Imperial College London

Imperial College London

Department of Computing

A Multi-Agent Simulation Framework and Analysis of the IOTA Tangle

Author:
W. Manuel Zander

Supervisor:
Professor William J. Knottenbelt

Submitted in partial fulfilment of the requirements for the
MSc degree in Computing Science of Imperial College London

September 7, 2018

Abstract

The rising popularity of the Bitcoin and the underlying public distributed ledger, commonly known as blockchain, has paved the way for numerous similar protocols and platforms. These protocols are a way of securely implementing open, decentralised systems, which traditionally have been governed by trusted authorities. As an alternative to blockchain-based protocols, another class of distributed ledger protocols has been established. This class of protocols uses directed acyclic graphs as data structure to store transactions. One of the protocols using this approach is the IOTA protocol, calling its ledger a Tangle. A way to formally model this protocol is by using a single agent model, presented in the IOTA white paper.

We extend this model to a multi-agent model by assuming an asynchronous network – the actual setting in most peer-to-peer networks. In contrast, the single agent model as synchronous network does not adequately represent reality. A simulation framework for this model is developed and used to simulate and study properties of the resulting multi-agent Tangles. We saw a need for further research on this, since DAG-based protocols deal with complex data structures and algorithms, and their behaviour is not well understood yet, especially for many participating agents. Furthermore, side chains built by (malicious) agents occur in reality, but are difficult to analyse in a single agent context. This is where the approach presented in this dissertation can potentially close a gap.

The multi-agent Tangle model allows for the inclusion of a parameter representing the network latency of agents in the asynchronous network. The IOTA protocol and its consensus mechanism are heavily based on a specific method to build up the Tangle topology. This is done by performing a Markov Chain Monte Carlo (MCMC) algorithm, a weighted random walk, biased towards Tangle branches with a high cumulative weight. We developed a multi-agent simulation framework, which enables to simulate arbitrary scenarios and evaluate whether the proposed MCMC algorithm has the desired effects in terms of stability and security, assuming the majority of nodes is honest. We also highlight the challenges of certain calculations needed for the MCMC tip selection algorithm, which justifies the use of a concept called milestones in the IOTA implementation in practice.

For a two-agent case we illustrate how hashing power and the strength of the bias in the random walk affect the probabilities of referencing tips issued by each agent. We quantify this by a new measure we propose, called attachment probability. Furthermore, for a simulation setup with multiple agents based on randomly generated sparse network graphs, we show how well connected agents gain an advantage over less connected agents in terms of attachment probability, despite hashing power being equally split. This is the first time that Tangle properties and consensus are analysed in a multi-agent setting. We also present analysis of tip count stability in this setting for fully connected and disconnected agents.

The findings presented in this thesis have several important implications. First, the results suggest that the effectiveness of the consensus mechanism (and correspondingly the security of the IOTA protocol) depends on the hashing power of each agent and, by design, on the parameter by which the consensus mechanism is governed. Second, latency in an asynchronous network can play together with the consensus mechanism for the advantage of well connected agents. However, the computational power needed to carry out these analyses is substantial. Future large-scale simulations and formal analysis are needed to understand these phenomena in more detail. Nevertheless, we believe that Tangle research can greatly benefit from a realistic multi-agent model.

Acknowledgements

I am grateful to the Centre for Cryptocurrency Research and Engineering for giving me the opportunity to do research in the exciting world of cryptocurrencies and distributed ledgers. In particular, I would like to thank my supervisor and co-supervisor, William J. Knottenbelt and Dominik Harz, for giving me helpful advice, providing novel insights and support. Without their guidance and excellent supervision this thesis would not have been possible.

I would also like to thank Alon Gal from the IOTA Foundation for his help and insightful discussions around all IOTA-related questions.

Many thanks to my friends, who have made my studies in London such a fun, enriching and memorable experience.

Last, but most important of all, I owe my family the deepest gratitude for their ongoing support, encouragement and love, and for giving me the opportunity to fulfill my dreams.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Ever since the first Bitcoin [43] block got mined in 2009, distributed ledger technology has become a worldwide buzz and many such cryptocurrency protocols using peer-to-peer networks and consensus algorithms have been developed to challenge traditional centralised systems. In recent years the underlying distributed ledger of the Bitcoin, commonly known as blockchain, has seen tremendous growth in popularity, where the total market value of cryptocurrencies has temporarily reached values of more than 800 billion USD[1]. Cryptocurrencies result from the idea of developing decentralised systems, which enable participants to transact monetary value (or other data) with each other, without the need of a central authority. Instead, consensus mechanisms ensure trust among participants by combining cryptographic proof and an incentive for honest participants to support the network.

Nevertheless, blockchain-based systems suffer from inherent drawbacks, in particular the limited scalability of the technology [53, 16, 3]. The limited throughput of these approaches opposes the high volume of transactions needed in a global payment system and thus blockchain scalability has evolved as an active research field. The maximum throughput of Bitcoin is estimated to be around seven transactions per seconds, which is considerably lower then transaction rates mainstream payment providers such as Visa offer [14]. Many different approaches to deal with the scalability issue have been proposed, such as micropayment channels [46, 40, 17], sharding [41], and fundamental changes to protocol characteristics, like block generation parameters and size [19, 30], consensus [6, 33], proof-of-stake instead of proof-of-work [32, 15, 8], or an alternative data structure [38, 51, 55, 9].

The latter is where the IOTA protocol is adding to the list of potential solutions to the scalability bottleneck, i.e. by using a different data structure and consensus mechanism. IOTA's approach is primarily intended to enable machine-to-machine (M2M) micro-payments and communication in the IoT ecosystem. Especially with a future rise of IoT devices and machine-to-machine micropayments – Gartner predicts 20 billion connected devices in 2020 [26] – limited throughput serves as motivation to establish a novel distributed-ledger architecture, which arguably works well in such contexts. Hence, the "Tangle" [47] has been introduced and the IOTA Foundation has implemented the protocol as a cryptocurrency, which currently has a market capitalisation of 1.9 billion USD.[2]

The distributed ledger architecture implemented by IOTA uses a directed acyclic graph (DAG) data structure to store transactions, which is the main difference between this protocol and other blockchain-based protocols such as Bitcoin or Ethereum. The protocol does not use blocks and chains – instead each vertex in the DAG represents a transaction. In addition, a fundamentally different consensus mechanism is used, based on a Markov Chain Monte Carlo (MCMC) algorithm

[1] Source: https://coinmarketcap.com. Accessed: 2018-09-02.
[2] See footnote 1.

for attaching transactions to the ledger and determining confirmation levels or confidence in those.

The main motivation of this dissertation is to extend the single agent model of the Tangle as synchronous network, presented in the IOTA white paper [47], to a multi-agent model, and to develop an appropriate simulation framework for this model to examine behaviour of sub-Tangles and Tangle branches of multiple agents in an asynchronous setting. This is a very relevant research topic for both the IOTA Foundation and the broader ecosystem, since the Tangle is very complex mathematical object and in the current IOTA reference implementation (IRI) a centralised node, called coordinator, secures the network. Moreover, sub-Tangles (called side Tangles in practice) and spam have become a frequent phenomena (see Figure 1.1) [21]. Hence, "Analysis, modelling, and simulations for coo-less IOTA" [27] is a core research focus, as well as partitioning: "What happens when two nodes are very distant from each other?" [39]. Therefore, this dissertation aims at providing a more realistic model and analyses of the Tangle as a truly decentralised, asynchronous network.

