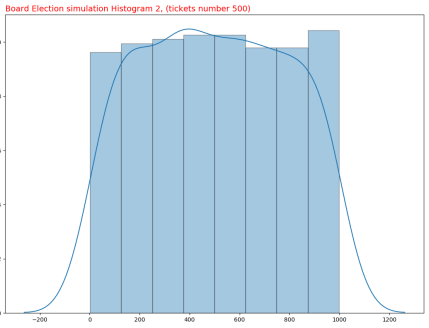


Figure 5.5: *Simulation 5: 500 tickets extracted.*

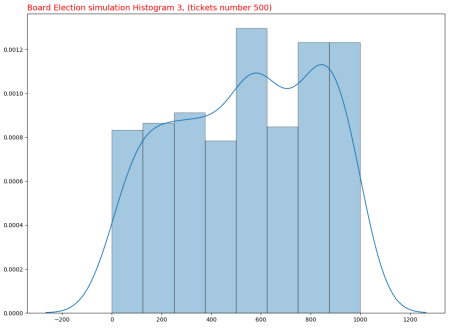
(a) *Experiment 1*



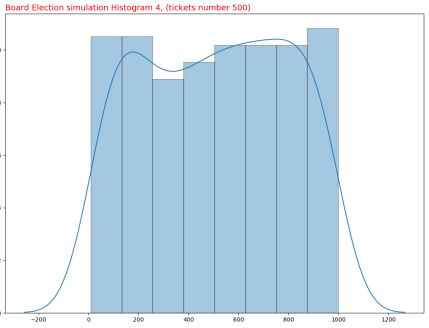
(b) *Experiment 2*



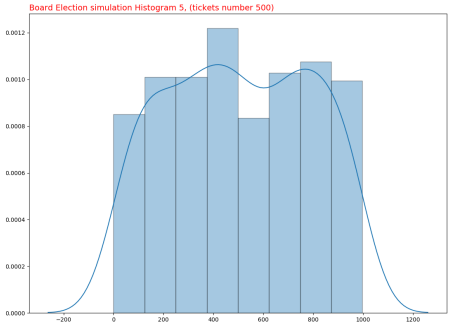
(c) *Experiment 3*



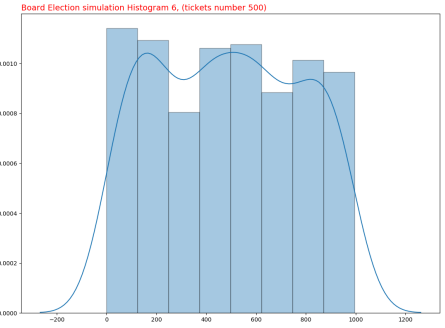
(d) *Experiment 4*



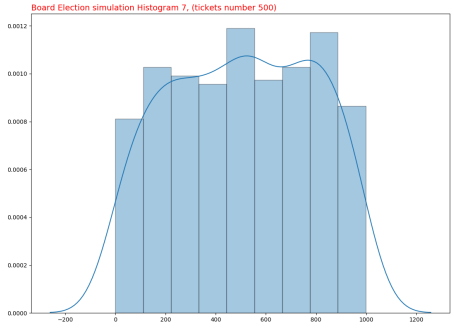
(e) *Experiment 5*



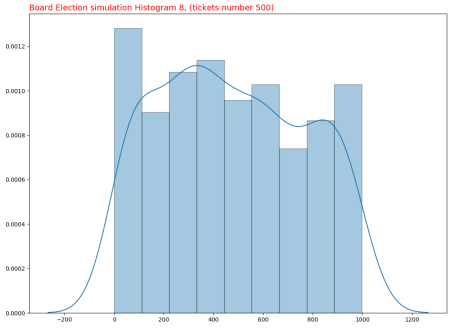
(f) *Experiment 6*



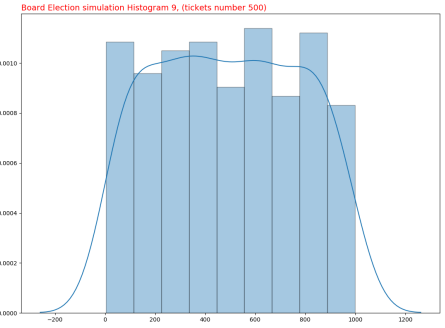
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

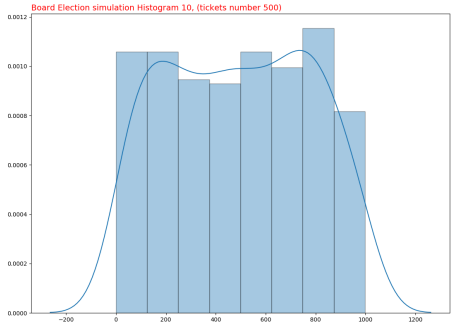
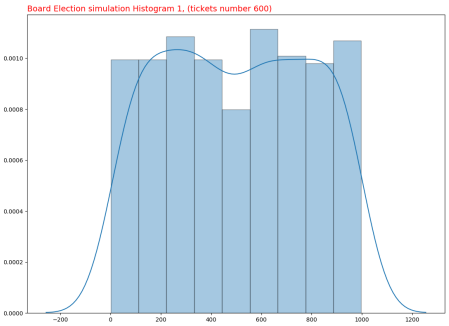
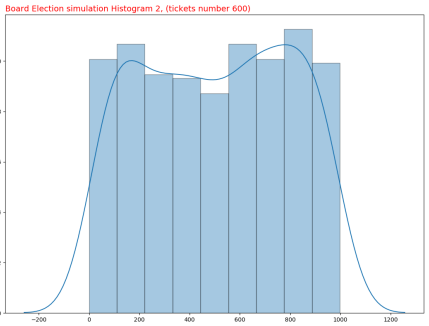


Figure 5.6: *Simulation 6: 600 tickets extracted.*

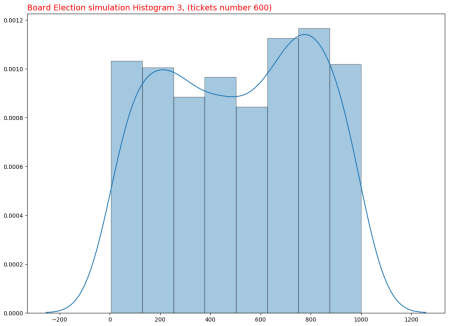
(a) *Experiment 1*



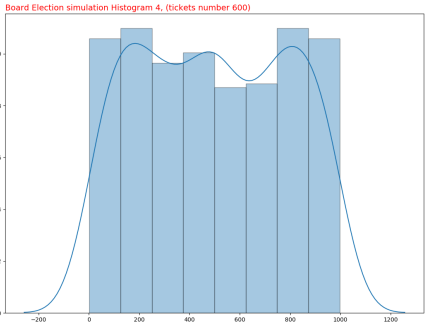
(b) *Experiment 2*



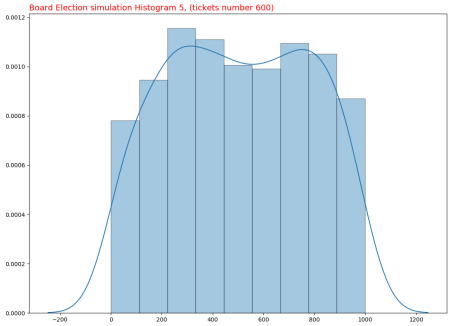
(c) *Experiment 3*



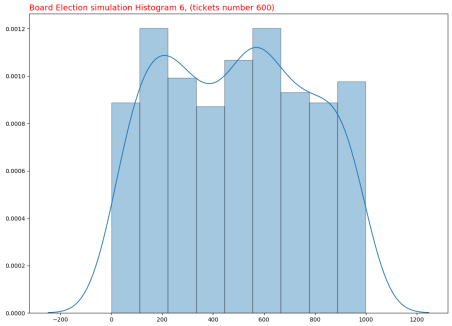
(d) *Experiment 4*



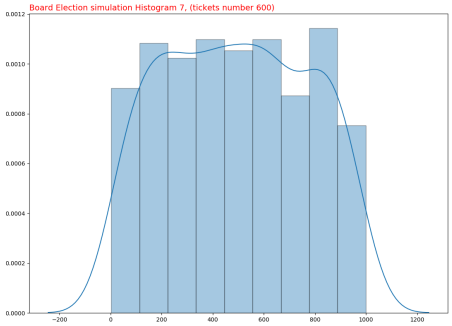
(e) *Experiment 5*



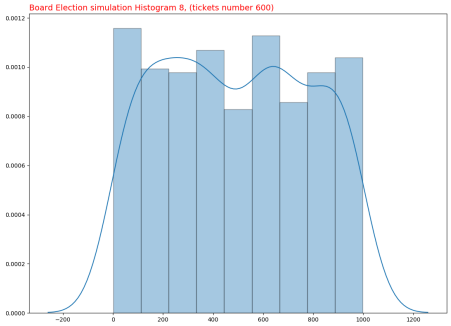
(f) *Experiment 6*



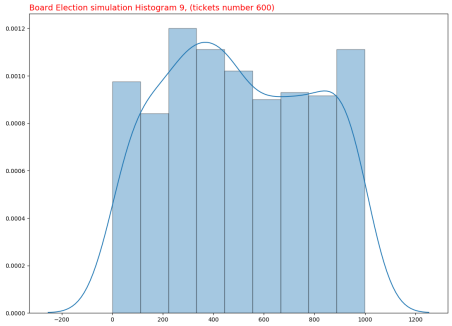
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