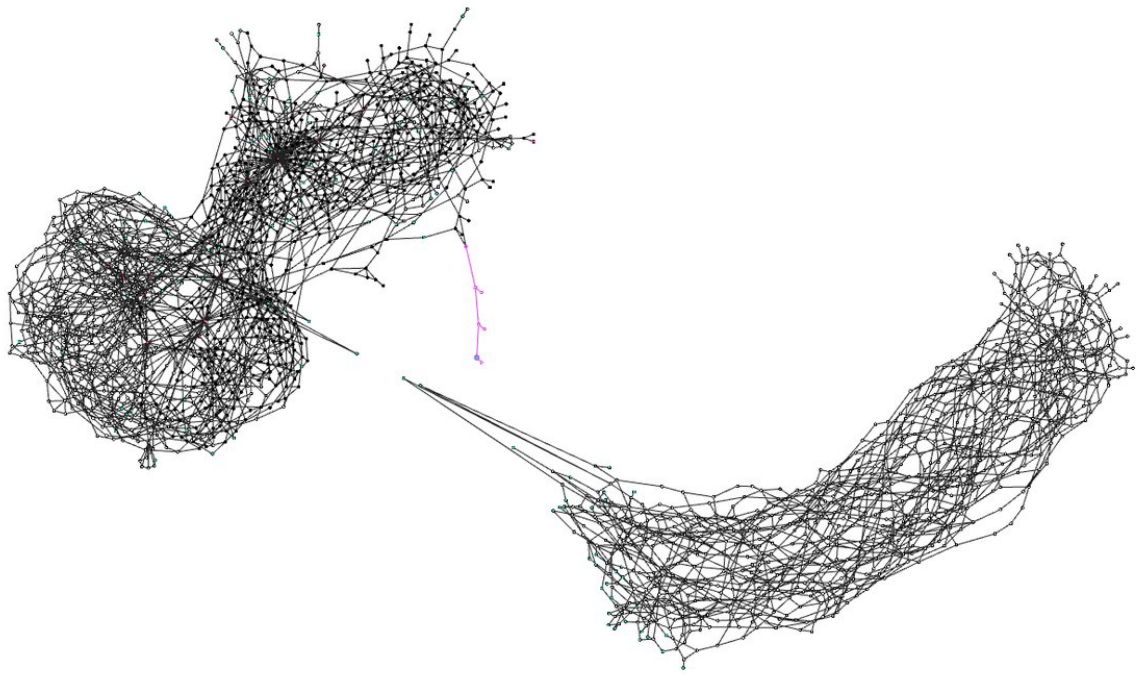

Figure 1.1: Screenshot from a real-world side Tangle in July 2018 [21, 56].

The project is conducted together with the Centre for Cryptocurrency Research and Engineering at Imperial College London and the IOTA Foundation based in Berlin.

1.2 Objectives

The aims of this dissertation are to:

1. Develop a simulation framework in Python for DAG-based distributed ledger protocols and implement the algorithms for the IOTA protocol for multiple agents, as well as methods to collect data and analyse the generated Tangles. The simulator shall be extendable, allowing for future work with simulations of other DAG-based protocols.

2. Additional features include graphs and visual illustrations of Tangle metrics, multiprocessing to run several simulations in parallel, and a feature to allow for changing certain simulation parameters (transaction rate and network latency) during a simulation. This flexibility is

needed to simulate different scenarios like partitioned, separately evolving sub-Tangles and Tangle branches.

3. Examine multi-agent simulation results and use different simulation scenarios to analyse proposed measures related to the consensus mechanism, security and stability of the generated Tangles. In particular, the two-agent case as a minimal model is interesting to analyse, as well as Tangles with larger numbers of participating agents.

1.3 Contributions

1.3.1 Formalisation of a multi-agent model and algorithms

Based on the single agent model presented in the white paper [47, 48], effectively a synchronous network, we describe how a multi-agent model can be formally defined as an extension. Especially through the introduction of the agent network notion and corresponding latencies between agents we make the model more realistic and closer to the real world, where hundreds or thousands of agents in P2P networks exist, and in fact represent asynchronous networks [36]. The latency plays a major role in the developed Tangle algorithms, especially to determine the transactions that are available (or visible) for tip selection.

1.3.2 A simulation framework for the multi-agent model

A multi-agent simulation framework has been developed in Python[3] and allows for simulations of Tangles with the parameters required by the model. Thus, the simulator can be used to simulate any scenario with an arbitrary number of agents, especially through the user's ability to provide a configuration file to specify parameter changes during simulations. The framework also includes algorithms to calculate and measure transaction- or agent-specific data. The user can use these to collect data and examine how multi-agent Tangles evolve over time, and analyse metrics like exit and attachment probabilities or confirmation confidences. The framework is easily extendable to further algorithms, if necessary. Furthermore, the framework has been build in a modular way, such that its basis can be used to implement other DAG-based protocols[4]. It also includes various plotting methods, a data export function, and a multiprocessing script.

1.3.3 Potential issues with the tip selection algorithm

In various chapters of this dissertation the computational complexity of the calculations required for the MCMC tip selection algorithm is stressed, which makes it infeasible to run simulations with very large numbers of transactions. The necessity to update cumulative weights of all referenced transactions after a completed tip selection round has been identified as a main performance bottleneck. Two different algorithms have been compared and tested and although not formally analysed, performance of simulations has been measured and revealed how challenging this task can be. This is a justification for a different approach in the IOTA implementation in practice, where only parts of the Tangle are used for the calculations.

1.3.4 Evaluations and comparisons of different multi-agent Tangle scenarios

The simulation framework has been designed in a way to allow for the implementation and evaluation of scenarios in which agents have different hashing-power. It is also possible to change transaction rate and the latency between agents during the simulation. This functionality has been used to examine how Tangle metrics evolve in different phases of the simulation, to draw conclusions about the impact of hashing power and latency. First, we examine the stability of the number of tips in a two-agent Tangle, and point out differences to the single agent model, caused by the latency between the two agents. Second, we evaluate different scenarios where agents have different hashing-power,

[3] Available under https://github.com/manuelzander/iota_simulation. Accessed: 2018-09-04.

[4] Other research projects have already started working on extensions ([51, 52]).

and analyse how the consensus mechanism governs the development of Tangle branches of each agent. We show that the consensus mechanism works as it is designed to work, or in other words, that the generated Tangles have the desired properties. For example, if a "heavy" main branch builds up, new incoming transactions are more likely to reference its tips, which in practice would contribute to the security of the network under certain conditions (discussed later in more detail, see [10]). Finally, we evaluate randomly generated networks of agents and show how centrally located agents are able to gain an advantage over less centrally located agents, that is they become more likely to be referenced, although hashing power is equally split in these experiments.

1.4 Outline

Chapter 2 describes the history of DAG-based distributed ledger projects, the intuition and main concepts behind the distributed consensus mechanism of IOTA, and the formal single agent model, as described in the white paper.

Chapter 3 extends the single agent model into an asynchronous multi-agent model and explains important concepts and definitions needed to implement the multi-agent model in code.

Chapter 4 discusses the main design and implementation choices of the multi-agent simulation framework. The main algorithms and implementation issues are pointed out, as well as the challenge of making large scale simulations feasible.

Chapter 5 assesses results of two-agent simulations, the minimal model in the multi-agent framework. Different scenarios are simulated and the results are used to show the effectiveness of the underlying consensus algorithm.

Chapter 6 presents simulations results of multiple agent networks and illustrates how latency in the asynchronous network model affects the agents' chances of being chosen in tip selection.

Chapter 7 concludes this dissertation by summarising and evaluating the main achievements and highlighting limitations and potential areas for future research.

1.5 Statement of originality

I declare that this thesis was composed by myself, and that the work that it presents is my own except where otherwise stated.

Chapter 2

Background

This section explains the basics of DAG-based distributed ledger technologies with a main focus on the IOTA protocol. Moreover, the basics of the formal single-agent Tangle model and related concepts for the simulation are presented.

2.1 DAG-based distributed ledger protocols

Since IOTA's DAG-based distributed ledger technology is a novel concept compared to traditional blockchain-based protocols, its mathematical foundations have been described and analysed in a white paper [47] and numerous other research papers from both inside and outside the IOTA Foundation [48, 35]. However, this protocol is not the only DAG-based one, and over the past years this area has become an active research field.

Early work which focused primarily on increasing the throughput time (number of transactions per second) of Bitcoin was done in 2013 by [53]. To address the scaling issue they propose improvements in the protocol, in particular the GHOST chain selection rule, which makes use of a tree structure to achieve high transaction rates. In 2015 they published another paper on GHOST [54], of which variants have been implemented by other protocols including Ethereum, and were then the first to present a distributed ledger architecture using a a directed acyclic graph structure to store *blocks* (block DAG) [38]. In contrast to the IOTA protocol, this approach still includes the concepts of blocks, miners and fees. Another approach is Byteball, which is the first model that eliminates blocks and stores *transactions* in a DAG structure, but uses a different consensus algorithm than proof-of-work [13]. Recently, the Hashgraph consensus algorithm [2], which uses a gossip protocol to build a DAG, has received attention for being a scalable and fast solution. Similarly, Avalanche [55] is also using a DAG of transactions and first experiments have shown high throughput and scalability. The approach most similar to the IOTA protocol is probably the DagCoin concept, which stores transactions in a DAG and each transaction does its own proof-of-work as well [37]. However, this proposal lacks formalisation and tip selection algorithms.

The authors of GHOST have published more relevant work, in particular the SPECTRE and, more recently, the PHANTOM protocols [52, 51]. SPECTRE uses a DAG to store blocks of transactions – a block DAG is essentially a generalisation of a blockchain protocol, as introduced in [38]. Moreover, they develop a new approach that separates mining and consensus from each other, resulting in a more scalable and secure protocol. This protocol supports transaction confirmation rates within the magnitude of seconds, while maintaining a security threshold of at least 50% [52]. In particular, they developed a voting procedure (called TxO protocol), in which each block votes for a pair-wise ordering of blocks. These are then combined in an aggregated majority vote to determine the system's consensus on transactions. The underlying rules of this voting system ensure protection against dishonest nodes, however, transactions in the protocol may not be linearly ordered. Based on this work, the authors propose the PHANTOM protocol, which makes total order of blocks and transactions possible (but takes more time to reach consensus, i.e. slower confirmation time)[51].

2.2 Tangle intuitions

2.2.1 Tangle data structure

A directed acyclic graph is a data-structure used to store information in many computational problems and domains. In the Tangle model, the *vertices* of the graph represent transactions, which are issued by the participants of the network (called nodes, but not to be confused with nodes of the graph). *Edges* are obtained by one transaction directly referencing another transaction. There is also a genesis transaction, which is the *root* of the graph, as can be seen in Figure 2.1. In the actual implementation of the IOTA cryptocurrency all IOTA tokens have been created with this genesis transaction, i.e. no more tokens can be "mined", which is a significant difference to the Bitcoin and Ethereum blockchains. Transactions are approved in the following manner: First, an algorithm is used to choose two other transactions to approve. This algorithm, the so-called *tip-selection algorithm*, is one of the core components of the protocol. *Tips* are transactions, which are not currently approved by any other transaction. In the next step, the node issuing a transaction checks if the two chosen transactions are conflicting. If this is the case, it does not approve them, but runs the tip-selection again. Lastly, in a process similar to the proof-of-work (POW) done in the Bitcoin protocol, the node tries to find a nonce, such that the hash of this nonce together with the transaction data has a certain number of trailing zeros. However, in terms of difficulty this "puzzle-solving" is by far not as computationally costly as the POW done in Bitcoin, but rather comparable to the *hashcash* used for preventing spam and DoS attacks in e-Mails [1], since it is done for each transaction separately.

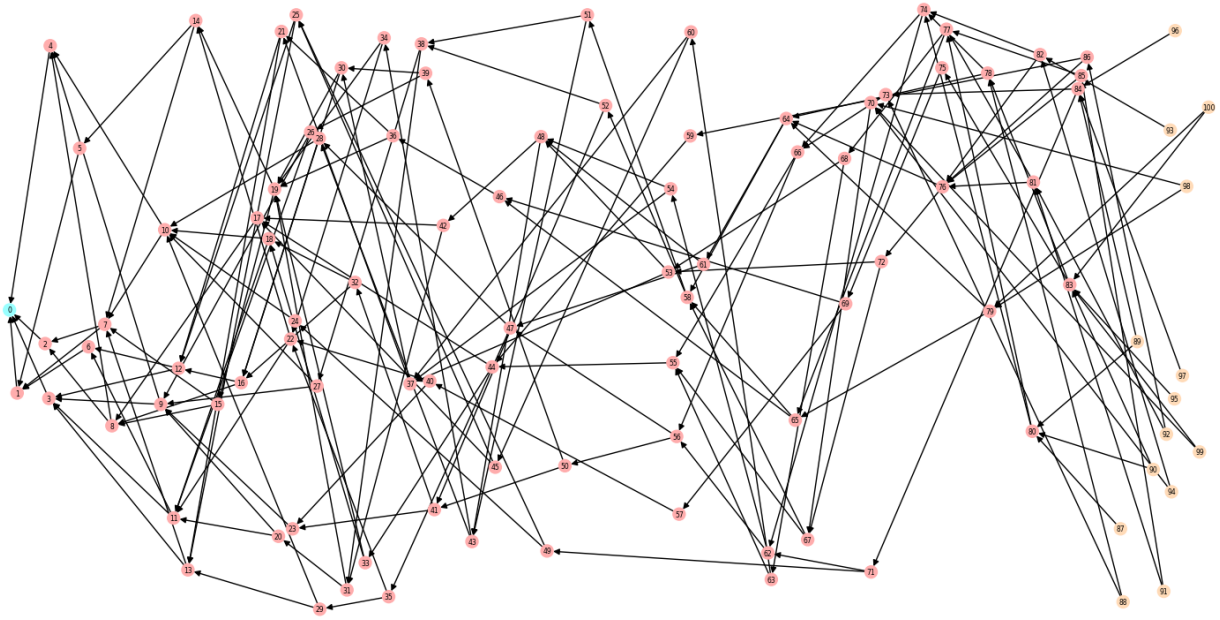

Figure 2.1: Example illustration of a Tangle with 100 transactions. The genesis is shown in cyan and the tips in orange. The Tangle "grows" towards the right side.

2.2.2 Consensus

Distributed ledger systems rely on certain methods and rules to ensure consistency and to make decentralised consensus work, which are commonly referred to as *consensus rules*. The earlier described proof-of-work mechanism is the backbone of such POW-based systems, since it allows nodes to participate in the consensus process with the weight of each node in the consensus voting process being proportional to its computational resources (*hashing power*). This means that no central authority is needed to decided on which state of the ledger is correct – instead the nodes agree on a state. There are also alternative consensus mechanisms, such as proof-of-stake, which do not rely on computational resources but on stakes (e.g. currency holdings) in the network. The consensus rule

in Bitcoin – called *Nakamoto consensus* – is the following: In case the nodes see several competing chains, they only consider the longest one. The idea behind this is based on the POW paradigm, i.e. that the longer chain was created by using large amounts of computational power and as long as the honest nodes accumulate more than 50% of the networks total computing power, it is infeasible for an attacker to fork the chain (and for instance conduct a *double spend-attack*). The Bitcoin white paper shows that the probability of an attacker outpacing the majority diminishes exponentially and thus the longest chain can be taken as the true state of the ledger [43]. In Ethereum the process works almost identical with a POW algorithm called Ethash [11, 57].

The IOTA consensus rule works fundamentally different, since there are no chained blocks of transactions that are considered together. Instead, each transaction is considered on a stand-alone basis and has a so-called *confirmation confidence*, which is a measure that a merchant will look at when transacting over the IOTA network and deciding when to consider a transaction valid. In fact, the Tangle network can contain transactions with conflicting information, but the nodes shall achieve consensus over which of the transactions will be orphaned and which will be kept in the ledger, with means in the Tangle terminology that there do not exist vertices pointing to both transactions. Thus, eventually only one of the transactions will be part of the "main branch" of the DAG and the orphaned transaction will not be indirectly approved by other transactions any more. This main branch is build by the use of a special consensus mechanism involving random walks for tip selection. The same mechanism can be used to establish the confirmation confidence, giving a probabilistic estimate for the acceptance level [28]:

1. Run the tip-selection algorithm many times, e.g. 100 times.
2. Calculate the proportion of how many selected tips directly or indirectly reference the transaction in question, e.g. 95 out of 100: 95% confirmation confidence for that transaction. 100 runs is just an arbitrary number, for a more statistically significant result the tip-selection algorithm could be run more times.

Hence, the tip selection algorithm must be designed in a way that is favours only valid and "important" transactions, based on the concepts of the *weight* and *cumulative weight* of a transaction, which will be elaborated on in the next section. Note that there is a more accurate and computationally efficient method to calculate confirmation confidence, which will be presented in a later chapter. This method is implemented in the simulation framework part of this dissertation and uses the concept of *exit probabilities*, i.e. calculated probabilities that a tip selection algorithm using a random walk is returning specific tips.

Note that this distributed consensus approach is not currently used in the IOTA implementation in practice. The IOTA white paper [47] shows that in "early stages" of the network, i.e. when the number of incoming honest transactions is not large enough compared to a potential attacker's computational resources, additional security measures are needed. Taking this further, [10] has shown that the network is only secure when the majority of nodes is honest and constantly uses its hashing power to create new transactions. This is similar as explained above for the Bitcoin protocol (honest nodes must accumulate more than 50% of the total hashing power, such that the longest chain rule can be used to form consensus). The conclusion is that if this condition is not given for the Tangle (for any tip selection strategy), an additional security measure, i.e. a centralised authority, is needed to prevent attacks and establish consensus.

This is why there is currently a different consensus mechanism used in the actual implementation, the *coordinator*, which is a node run by the IOTA Foundation. It issues zero-value *milestone* transactions in small, regular time intervals [28]. These special transactions approve other valid transactions without using the process described above – thereby immediately giving transactions a confirmation confidence of 100%. The foundation aims to shut down this security mechanism as soon as transaction volume reaches a sufficient level, but it remains open when exactly this is the

case. Adoption must be large enough such that the necessary security condition formulated by [10] holds. However, the simulation framework developed as part of this dissertation models the Tangle as described in [47], that is using the decentralised consensus algorithm described above.