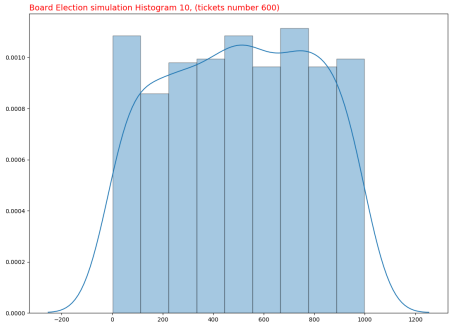
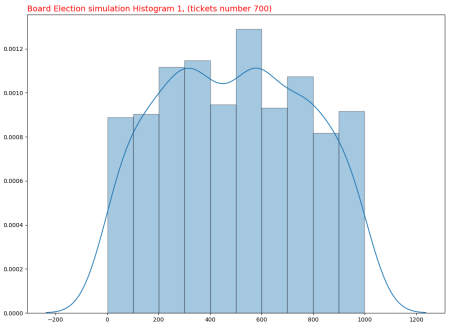
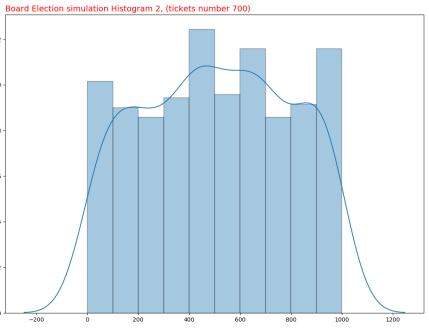


Figure 5.7: *Simulation 7: 700 tickets extracted.*

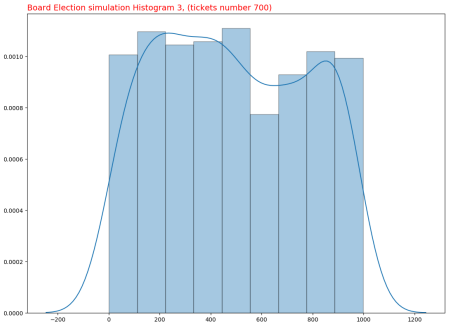
(a) *Experiment 1*



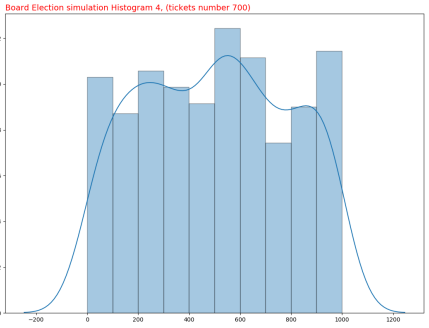
(b) *Experiment 2*



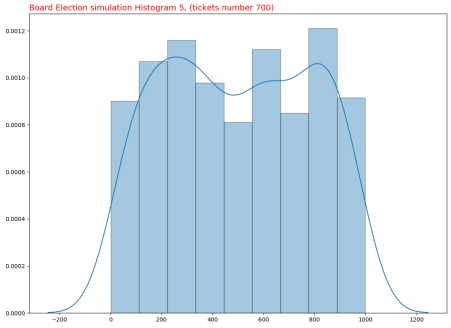
(c) *Experiment 3*



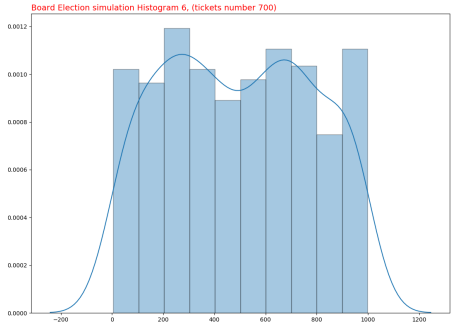
(d) *Experiment 4*



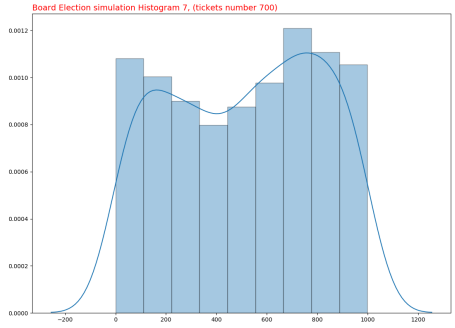
(e) *Experiment 5*



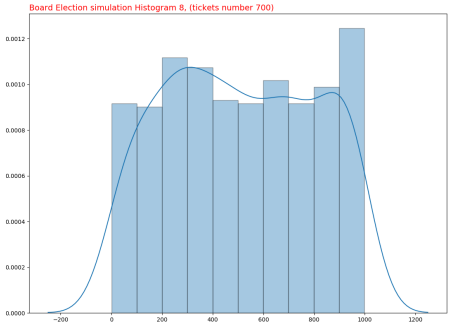
(f) *Experiment 6*



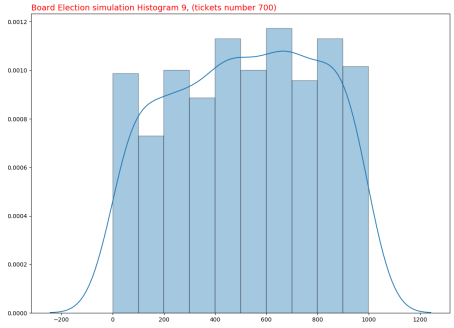
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

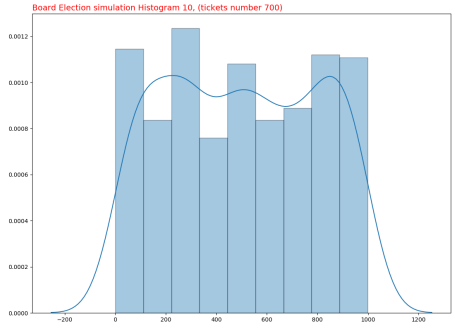
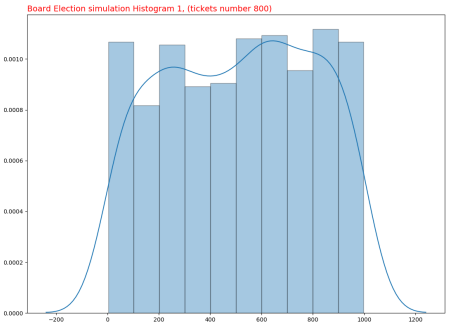
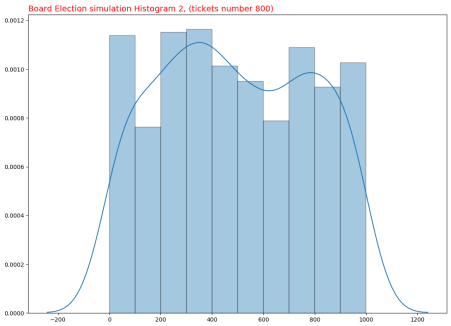


Figure 5.8: *Simulation 8: 800 tickets extracted.*

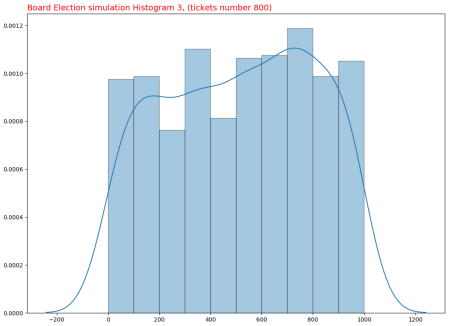
(a) *Experiment 1*



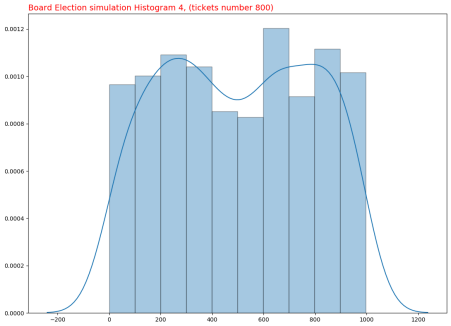
(b) *Experiment 2*



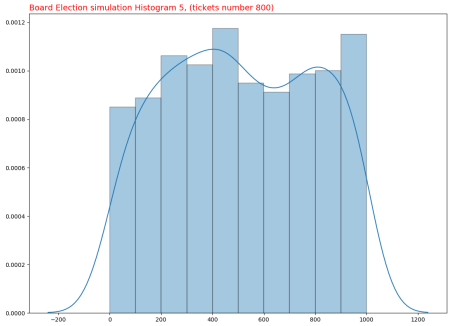
(c) *Experiment 3*



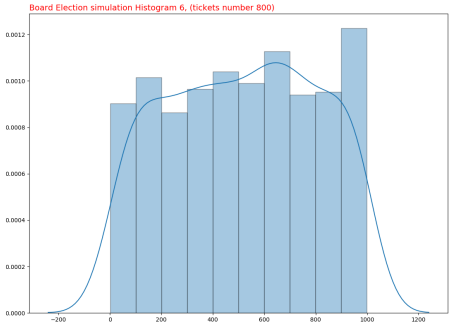
(d) *Experiment 4*



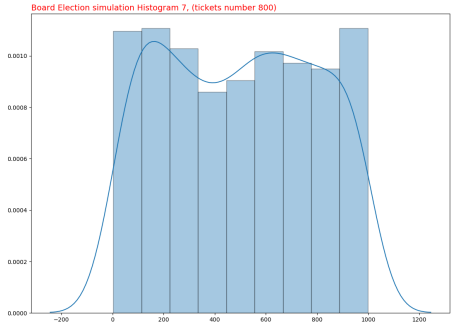
(e) *Experiment 5*



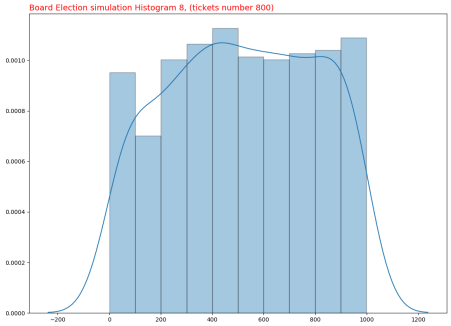
(f) *Experiment 6*



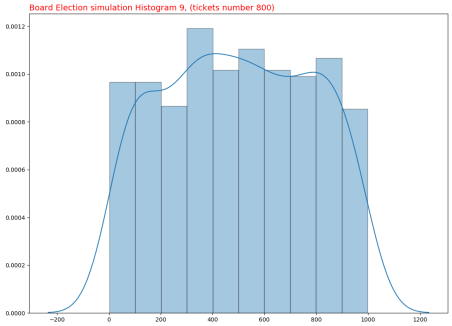
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