2.2.3 Tip selection and random walks

The IOTA white paper suggests a specific tip selection algorithm used to determine which tips (unreferenced transactions) are chosen by a new incoming transaction. Note that this cannot be enforced by the protocol – tip selection is internal to a node – but it has been shown that "selfish" players will choose tip selection strategies that are similar to the "suggested" one [48] under the single agent model assumed in [47]. The tip selection algorithm is at the heart of decentralised consensus in IOTA and an active research topic. In particular, the recommended one (and also practically implemented in most nodes) is a *weighted random walk* – a Markov Chain Monte Carlo (MCMC) algorithm. In comparison to Bitcoin, where the longest chain rule is used, in IOTA the "heaviest branch" is used to determine which part of the ledger can be regarded as confirmed – again under the assumption that more than 50% of network participants are honest and constantly create transactions [10].

Heaviness can be quantified as follows: Every transaction (every vertex) in the DAG, is associated with a *weight*, which is a positive integer value [47]. In general, this weight indicates how much work the node issuing the transaction has invested in it, and more work is reflected in a higher weight. It becomes clear that this is closely related to the consensus algorithm, since if the consensus algorithm is based on weight, attackers would have to generate many (high-)weight transactions to successfully trick the network into having his spam or double-spend transactions to be confirmed or to be in the main branch. In fact it is not the weight itself that is being considered by the algorithm, but a transaction's *cumulative weight*. This is defined as the transaction's own weight plus the weights of all vertices approving it. For simplicity of the model used in this dissertation, a transaction's own weight can assumed to be equal to one, in which case the cumulative weight simply becomes one plus the total number of direct and indirect approvers.

The higher the cumulative weight, the more "important", or relevant, a transaction is. An attacker can theoretically try to build an own "heavy" branch in order to make a double-spend transaction possible, which is why [47] states that cumulative weight alone would be a misleading metric to decide on which of two conflicting transactions should be considered valid. Thus, the MCMC algorithm using random walks is proposed – under the assumption that the main branch combines the majority of hashing power in the network. In a *Markow chain* (or process) each step is independent from all previous steps, i.e. the outcome of the next step is solely dependent on the current state [49].

A random walk on the DAG works as follows: A *walker* is placed on the genesis vertex and starts walking towards the tips. Note that this is the technique used in the model and simulation framework part of this dissertation, but in practice the algorithm is local, that is the walker can start from any vertex "deep enough" in the Tangle [47]. In each step, the walker walks to a vertex directly approving the vertex it is currently sitting on, according to a vector of *transition probabilities*. If this is done in a completely random fashion, i.e. the walker randomly chooses an approving vertex is as next destination, this is called *unweighted random walk* (uniform transition probabilities). However, this approach suffers from several drawbacks, in particular it does not support security and aid in building a heavy branch. Moreover, it does not avoid "lazy tips", which are tips that tend to just confirm old transactions instead of recent ones. Lazy tips are not desired – recent transactions should be more likely to be approved, since the protocol aims at high transaction confirmation times and scalability. If transactions always just reference the same old transactions, the Tangle is not growing "forward" but something like a star graph could evolve. Overall, this results two desired features of the tip selection method [23]:

1. Transactions are unlikely to get abandoned once they accumulate a large number of approvers.

2. Honest transactions should get approved quickly.

In other words, the tip selection algorithm should favour the more "important" tips. Therefore, the random random walk is biased to favour heavier branches (and discourage lazy behaviour). This is where a parameter α comes into play, as a controlling parameter of how strong the bias is (or how important the cumulative weight is). An α of zero results in the previously mentioned unweighted random walk, whereas a very large α results in a deterministic walk, where at each step the heaviest transaction is chosen. Research has been conducted to determine the "ideal level" of α [24], given certain other parameters, which will be elaborated on later in this dissertation.

The transition probability for the random walk is defined as follows [47]:

$$P_{xy} = \frac{exp(-\alpha(W_x - W_y)}{\sum_{z:z \rightsquigarrow x} exp(-\alpha(W_x - W_z))} \tag{2.1}$$

where P_{xy} is the probability to walk from x to y, W_x is the cumulative weight of transaction x, $\alpha > 0$ can be chosen and $z \rightsquigarrow x$ means that transaction z directly approves transaction x (see Appendix for a simplified form we will use for implementation later).

Using this transition probability function it becomes clear that the probability to walk to y is exponentially increasing with its cumulative weight and hence the algorithm tends to walk to the relevant, heavy tips. These relevant tips are referred to by new incoming transactions and the heavy branch of the Tangle grows. At the same time, the probability to choose a lazy tip is small, since a lazy tip would have a very small cumulative weight compared to another transaction which is not lazy. Eventually, lazy branches get orphaned. Furthermore, the security of the network is strengthened: For an attacker the same argument holds, i.e. assuming that an attacker is not able to catch up with honest nodes by artificially inflating the cumulative weight of a malicious side chain, the tip selection algorithm would favour transactions in the main branch with much higher cumulative weights. This again holds only if the honest nodes combine 50% of the total hashing power and constantly create transactions [10].

Finally, we can connect this logic back to the consensus mechanism discussion: If the same weighted-random walk tip-selection algorithm used for building the Tangle is run many times to determine a probabilistic measure of how often a transaction is referenced by relevant tips, this transaction can be assumed to be (and stay) in the main branch with a high confidence, if the measure is high.

2.3 Modelling the system – a single agent model

The Tangle model presented in [47] is a single agent model, which means that only one node (which we refer to as *agent*) has a complete view on the Tangle – the network is *synchronous*. Here, this model will be described, in order to establish a common language which is in line with other Tangle research. This makes it easier to enable collaboration and further research efforts, as well as to compare the results with other Tangle simulations based on the same model and assumptions.

It is for the same aforementioned reasons, that the presented model, a *continuous time model*, is chosen over an alternative *discrete time model*. Most research and simulations to date have used the continuous time model [35], which allows for adding more realism to the model with respect to timing and visibility of transactions (this will be especially useful for the extension to a multi-agent model). At the same time, more realism means more complexity, whereas the discrete model is simpler, easier to implement and to analyse [34]. In the discrete time model, each time step can be associated with several newly issued transactions. The exact number is chosen from Poisson distribution, which is on average λ transactions. As explained in the next paragraph, a parameter h plays an important role with respect to visibility of transactions in the continuous model, but

is omitted in the discrete one, since it assumes that the preparation and propagation calculations for a transaction are done within one time step [34]. Visibility will play an even more important role in a realistic model for multi-agent Tangles, since these represent asynchronous networks in which agents have their own view on the network. An *asynchronous network* is a more adequate representation of the reality of peer-to-peer networks [36]. This is why we chose complexity over simplicity, in order to model the system as close as possible to the IOTA network in real world.

The continuous time model is based on several important assumptions. First, when transactions are assumed to be issued by many roughly independent entities, a *Poisson point process* can be used for modelling [47]. These processes are commonly used to model processes where events happen randomly, but still with a certain rate, such number of car accidents in a specific area. In other words, this means that inter-arrival times (i.e. the elapsed time between to consecutive transactions) follow an exponential distribution Exp(λ). Furthermore, in [47] this has been analytically shown to be an adequate assumption. The rate parameter is called λ, describing the rate of incoming transactions. For simplicity, this rate is assumed to stay constant over time. Moreover, a crucial assumption is that the agents (or rather the single agent) issuing the new transactions have approximately the same computational power, which is described by a parameter h, the average time needed to prepare and propagate a transaction (computational cost for tip selection, verification and POW plus network latency).

In other words, this parameter h represents a time period where a transaction is not visible to the other newly incoming transactions, thus it could be called *invisible period*. This means that a transaction v issued at time $T(v)$ only becomes visible at time $T(v) + h$ and, equivalently, the agent always just knows the state of the Tangle from h time units ago. Further, this implies that transactions referenced by v are still seen as tips (unreferenced) for every other newly incoming transaction – until h has passed.

Having this in mind, we can define the single agent Tangle as a stochastic process on the space of rooted DAGs, along the lines of [48]:

Let $\mathcal{G}$ be the set of all DAGs $G = (V, E)$ with the following properties:

- G is finite and the multiplicity of any edge is at most 2.
- There is a unique vertex G, called the genesis, such that $\deg_{\text{out}}(v) = 2$ for all $v \in V \backslash \{\text{G}\}$, and $\deg_{\text{out}}(\text{G}) = 0$.
- Any $v \in V$ such that $v \neq \text{G}$ references G.