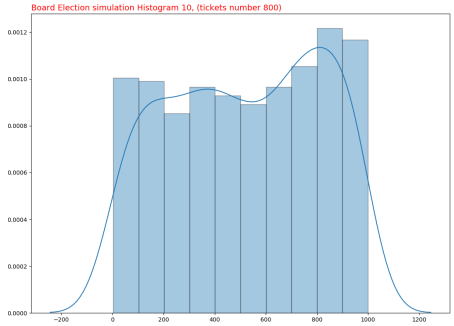
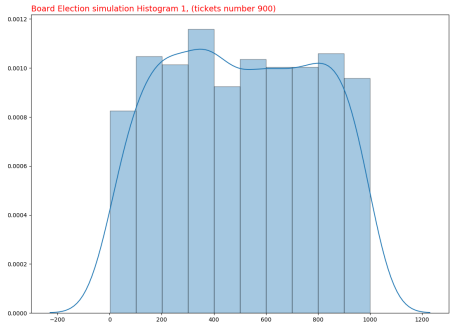
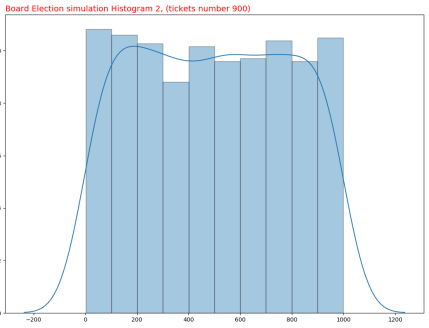


Figure 5.9: *Simulation 9: 900 tickets extracted.*

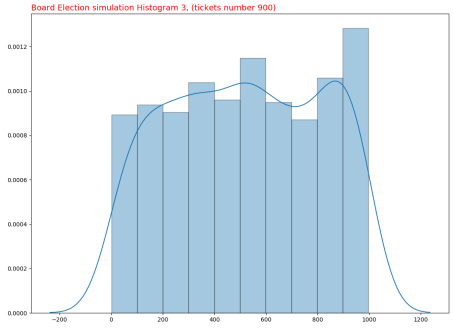
(a) *Experiment 1*



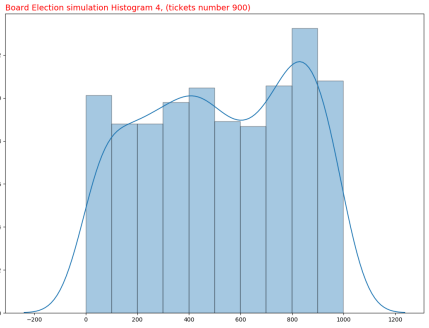
(b) *Experiment 2*



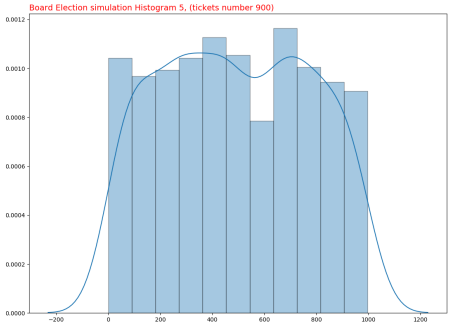
(c) *Experiment 3*



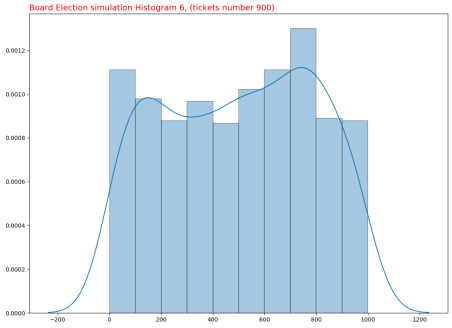
(d) *Experiment 4*



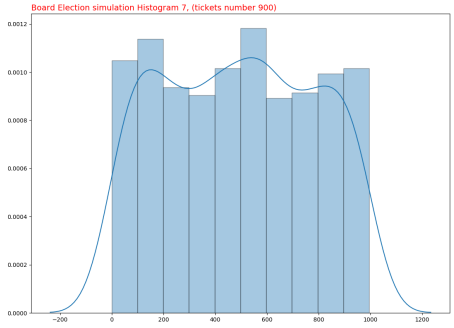
(e) *Experiment 5*



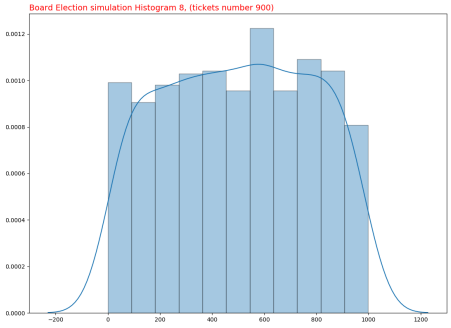
(f) *Experiment 6*



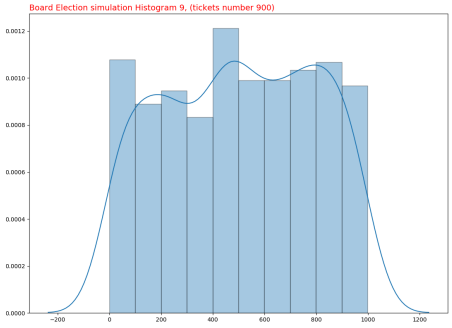
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

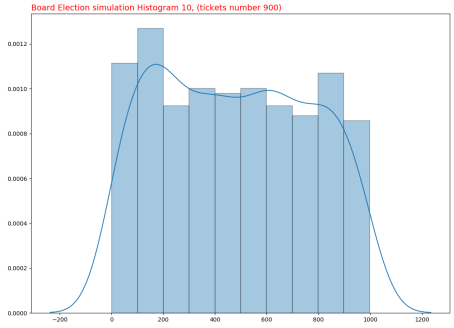
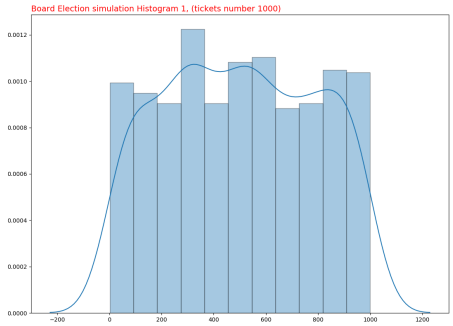
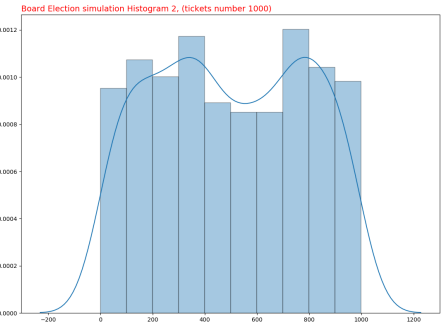