The single agent Tangle is a continuous-time Markov process on the space $\mathcal{G}$. At time $t \geq 0$ the state of the Tangle is a DAG $\mathcal{T}(t) \equiv (V(t), E(t))$, where $V(t)$ is the set of vertices and $E(t)$ is the multi-set of directed edges at time t. Further dynamics are described by the following rules [48]:

- The initial state of the Tangle at $t = 0$ is defined by $V(0) = \{\text{G}\}, E(0) = \emptyset$.
- Over time, the Tangle grows: if $0 < t_1 < t_2$ then $V(t_1) \subseteq V(t_2)$ and also $E(t_1) \subseteq E(t_2)$.
- New transactions are arriving with a fixed rate $\lambda > 0$ according to a Poisson process, and become vertices of the Tangle.
- Each arriving transaction v chooses two vertices v', v'' to approve, not necessarily distinct. We then say that v approves v' and v'' and we add the edges (v, v') and (v, v'') to the Tangle.
- If $T(v) = t$ then $V(t+) = V(t) \cup \{v\}$ and $E(t+) = E(t) \cup \{(v, v'), (v, v'')\}$.

The model does not require one specific tip-selection strategy to be used, we briefly pointed out before that this is internal to a node (the issuing agent). The reasoning is also summarised in [48]. The recommended tip selection algorithm is a *weighted random walk* with transition probabilities as defined by Equation 2.1. There is also a version of the algorithm including backtracking, but this is out of the scope of this dissertation to reduce complexity in the model and simulation, which would have been higher with an additional backtracking probability parameter.

In summary, the main model parameters so far are:

1. λ, *the average rate of incoming transactions*
2. h, *the average computational cost for tip selection, verification and POW plus network latency*
3. α, *the randomness of the MCMC algorithm*

This mathematical description will serve as a basis for the development of the simulation, however for a multi-agent simulation, the model has to be extended in order to allow for each agent having an own view on the Tangle.

Chapter 3

Extension to a multi-agent model

One of the main motivations of this project is to develop a simulation for a *multi-agent model*. This allows for analysis and experiments with any number of agents in a more realistic asynchronous network setting, where each agent has a different view on the Tangle and subsequently various phenomena with regard to partitioning, security and stability can be explored.

In particular, *partitioning* means that the different agents are approving only transactions in their own version of the Tangle (or sub-Tangle) [25]. These sub-Tangles can overlap or be completely disjoint, depending on the network latency between agents.

3.1 A network of agents

First, it is helpful to formally describe how agents are connected. We define a network of agents, which is a weighted undirected graph. We define the network $G \equiv (A, E, f)$ where A is the set of agents, $E \subseteq A \times A$ is the set of edges and $f : A \times A \to \mathbb{R}$ is the weight function. We enforce $(a_1, a_2) \in E \Leftrightarrow (a_2, a_1) \in E$ (because G is an undirected graph) $f(a, b) \geq 0$ (non-negative weights), $f(a, a) = 0$ (zero self-distance), and also extend f so that $f(a, b) = \infty \Leftrightarrow (a, b) \notin E$ [25].

The key point of modelling a multi-agent model is, as mentioned before, the visibility of transactions, which is distinct for each agent, depending on a parameter d, which determines the network latency, i.e. the distance between the agents. This is essential because each agent must only consider the currently visible sub-Tangle for algorithms including tip selection and calculating cumulative weights.

We are always just interested in the shortest path between two agents $d\,(a_1, a_2)$, that is the distance function $d\ A \times A \to \mathbb{R}$ is defined as: $d\,(a_1, a_2) \equiv \min_{P \in P(a_1 \to a_2)} \sum_{e \in P} f(e)$ where $P\,(a_1 \to a_2)$ is the set of all paths from a_1 to a_2. Note that d is measured in the same unit as h before, that is in arbitrary units of time.

This in turn means the previously defined parameter h now only defines the average computational cost for tip selection, tip verification and POW, and d exclusively represents network latency. Accordingly, a transaction v issued at time $T(v)$ becomes visible to an agent a after $T(v) + d\,(a, a_v) + h$.

Now the in the previous chapter described stochastic process can be extended to a multi-agent model: There can be $N \in \mathbb{N}$ agents, and each time a new transaction v arrives, an issuing agent n for this transaction is determined with probabilities in a vector c (uniform by default), such that $1 \leq n \leq N$. We then say that $A(v) = i$ when v was issued by the i-th agent. Moreover, the agent-network from above translates into a $N \times N$ distance matrix d over $\mathbb{R}$, such that each distance from one agent to another is the shortest possible distance:

- For all $i : d_{ii} = 0$
- For all $i \neq j : d_{ij} > 0$
- For all $i, j, k : d_{ij} + d_{jk} \geq d_{ik}$

3.2 Visibility of sub-Tangles

This model adequately equips us with the notation to model multiple agents with different views on the Tangle, which can be determined by $A(v), T(v), d,$ and h. These parameters will thus play a key role in the simulation to return the visible transactions (and valid tips) that an agent can attach a new transaction to.

The set of visible transactions $V_i(t)$ for agent i at time t is determined by:

$$V_i(t) \equiv \{v \in V(t) \backslash \{\mathrm{G}\} | t - T(v) \geq d_{iA(v)} + h\} \cup \{\mathrm{G}\} \tag{3.1}$$

In addition, it will be useful to define the i-th edge set at time t:

$$E_i(t) \equiv \{(v_1, v_2) \in E | \{v_1, v_2\} \subseteq V_i(t)\} \tag{3.2}$$

Finally, the i-th sub-Tangle set at time t is:

$$\mathcal{T}_i(t) \equiv (V_i(t), E_i(t)) \tag{3.3}$$

In summary, we have extended the model described in the previous section by adding the notion of multiple agents and a parameter d for a matrix of shortest paths between all agents. Hence, the final parameter set is:

1. λ, *the average rate of incoming transactions*
2. h, *the average computational cost for tip selection, verification and POW*
3. α, *the randomness of the MCMC algorithm*
4. N, *the number of agents*
5. c, *the agent choice probabilities (default distribution is uniform)*
6. d, *the distance matrix with network latencies between agents*

Chapter 4

Development of a multi-agent simulation

This chapter describes the development of the simulation framework for multi-agent simulations, which is based on the formally described Tangle model in the previous chapter and can be used to conduct analysis of multi-agent Tangle metrics for different scenarios, effects of different tip-selection strategies and visualisations. First, the initial aims for the simulator are highlighted, before describing the design of an efficient simulation algorithm and the implementation in an objected-oriented style. Simulation results and main findings are presented in later chapters.

4.1 Aims for the simulator

A main aim of this dissertation is to develop a simulation for IOTA networks with an arbitrary number of participating agents, and to use this simulator to add to existing Tangle research by exploring behaviour of sub-Tangles in the asynchronous network setting. Thus, the main aims for the simulator are as follows:

- The simulator should allow for the simulation of DAG-based distributed ledger systems, it should build a DAG data-structure holding the transaction data. The focus of this dissertation is the implementation of the algorithms specific to the IOTA protocol.
- The user should be able to specify the main model parameters for the simulation, and a desired number of transactions, to simulate an appropriate Tangle. It should be possible to change certain parameters at specified points during the simulation, for example to simulate a large main-Tangle and a smaller sub-Tangle. The user should also be able to choose between different tip selection algorithms for comparison purposes. This means that the simulator should be build in a flexible and modular way.
- The simulator should be able to collect data (number of tips, cumulative weights, confirmation confidence, attachment probabilities), which can be used to measure and analyse Tangle behaviour. The user should also be able to export this data.
- The simulator should be able to produce appropriate visualisations and support a multiprocessing mode to run multiple simulations in parallel.

4.2 Design

The simulatior has been developed in two main phases: First, a single agent model has been implemented. This has been used to reproduce results of [35] to ensure the correct functioning of the implemented IOTA protocol algorithms. Second, building upon this work, a main `class Multi_Agent_Simulation` includes the necessary extensions for simulating multiple agents. Since this dissertation focuses on the multi-agent model, this chapter will mainly deal with the `Multi_Agent_Simulation` class and how it is implementing the appropriate algorithms to simulate multi-agent Tangles. A UML diagram can be found in Appendix A.