Figure 5.10: *Simulation 10: 1000 tickets extracted.*

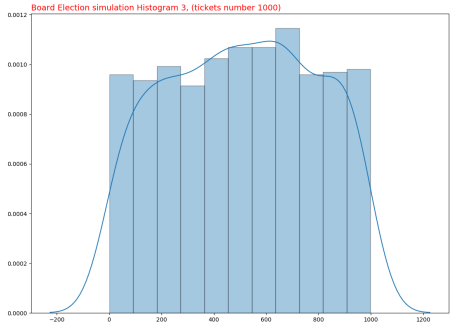
(a) *Experiment 1*



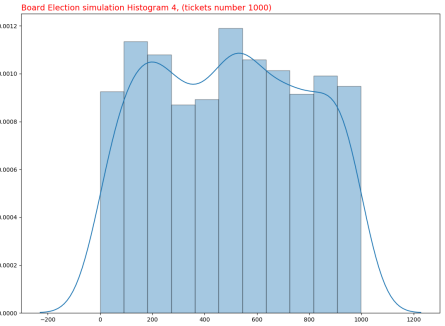
(b) *Experiment 2*



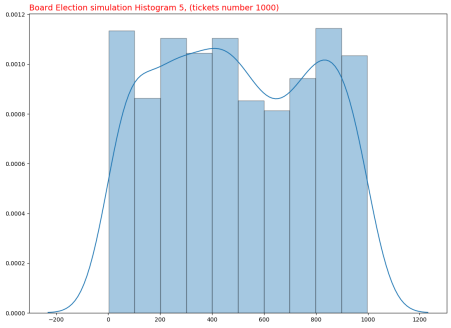
(c) *Experiment 3*



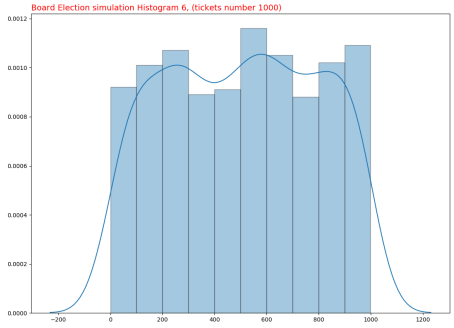
(d) *Experiment 4*



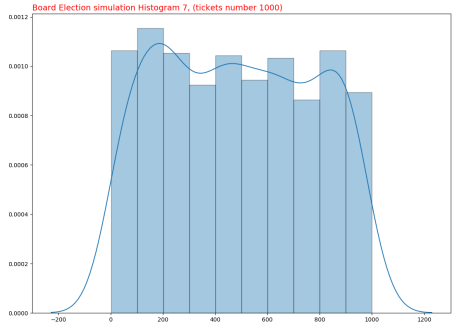
(e) *Experiment 5*



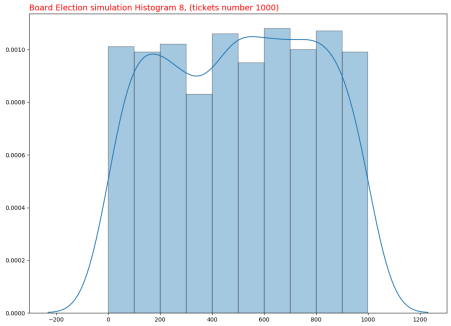
(f) *Experiment 6*



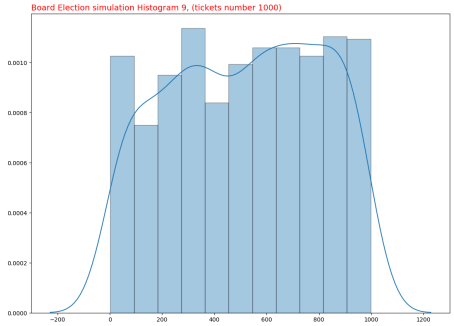
(g) *Experiment 7*



(h) *Experiment 8*



(i) *Experiment 9*



(j) *Experiment 10*

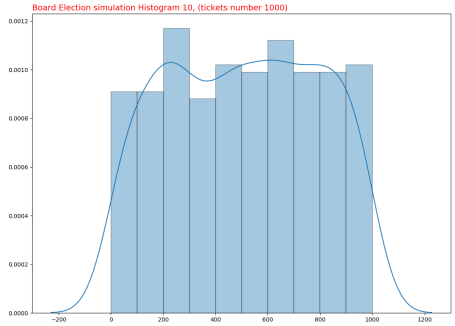
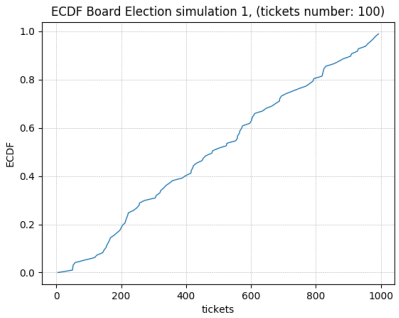
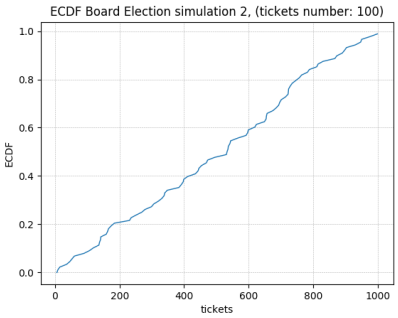


Figure 5.11: *ECDF Simulation 1: 100 tickets extracted.*

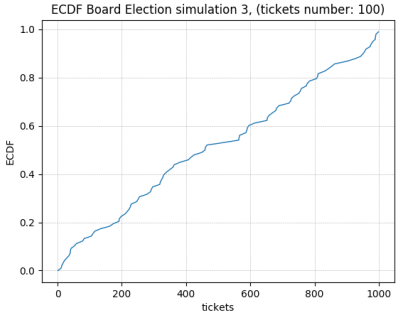
(a) *ECDF Experiment 1*



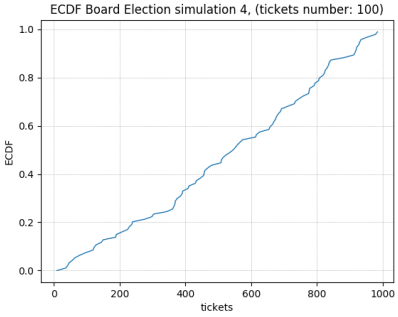
(b) *ECDF Experiment 2*



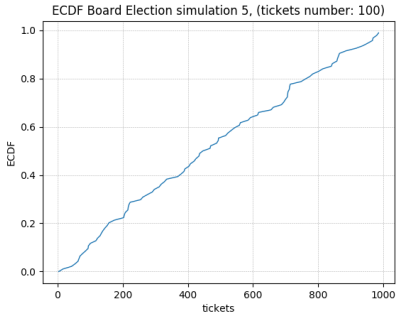
(c) *ECDF Experiment 3*



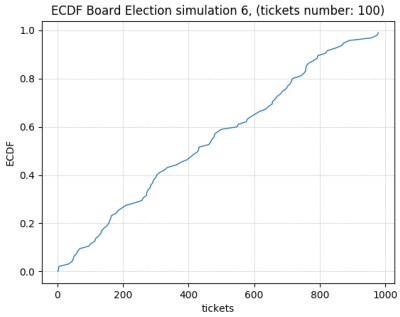
(d) *ECDF Experiment 4*



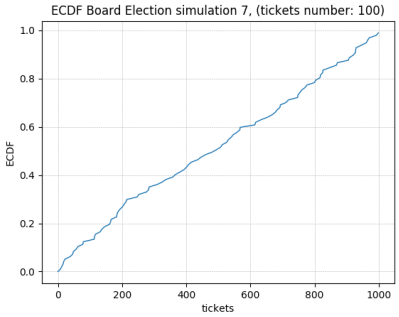
(e) *ECDF Experiment 5*



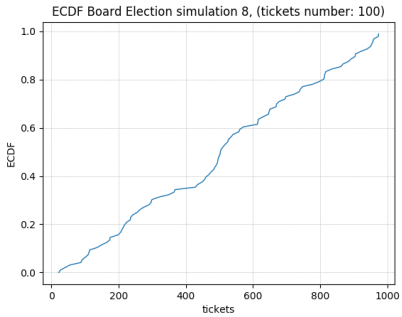
(f) *ECDF Experiment 6*



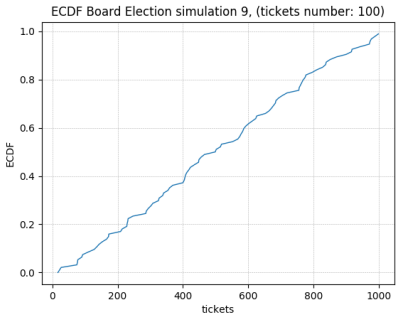
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

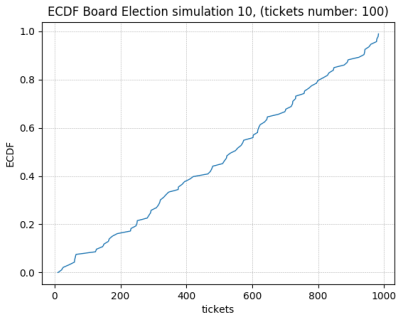
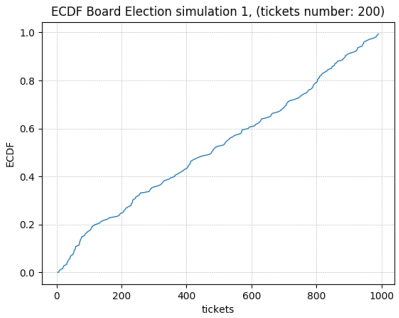
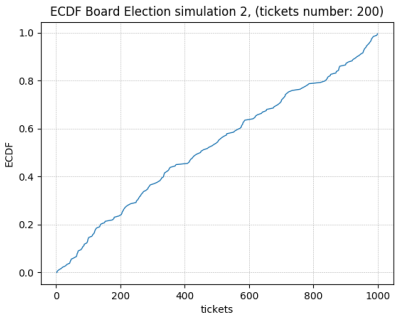


Figure 5.12: *ECDF Simulation 2: 200 tickets extracted.*

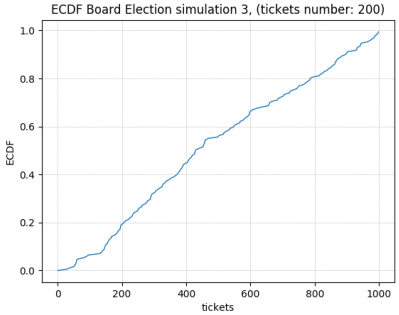
(a) *ECDF Experiment 1*



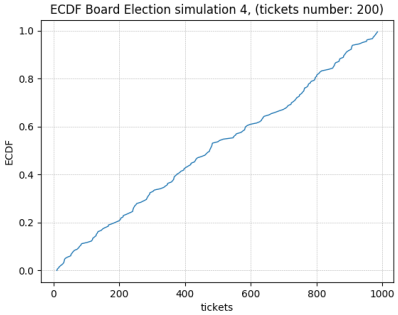
(b) *ECDF Experiment 2*



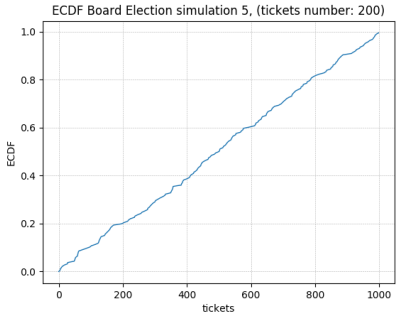
(c) *ECDF Experiment 3*



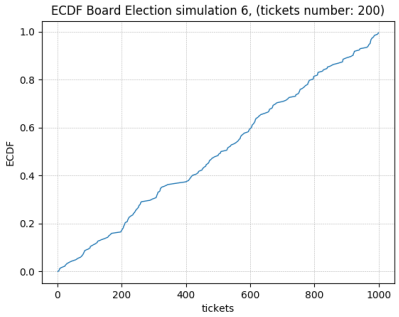
(d) *ECDF Experiment 4*



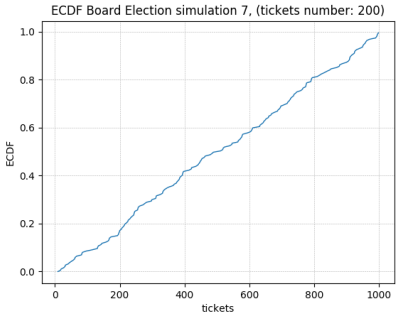
(e) *ECDF Experiment 5*



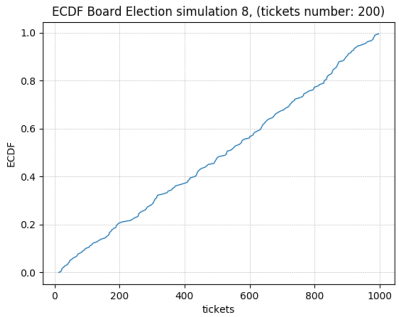
(f) *ECDF Experiment 6*



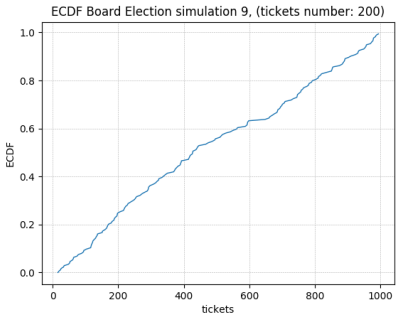
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

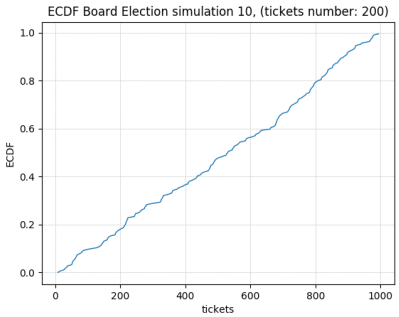
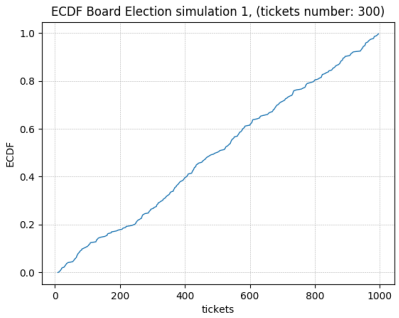
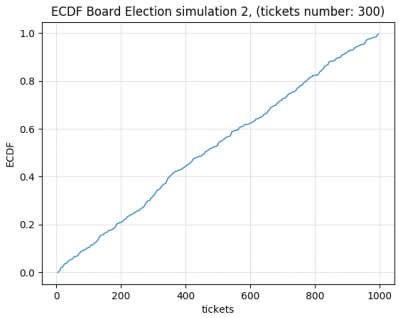


Figure 5.13: *ECDF Simulation 3: 300 tickets extracted.*

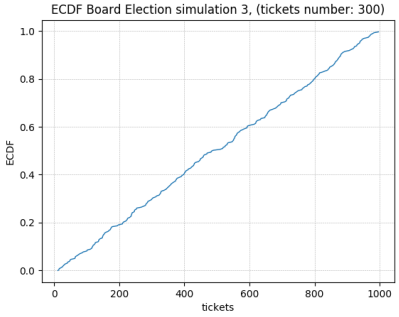
(a) *ECDF Experiment 1*



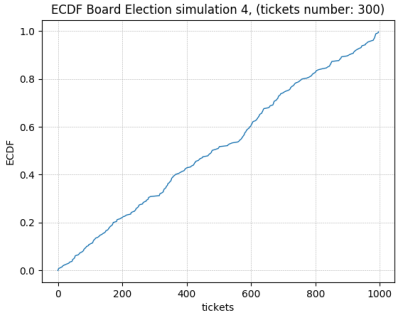
(b) *ECDF Experiment 2*



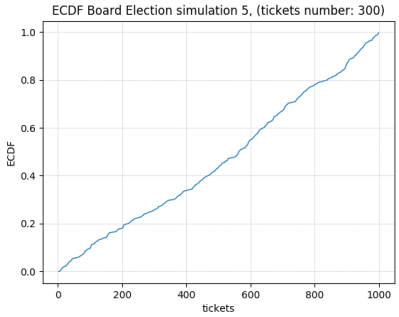
(c) *ECDF Experiment 3*



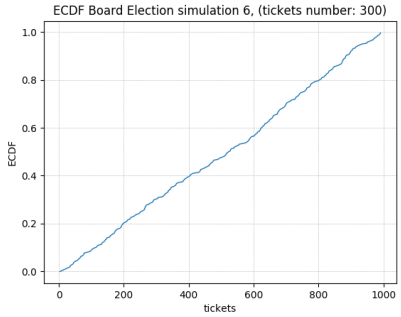
(d) *ECDF Experiment 4*



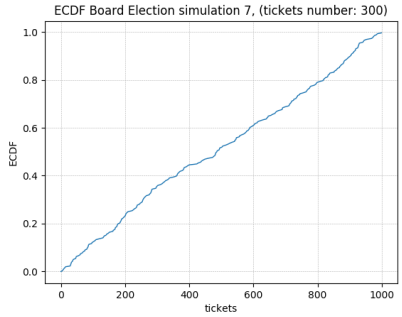
(e) *ECDF Experiment 5*



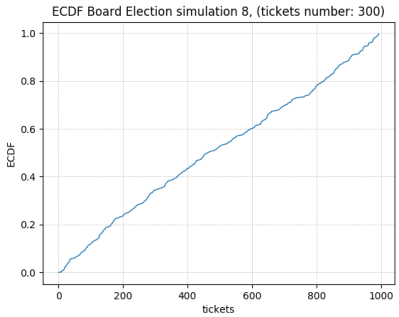
(f) *ECDF Experiment 6*



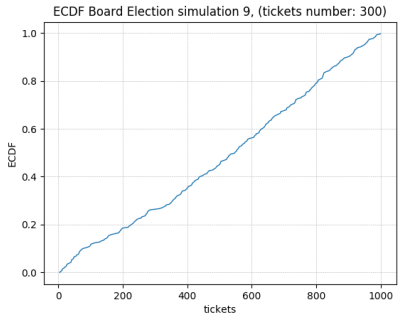
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

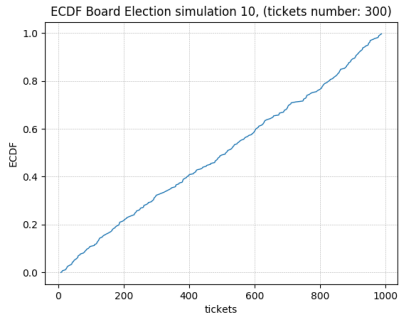
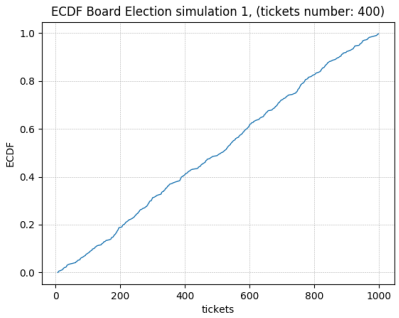
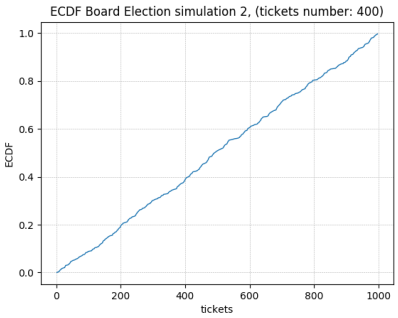


Figure 5.14: *ECDF Simulation 4: 400 tickets extracted.*

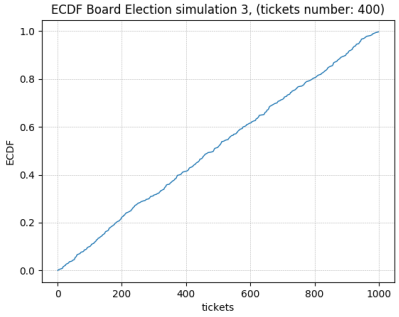
(a) *ECDF Experiment 1*



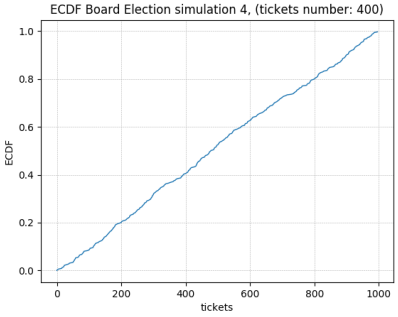
(b) *ECDF Experiment 2*



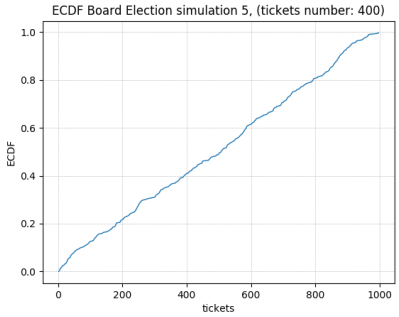
(c) *ECDF Experiment 3*



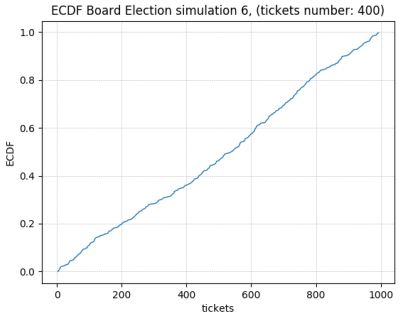
(d) *ECDF Experiment 4*



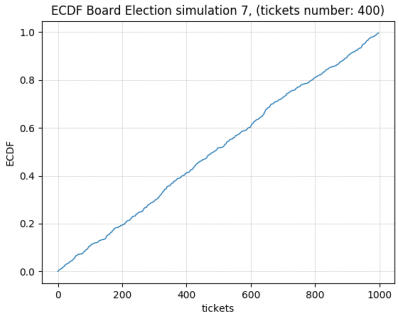
(e) *ECDF Experiment 5*



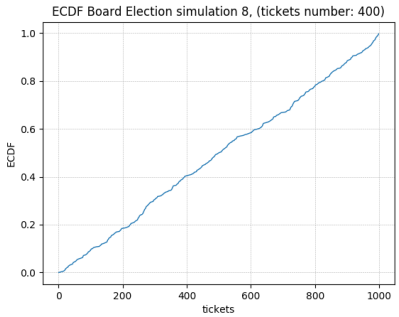
(f) *ECDF Experiment 6*



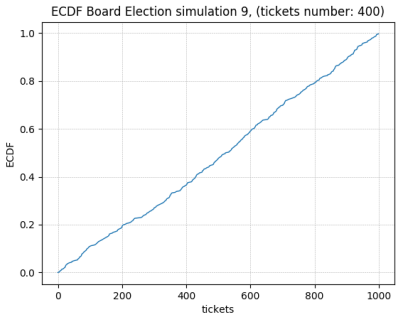
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