4.2.1 An efficient simulation algorithm

The `class Multi_Agent_Simulation` has two main functions a user needs for running a simulation, `setup()` and a `run()`. `setup()` initialises `Agent` and `Transaction` objects, whereas `run()` iterates over all `Transaction` objects except for the genesis ($v \in V(t)\backslash\{\mathrm{G}\}$) in the order of their arrival times – this is the main simulation loop. These two functions together cover the main simulation algorithm:

Algorithm 1: Setup and run methods of a simulation instance

Input : Number of transactions x, number of agents N, λ, h $(= 1)$, d, α, c

Output: Set of all sub-Tangles $\mathcal{T}_i(t)$, where $t = at_x$ (arrival time of last transaction), for every agent $i \in A$

1 **procedure** setup()
2 Create A, consisting of N agents
3 Generate x inter-arrival times $\tau_1, \ldots, \tau_x \sim \mathrm{Exp}(\lambda)$
4 Initialise $V(t)$ with arrival times: at_v for each $v \in V(t)\backslash\{\mathrm{G}\}$ is given as $at_1 = \tau_1$ and $at_i = at_{i-1} + \tau_i$
5 $at_{\mathrm{G}} = 0$ `// Genesis has arrival time 0`
6 **end**

7 **procedure** run()
8 **foreach** $v \in V(t)\backslash\{\mathrm{G}\}$ **do** `// For every transaction except genesis, in order of arrival times`
9 Choose issuing agent $A(v) = i \in A$ with probability according to c
`// Get set of visible transactions, time is arrival time of current transaction`
10 $V_i(t), I_i(t) = visible\_transactions(t, h, d, A(v))$, where $t = at_v$
`// Get set of valid tips`
11 $T_i(t) = valid\_tips(V_i(t), I_i(t))$
`// Choose tips and add edges to Tangle`
12 $tip\_selection(v, T_i(t), V_i(t), \alpha)$ `// Different algorithms to choose from`
`// Update cumulative weights`
13 $update\_cumulative\_weights(v)$
14 **end**
15 **end**

Note that this algorithm is not much different from implementing a single agent model, the main differences are the functions to determine $V_i(t)$ (line 10) and the valid tips $T_i(t)$ (line 11) to attach to. Moreover, the calculation of exit probabilities and confirmation confidences are more complex with multiple agents – these algorithms will be elaborated on later in this chapter.

The `Multi_Agent_Simulation` class has several additional important functions not covered in the above algorithm, most importantly to measure Tangle behaviour and determine confirmation confidences. Calculating confirmation confidences could either be done after each iteration in the main loop – after updating cumulative weights, or just in the very end, when the Tangle creation has been completed.

4.2.2 Parameter changes during a simulation

There are two simulation parameters, i.e. the distance matrix d and the agent choice probabilities c, which we might want to change during a running simulation, i.e. during the execution of `run()`, for example to create a large main-Tangle and a smaller sub-Tangle, and let them merge at some later point. Suppose we have $N = 2$ agents, an initial distance of ∞ (infinity), such that each agent can only see the transactions in his own sub-Tangle, and a non-uniform agent choice distribution with probabilities $c = [1.0, 0.0]$, such that every transaction is initially issued by agent 0. To put it differently, a main-Tangle builds up, whereas agent 1 issues no transactions at all.

However, if we want this to change at some point, we can specify a certain `transaction.id` as a checkpoint (which we will call *step*), for example "when the 100th transaction v comes in, change the agent choice distribution to be uniform", such that both agents issue the approximate same number of transactions. In this example we have $step = 100$. We could keep the distance between the agents at ∞, or we could set it to a smaller value, meaning that the agents see part of the others sub-Tangle, start to reference it, and the sub-Tangles merge. This means that we need some kind of mechanism to check for changes at a certain v. Note that it is not possible to do checks at a certain time t, because the main simulation loop iterates over all transactions in the order of their arrival times. In each iteration of the main loop $t = at_v$, so saying "check at $t = 5$" would not work and therefore we use the `transaction.id`, or *step*, to do checks. In order to make such parameter changes possible and easy to specify for the user, a special configuration file `config.ini` can be provided as command line argument, which holds the information of which *step* triggers a change, and what the new parameter values should be (see Appendix D for an example). It should also hold the main simulation parameters, but details are discussed later. A check in each iteration of the main simulation loop detects the *change events* and changes the parameter values.

4.2.3 Use of Python and supporting libraries

The main language the simulation framework has been developed in is Python 3.6.3. This has been decided since Python offers many helpful libraries for this project, in particular `NetworkX`, which is a popular package for the "creation, manipulation, and study of the structure, dynamics, and functions of complex networks" [44]. As such, it offers all common graph data structures and allows for a convenient creation of a DAG structure. This also makes the project extendable for other DAG-based protocols, since users do not need to worry about the underlying data structure too much since it is implemented in an optimised way by the library using dictionaries, which are hash-tables in Python. This allows "fast look-up with reasonable storage for large sparse networks" [44]. Moreover, `NetworkX` offers many standard algorithms for directed acyclic graphs written in C, C++, and FORTRAN, which are useful and fast for implementing the Tangle algorithms. The use of the `NetworkX` library therefore abstracts away the complexity of creating an efficient data structure and should overall improve the performance of the simulator.

Another useful library for Tangle visualisations is `Matplotlib`, a standard library for the creation of high quality figures [42]. Because `NetworkX` has a drawing interface using `Matplotlib`, these two packages make it convenient to create appropriate illustrations, which will be used to present the analysis and finding in later chapters.

The next sections will explain the implementation of the simulation framework in further depth, especially the main-Tangle algorithms. Additionally, implementation challenges are discussed.

4.3 Implementation of the main simulation loop

In order to create a new instance of a `Multi_Agent_Simulation`, the user has to specify the following parameters, according to the multi-agent model described in Chapter 2: the number of transactions (Tangle size), the transaction rate λ, the number of agents N, α for the randomness of the random

walk, h, d, c, and the desired tip selection algorithm. Note that c is by default a probability vector with uniform probabilities, but can be changed by the user if desired. Also important to mention is that the user can specify an arbitrary value for h, but we usually assume $h = 1$ (units of time) to be fixed. On the one hand this reduces complexity and cancels out one parameter to deal with, and on the other hand is has been shown by [35] that properties of the Tangle do not depend on both λ and h, but on the product $\lambda * h$. Once a simulation instance has been created, the user can `setup()` and `run()` it.

In the following subsections the most important and notable implementation details are explained.

4.3.1 Simulation setup

As explained earlier, the `NetworkX` library is useful as data structure for the Tangle, and thus every simulation instance has a `self.DG`, a directed graph object to which we can easily add nodes (transactions) and edges (after tip selection):

```
self.DG = nx.DiGraph()
```

Moreover, since we model the arrival of new transactions with as a Poisson process with rate λ, the inter-arrival times are sampled from an exponential distribution and concerted into the transaction arrival times by using a cumulative sum function from `NumPy`. Furthermore, it is worth mentioning that each simulation instance has two lists holding the `Transaction` and `Agent` objects, which are both initialised in the `setup()`.

4.3.2 Main simulation loop

In `run()` the main simulation loop is implemented almost one-to-one as outlined in the designd section before, by calling the appropriate helper functions. Before this is done, we just need to add a transaction to the `self.DG` object. Note that the `NetworkX` library makes it possible to store any kind of (hashable) data in a graph node, hence we can add a `Transaction` object as new node in each iteration:

```
self.DG.add_node(transaction,...)
```

Subsequently, the tip selection and update of cumulative weights functions are called. For this to work we first need to determine the currently visible transactions and valid tips.

4.4 Algorithms for visible transactions and valid tips

4.4.1 Visible transactions

In the previous chapter we have already defined $V_i(t)$, the set of visible transactions for agent i at the current time t. We explained that the multi-agent model allows us to simulate agents having different views on the Tangle, which can be determined by $A(v), T(v), d,$ and h. Consequently, in Algorithm 2 we implement Equation 3.1 – repeated here for convenience:

$$V_i(t) \equiv \{v \in V(t) \backslash \{\mathrm{G}\} | t - T(v) \geq d_{iA(v)} + h\} \cup \{\mathrm{G}\}$$

Remember that $A(v) = i$ when transaction v was issued by the i-th agent and $T(v)$ is the issuance (arrival) time of v. This is one of the most crucial functions for the simulator and has been implemented in code as `get_visible_transactions()`. This function is used during the tip selection algorithm, for two reasons. First, we need it to determine the set of valid tips, since only visible tips can be selected. Second, during the random walks we need to make sure to walk only on visible transactions of the respective agent. Furthermore, $V_i(t)$ plays a role in the update of cumulative weights, and in the calculation of exit probabilities and confirmation confidences, since we always

have to make sure to deal with the appropriate sub-Tangle of an agent. For these reasons the function plays a central role in the simulation framework and further implementation details are explained next.