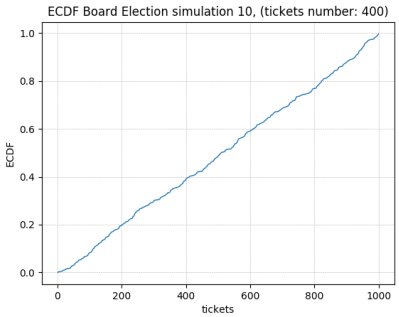
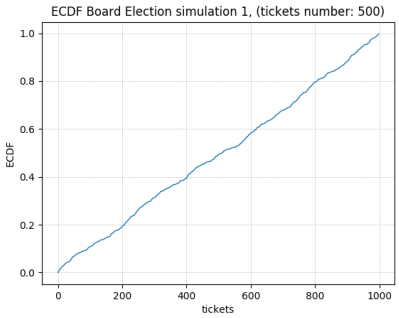
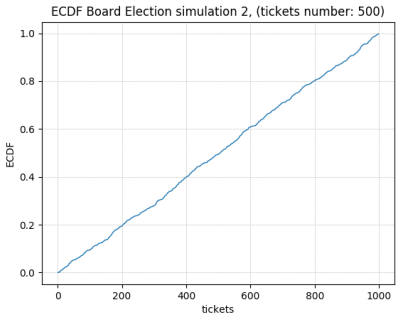


Figure 5.15: *ECDF Simulation 5: 500 tickets extracted.*

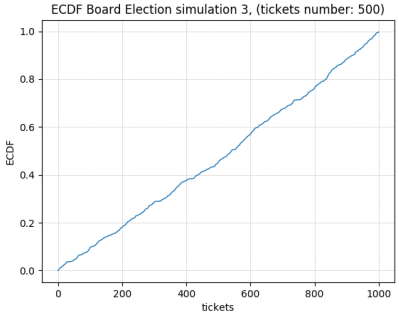
(a) *ECDF Experiment 1*



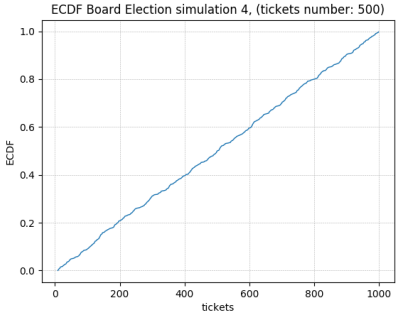
(b) *ECDF Experiment 2*



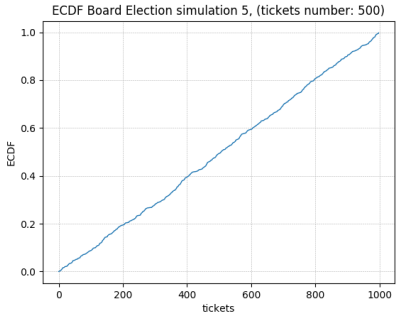
(c) *ECDF Experiment 3*



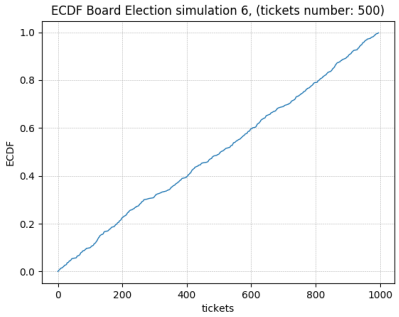
(d) *ECDF Experiment 4*



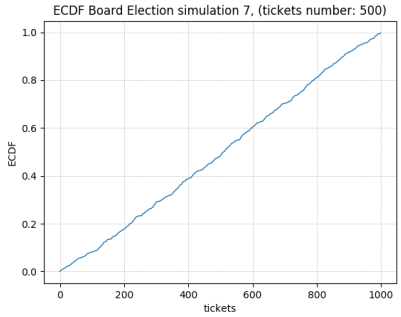
(e) *ECDF Experiment 5*



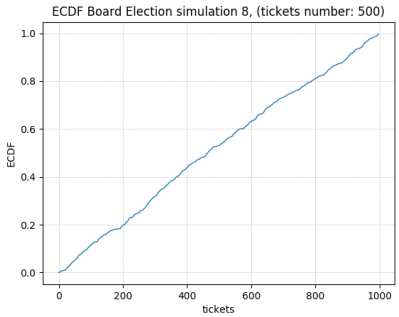
(f) *ECDF Experiment 6*



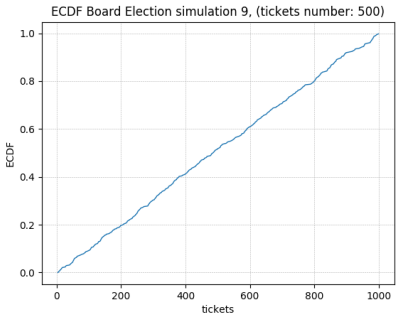
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

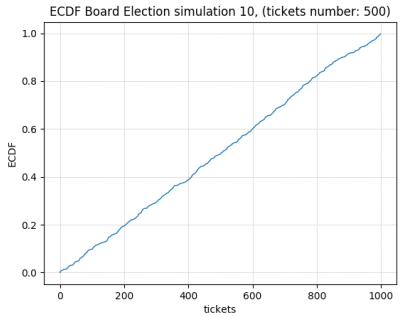
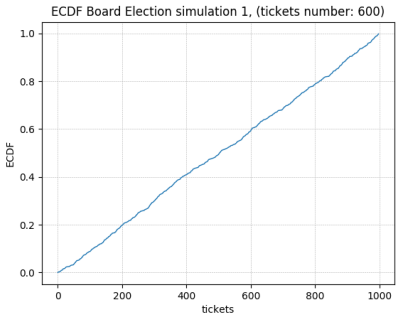
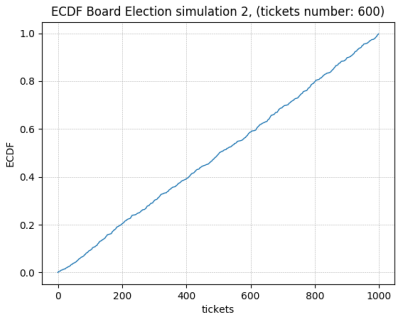


Figure 5.16: *ECDF Simulation 6: 600 tickets extracted.*

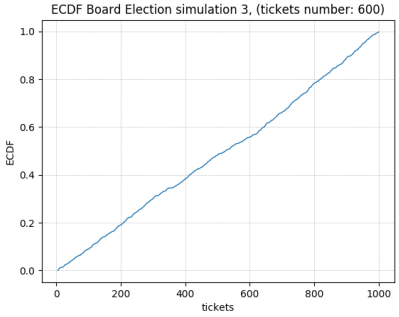
(a) *ECDF Experiment 1*



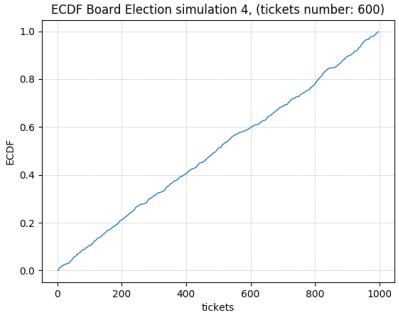
(b) *ECDF Experiment 2*



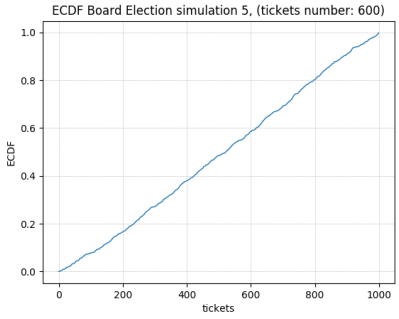
(c) *ECDF Experiment 3*



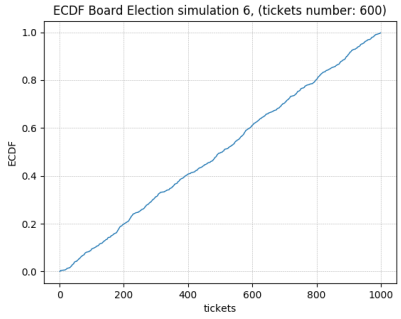
(d) *ECDF Experiment 4*



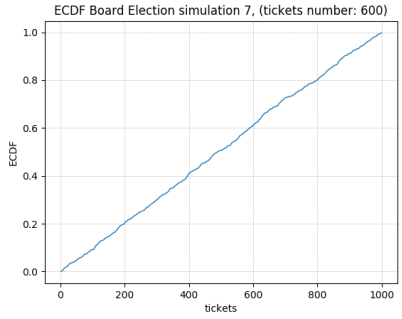
(e) *ECDF Experiment 5*



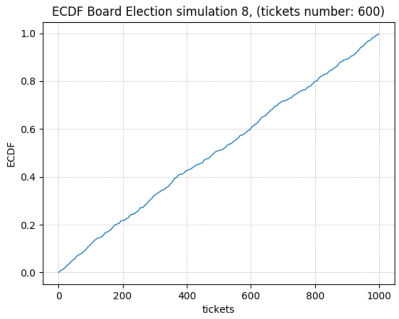
(f) *ECDF Experiment 6*



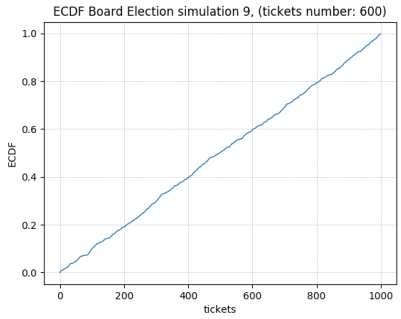
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

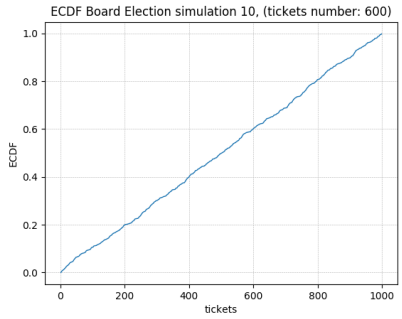
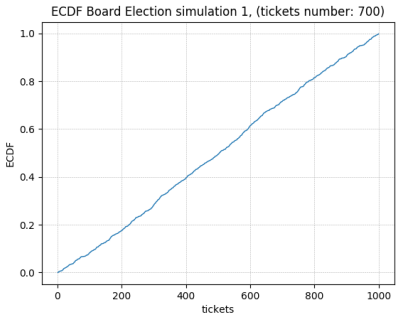
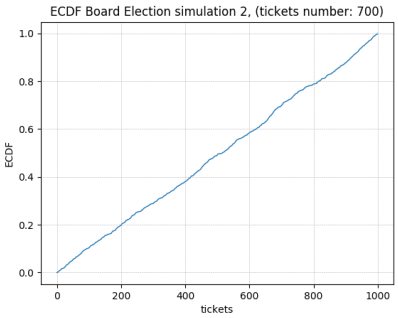


Figure 5.17: *ECDF Simulation 7: 700 tickets extracted.*

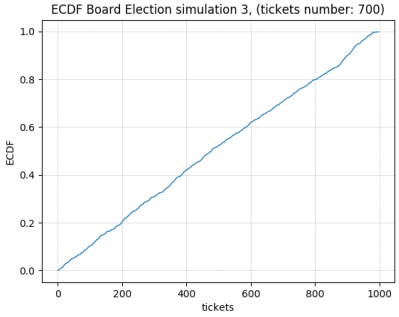
(a) *ECDF Experiment 1*



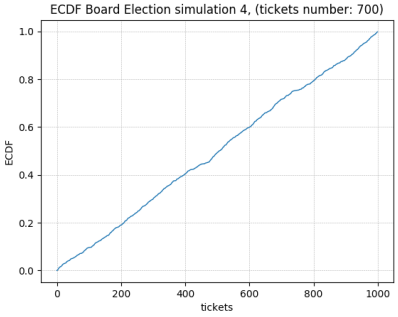
(b) *ECDF Experiment 2*



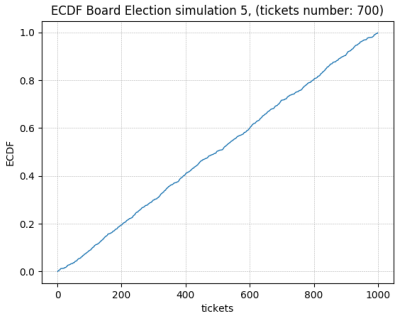
(c) *ECDF Experiment 3*



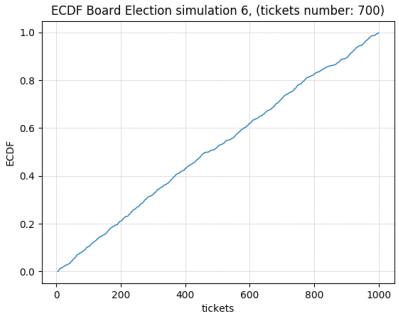
(d) *ECDF Experiment 4*



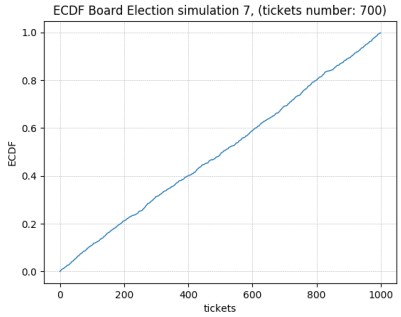
(e) *ECDF Experiment 5*



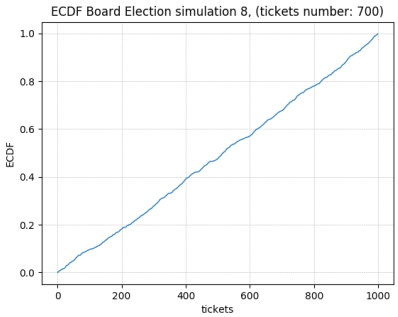
(f) *ECDF Experiment 6*



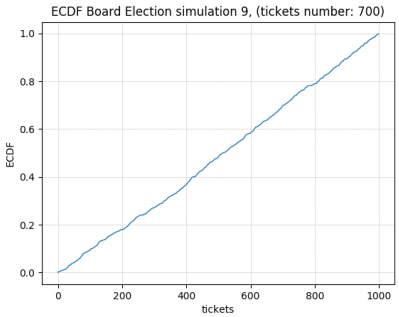
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

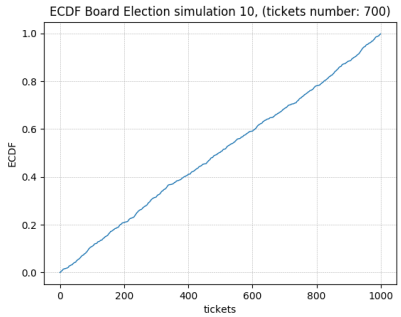
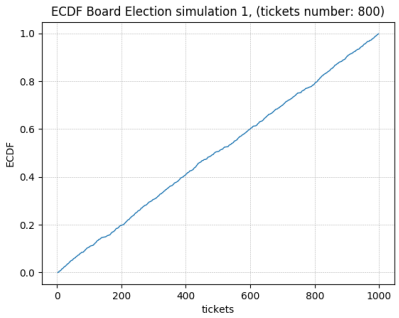
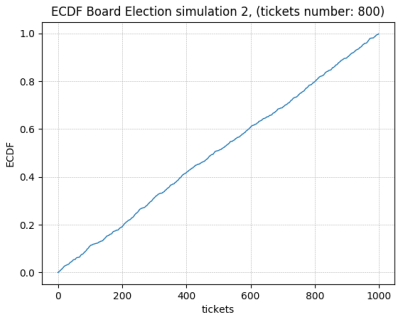


Figure 5.18: *ECDF Simulation 8: 800 tickets extracted.*

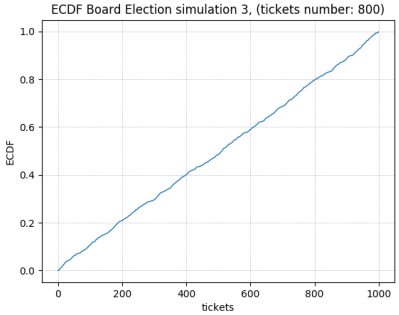
(a) *ECDF Experiment 1*



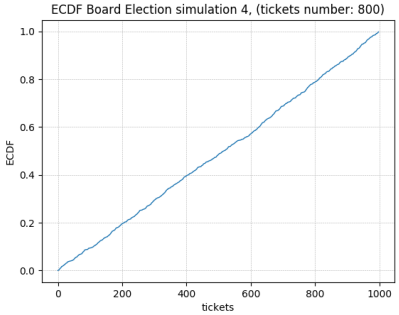
(b) *ECDF Experiment 2*



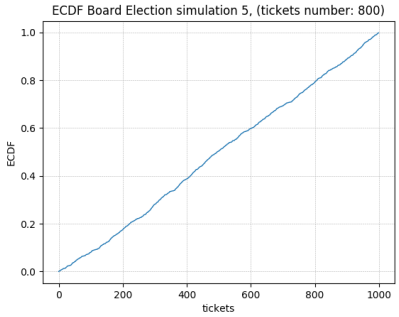
(c) *ECDF Experiment 3*



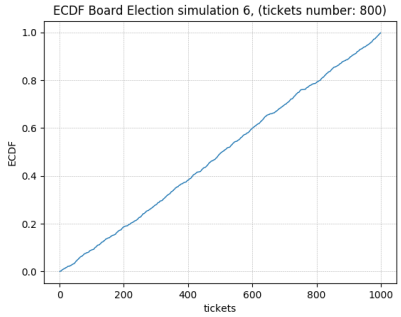
(d) *ECDF Experiment 4*



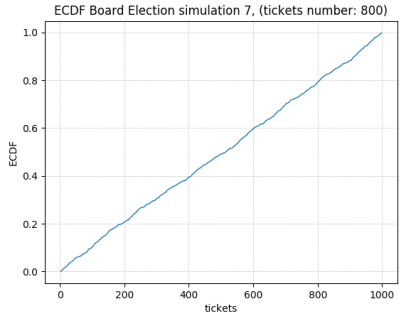
(e) *ECDF Experiment 5*



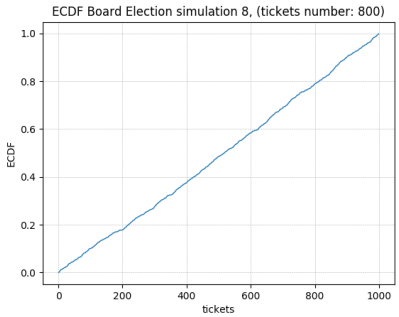
(f) *ECDF Experiment 6*



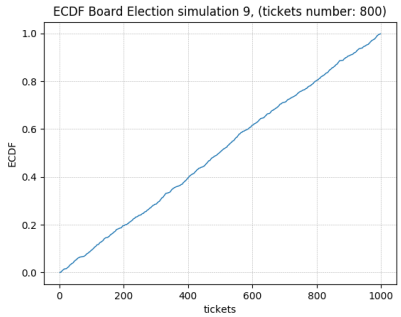
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