As shown in Algorithm 2, two nested loops are used. The outer one iterates over all transactions in the graph, whereas the inner one loops over all agents, since we need $V_i(t)$ for every agent $i \in A$. This is because in the cumulative weight update we need to consider all agents, whereas for tip selection we would effectively only need $V_i(t)$ for the issuing agent i of the incoming transaction. We then make sure that the genesis transaction is added to $V_i(t)$. The genesis is always visible and additionally it is not associated with an agent, which in turn means that there is no distance $d_{iA(v)}$ in the distance matrix if $v = \mathrm{G}$. However, $d_{iA(v)}$ is crucial for checking the visibility of all other transactions. Visibility essentially means that at a time t (in the case of tip selection the "current time" t is the arrival time of the transaction which we want to select the tip for) all transactions are visible, whose arrival time plus latency and distance has elapsed (is smaller than or equal to t). Thus we check the condition $t \geq T(v) + h + \mathit{distance}$ ($\mathit{distance}$ is given by d), and add the transaction to $V_i(t)$ if true. Since the loop is nested, we can get the correct $\mathit{distance}$ number from the matrix by:

```
distance = self.distances[agent.id][transaction.agent.id]
```

Note that in the code we also record all non-visible transactions for the agent who issued the incoming transaction, since we will use this in the next section to determine the set of valid tips. We call these "invisible" and write (in the opposite way of $V_i(t)$):

$$I_i(t) \equiv \{v \in V(t) \backslash \{\mathrm{G}\} | t - T(v) < d_{iA(v)} + h\} \tag{4.1}$$

4.4.2 Valid tips

For tip selection, we need a function to return the set of valid tips an incoming transaction can choose to reference. This has not been defined so far. Let $T_i(t)$ be the set of unapproved vertices for agent i at time t:

$$T_i(t) \equiv \{v \in V_i(t) | \deg_{\mathrm{in}}(v) = 0\}$$

We see that this definition uses the "output" of the previous function, $V_i(t)$. The definition determines all visible transactions with no approvers. While this seems like a sufficient condition to check, there is another factor adding complexity. It could happen, that a transaction x references a transaction y, but x is not visible yet for the agent of a newly incoming transaction z, for which a tip shall be selected. Consequently, we have $\deg_{\mathrm{in}}(y) = 1$ for $A(x)$, but $A(z)$ sees $deg_{\mathrm{in}}(y) = 0$ since it does not know about x.

The problem in the implementation is that once a `Transaction` object v has been referenced at least once, $\deg_{\mathrm{in}}(v) \geq 1$, no matter for which agent. This is because the `NetworkX predecessors()` function is used to identify the (number of) approvers of a transaction. For our purposes, the algorithm to determine $T_i(t)$ should check if all the approvers are not visible for the agent of the incoming transaction. If yes, we effectively have a valid tip. An example is illustrated in Figure 4.1, where we have two agents, and transactions 11 and 12 reference transaction 1, although this transaction has been attached to by transaction 3 of the other agent already. This is caused by the distance between the agents, i.e. the red agent does not see transaction 3 (latency too high) when it does tip selection for both transactions 11 and 12, and hence identifies transaction 1 as valid tip and references it.

Algorithm 2: Algorithm to determine $V_i(t)$ and $I_i(t)$

visible_transactions $(t, h, d, A(x))$

Input : A time t, h $(= 1)$, d, the agent of an (incoming) transaction $A(x)$;
all transactions $V(t)$ globally available

Output: $V_i(t)$, for every agent $i \in A$, $I_i(t)$, for agent $A(x) = i$

```
1  V_i(t) = ∅                                                    // Empty set
2  I_i(t) = ∅                                                    // Empty set
3  foreach v ∈ V(t) do                                           // For every transaction
4      foreach i ∈ A do                                          // For every agent
5          if v = G then                                         // Genesis is always visible
6              V_i(t) = V_i(t) ∪ G
7          else                                                  // Get correct distance from distance matrix
8              distance = d_{iA(v)}
9              if t ≥ T(v) + h + distance then                   // Check if enough time has elapsed
10                 V_i(t) = V_i(t) ∪ v
11             end
12             else if i = A(x) then                             // Record invisible transactions
13                 I_i(t) = I_i(t) ∪ v
14             end
15         end
16     end
17 end
```

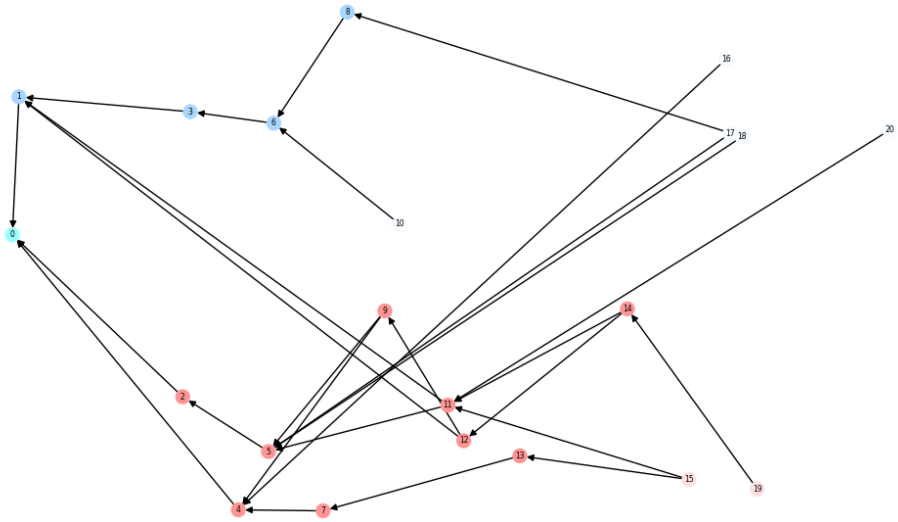

Figure 4.1: An illustration to help explaining the logic behind valid tips selection (partly connected agents).

This illustrates the complexity of having multiple agents with different views on their own sub-Tangle, and means that an appropriate algorithm had to be carefully developed. It is implemented in `get_valid_tips_multiple_agents()` and presented as Algorithm 3.

Algorithm 3: Algorithm to determine $T_i(t)$

valid_tips $(V_i(t), I_i(t))$

Input : All visible transactions $V_i(t)$, all invisible transactions $I_i(t)$, for agent i

Output: All valid tips $T_i(t)$, for agent i

$T_i(t) = \emptyset$ // Empty set
foreach $v \in V_i(t)$ **do** // For every visible transaction
  if $\deg_{\text{in}}(v) = 0$ **then** // Transaction has no approvers
    $T_i(t) = T_i(t) \cup v$
  else if $approvers(v) \subseteq I_i(t)$ **then** // All approvers are invisible
    $T_i(t) = T_i(t) \cup v$
  end
end

In the code, we check the last condition by the truth value of:

```
set(list(self.DG.predecessors(transaction))).
issubset(set(self.not_visible_transactions))
```

4.5 Implementation of the tip selection algorithms

In previous chapters the intuition behind the tip selection algorithm as one of the most important parts of the IOTA consensus mechanism has been explained. In the simulation framework build as part of this dissertation, three different tip selection algorithms have been implemented:

1. A uniform random choice (URTS): two tips are chosen from the list of available tips in a random fashion (uniform distribution).

2. An unweighted random walk: two random walks from the genesis to the available tips with uniform transition probabilities (randomly choosing the next step of the walk).

3. A weighted random walk (MCMC): two random walks from the genesis to the available tips with transition probabilities biased towards approvers with larger cumulative weights ("heavier" transactions are more likely to be selected as next step of the walk).

The reason to start with the uniform random choice is that it is easy to implement and that IOTA research has quantitative predictions for it. In other words, [47] state that "we still consider this model since it is simple to analyse, and may provide insight into the system's behaviour for more complicated tip selection strategies". So this serves as a sanity check and we can use it to reproduce predictions from the white paper. Moreover, it is a close relative of the unweighted random walk. This algorithm has been implemented as a further sanity check, because we expect it and the weighted random walk to have exactly the same behaviour in the case of $\alpha = 0$, because in this case the formula for the transition probabilities just returns equal probabilities (we will illustrate this later).

4.5.1 Uniform random choice

This algorithm is simple and works as shown in Figure 4.2. In the simulation framework it is implemented as `random_selection()`. We determine $V_i(t)$ and $T_i(t)$ and then choose two random tips from $T_i(t)$. If by chance the same two tips are selected, only one edge is added to the DAG object. Again, the `NetworkX` library proves useful, and allows us to attach to tips conveniently with:

```
self.DG.add_edge(transaction,tip)
```

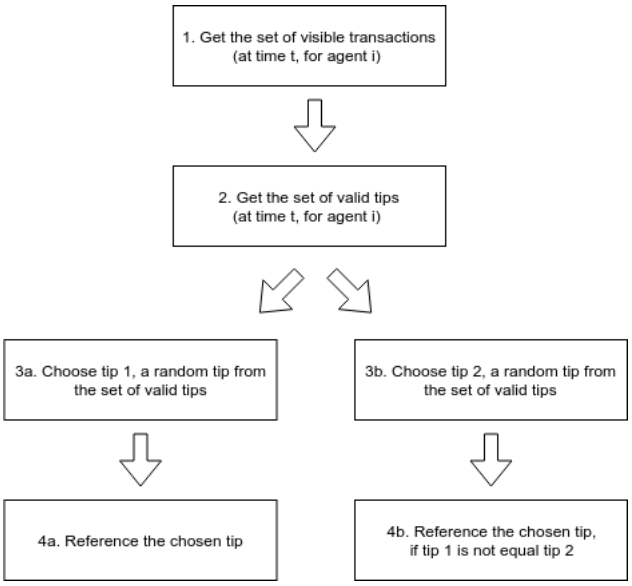

Figure 4.2: The uniform random tip selection algorithm.