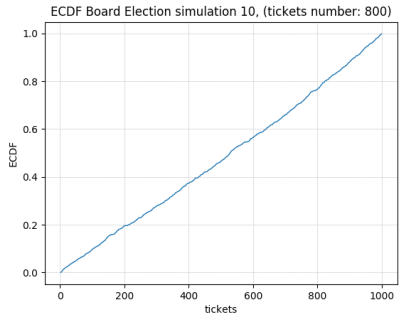
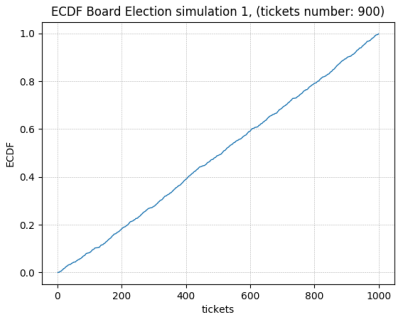
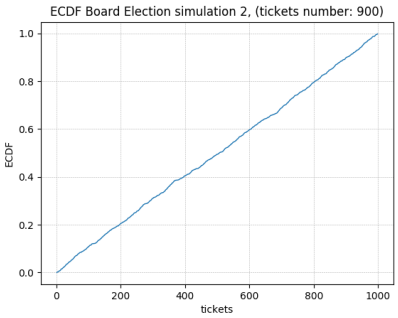


Figure 5.19: *ECDF Simulation 9: 900 tickets extracted.*

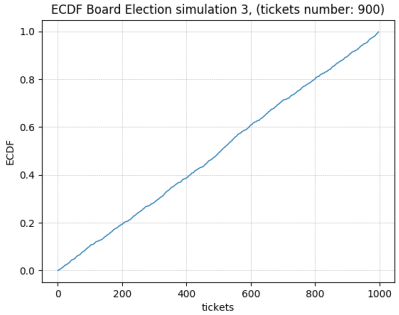
(a) *ECDF Experiment 1*



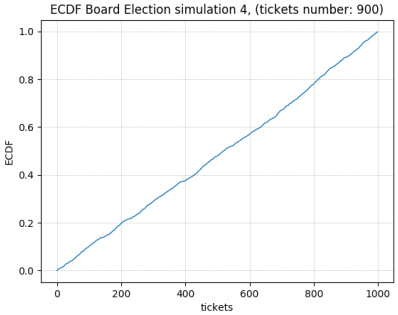
(b) *ECDF Experiment 2*



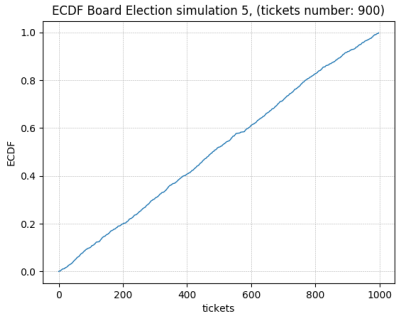
(c) *ECDF Experiment 3*



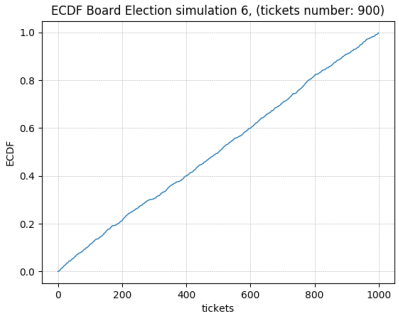
(d) *ECDF Experiment 4*



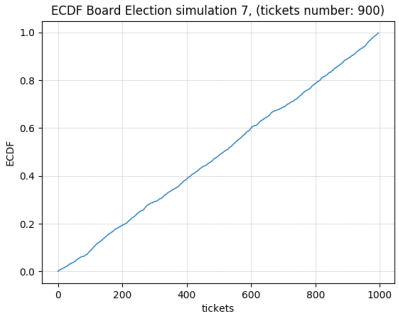
(e) *ECDF Experiment 5*



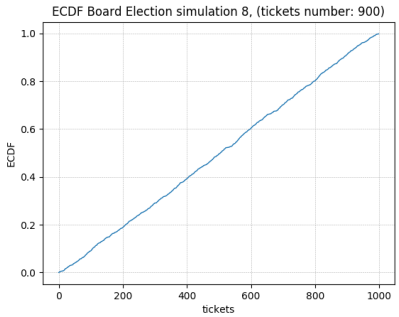
(f) *ECDF Experiment 6*



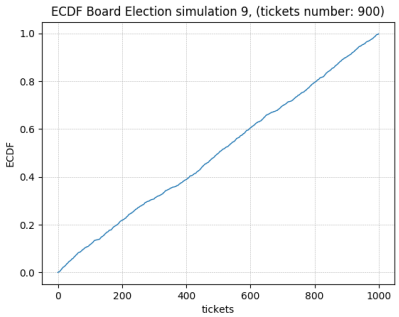
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*

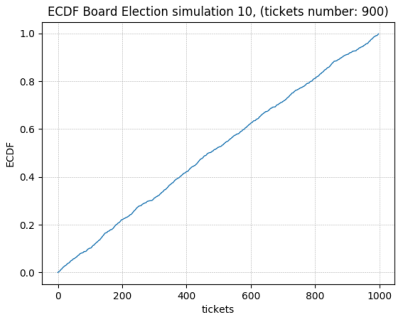
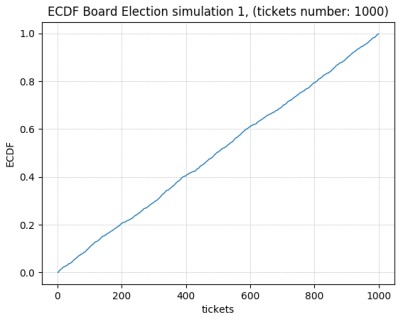
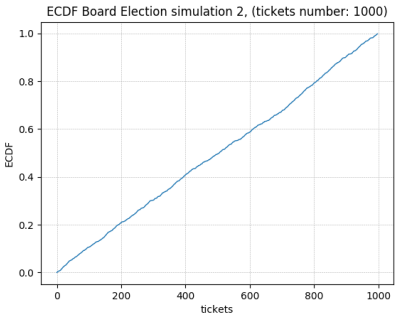


Figure 5.20: *ECDF Simulation 10: 1000 tickets extracted.*

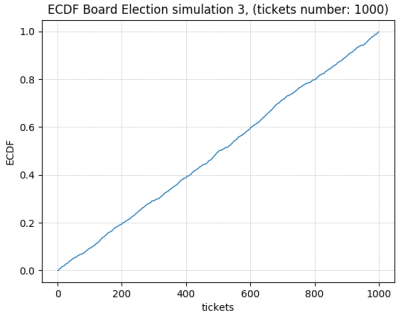
(a) *ECDF Experiment 1*



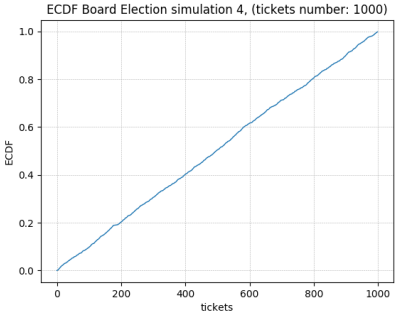
(b) *ECDF Experiment 2*



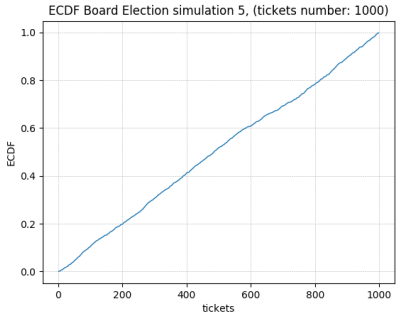
(c) *ECDF Experiment 3*



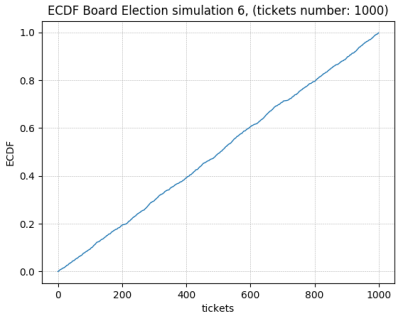
(d) *ECDF Experiment 4*



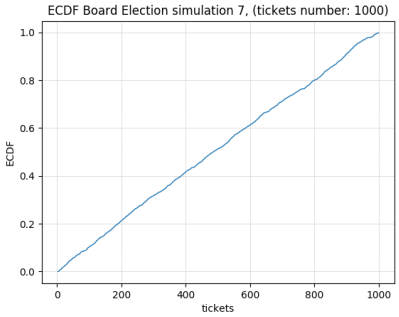
(e) *ECDF Experiment 5*



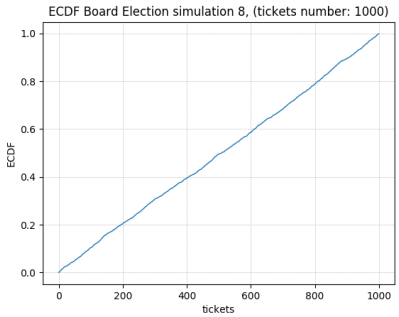
(f) *ECDF Experiment 6*



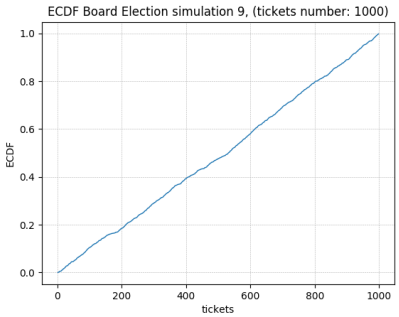
(g) *ECDF Experiment 7*



(h) *ECDF Experiment 8*



(i) *ECDF Experiment 9*



(j) *ECDF Experiment 10*