4.5.2 Unweighted random walk

The `unweighted_random_walk()` function implements the procedure illustrated in Algorithm 4, the unweighted random walk. Its input parameter is the current (incoming) transaction v and it returns an appropriate tip w – the end point of a random walk from the genesis G to the valid tips. First, the walker particle is initialised as the genesis transaction object. We then need to check if the genesis is the only valid tip, since then the incoming transaction has no other option than to simply reference the genesis. If this is not the case, the random walk is implemented as while loop, with the condition that a valid tip has not been reached. In each iteration of the loop, the walker's direct approvers are determined with the help of the `NetworkX predecessors()` function. Since the issuing agent i of the current transaction only knows the transactions in $V_i(t)$, the walker can just walk on these visible transactions and hence we have to eliminate all approvers that are not visible (from the set of direct approvers). Finally, we can randomly choose one of the approvers for the walker to walk onto and one iteration is finished. In other words, our *transition probabilities* (probabilities to walk from the current position of the walker onto one of its approvers) are following a uniform distribution. In case the walker has reached a valid tip, we can stop walking and return this tip to be approved by v.

This procedure is run twice, since we want to find two valid tips to approve. Note that in the Python implementation $V_i(t)$ and $T_i(t)$ are determined only once outside the `unweighted_random_walk()` function, to reduce running time. The two returned tips are compared and if distinct, we add two edges to the graph. If by chance the same tip got returned twice, we again only need to add one edge to the graph object.

Algorithm 4: Unweighted random walk algorithm

unweighted_walk $(v, V_i(t), T_i(t))$

Input : A transaction v, all visible transactions $V_i(t)$, all valid tips $T_i(t)$, for agent i

Output: A tip w (w for *walker*), that v can reference

$w = \mathrm{G}$ // Start walk at genesis
if $T_i(t) = w$ **then** // If only valid tip is genesis
  return w
end
while $w \notin T_i(t)$ **do** // While walker not valid tip
  $approvers = predecessors(w)$
  $visible\_approvers = approvers \in V_i(t)$
  $w =$ choose randomly one $v \in visible\_approvers$
end
return w

4.5.3 Weighted random walk

This algorithm, implemented as `weighted_random_walk()`, and shown in Algorithm 5, works very similar to the unweighted random walk described before. However, there is a small, but very important difference, which is the core of the IOTA consensus mechanism. The difference is the probabilities vector, with which the next step of the random walk is determined. As explained earlier, this is a *Markov Chain Monte Carlo* (MCMC) technique, since each step in the walk is independent from the previous one, but follows a pre-defined rule, which in this case is the formula

outlined in Equation 2.1 to calculate the transition probabilities. We want transactions with larger cumulative weight to have a larger probability to be chosen as next step of the random walk, and "lighter" transactions to have less chance to be chosen.

In the implementation we use a simplified version of Equation 2.1 (see Appendix B), which has the same desired effect and several others advantages as we will see shortly. The transition probability of the walker to walk from transaction x to y is now given by:

$$P_{xy} = \frac{exp(\alpha W_y)}{\sum_{z:z \rightsquigarrow x} exp(\alpha W_z)} \tag{4.2}$$

where W_y is the cumulative weight of vertex y and $\alpha > 0$ can be chosen. Moreover, $z \rightsquigarrow x$ means that transaction z directly approves transaction x, we therefore sum over all direct approvers of x to make the vector of transition probabilities sum to one. Hence, we do not think about weight differences like in Equation 2.1, but only about the approvers' weights. We can also easily see that this function gives us the effect we want, namely that the probability to walk to y is exponentially increasing with its cumulative weight, and that this effect is moderated by α.

Algorithm 5: Weighted random walk algorithm

weighted_walk $(v, V_i(t), T_i(t), \alpha)$

Input : A transaction v, all visible transactions $V_i(t)$, all valid tips $T_i(t)$, for agent i, α

Output: A tip w (w for *walker*), that v can reference

$w = \mathrm{G}$ // Start walk at genesis
if $T_i(t) = w$ **then** // If only valid tip is genesis
  return w
end
while $w \notin T_i(t)$ **do** // While walker not valid tip
  $approvers = predecessors(w)$
  $visible\_approvers = approvers \in V_i(t)$
  $transition\_probabilities = transition\_probabilities(\alpha, visible\_approvers, A(v))$
  $w =$ choose one transaction $\in visible\_approvers$ according to $transition\_probabilities$
end
return w

In the simulation framework, the function `calc_transition_probabilities_multiple_agents()` implements the calculation of the transition probabilities. However, for an implementation in code an important issue with regard to numerical precision has to be discussed:

A term of the form $exp(\alpha W_y)$ could become very large, and thus exit the range of floating point precision. This can be fixed by normalising the fraction by $exp(-\alpha W^x_{max})$, where $W^x_{max} \equiv \max_{z:z \rightsquigarrow x} W_z$, i.e. the largest cumulative weight value of all approvers. In other words, we normalise by subtracting the maximum cumulative weight (compare to Equation 4.2), making all values in the exponent ≤ 0:

$$P_{xy} = \frac{exp(\alpha\,(W_y - W^x_{max}))}{\sum_{z:z \rightsquigarrow x} exp(\alpha\,(W_z - W^x_{max}))} \tag{4.3}$$

This formula is equivalent to the original one and numerically sound. Therefore, we can implement it in Python code without issues and expect `calc_transition_probabilities_multiple_agents()`

to return a vector of probabilities summing up to one.

As for the unweighted walk algorithm before, we need to make sure to only consider approvers, which are currently in $V_i(t)$ for the issuing agent i during the execution of the while loop. The difference to Algorithm 4 is that we get probabilities in the form of:

```
transition_probabilities = [0.09,0.56,0.35]
```

Finally, the next step for the walker is decided by:

```
walker = np.random.choice(visible_approvers, p=transition_probabilities)
```

4.5.4 Calculation of cumulative weights

As explained in Chapter 2, the cumulative weight W_v of a transaction v is one (the transaction's "own weight") plus the number of all transactions directly of indirectly referencing it. In the description of the main simulation algorithm we see that in each iteration of the simulation loop we need to update cumulative weights after the tip selection is done, since the increase in cumulative weights influences the tip selection in the next iteration. The cumulative weight is implemented as an integer for every `Transaction` object and initialised to one. However, with a multiple agent model, we need to record a cumulative weight value $W_i(v)$ for each agent i separately, since the agents see their own sub-Tangles. Hence, only the visible transactions can be included when calculating cumulative weights.

Python has a convenient data structure, called `defaultdict`, which is used in the code to allow for separate cumulative weight values per agent. The difference to a normal dictionary is that we can specify a default value, in this case one, for a key (agent) encountered for the first time. This is more flexible for updating the cumulative weights. Thus, every `Transaction` has:

```
self.cum_weight_multiple_agents  = defaultdict(lambda: 1)
```

Algorithm 6: Algorithm to update the cumulative weights

update_cumulative_weights (v)

Input : A transaction v, for which tip selection has just been done;
all transactions $V(t)$ globally available, as well as $V_i(t)$, for all $i \in A$, where $t = at_v$

Output: An updated cumulative weight $W_i(d)$ for each descendant d of v, for every agent $i \in A$

```
1 foreach d ∈ descendants(v) do                    // For every descendant
2     foreach i ∈ A do                             // For every agent
3         if d ∈ V_i(t) then                       // If descendant is visible for the agent
4             W_i(d) = W_i(d) + 1
5     end
6 end
```

The update function is implemented as `update_weights_multiple_agents()` and illustrated in Algorithm 6. We update the cumulative weights of all transactions directly or indirectly referenced by transaction v (which we call descendants) for each agent i such that for each descendant d we have $W_i(d) = W_i(d) + 1$ if d is in $V_i(t)$.

By only dealing with the descendants we make the algorithm faster then looping over the whole Tangle and doing a full update off all cumulative weights. However, as will be explained later on in the challenges section, this algorithm still represents the largest performance bottleneck, since with growing Tangles more and more transactions will be referenced by an incoming transactions and thus only updating its descendants is almost the same as updating the whole Tangle.

4.6 Implementation of additional Tangle algorithms

These algorithms are not used in the Tangle creation itself, i.e. in the `setup()` and `run()` functions, but in measurements and evaluations of the Tangle data and behaviour.

4.6.1 Calculation of exit probabilities

Exit probabilities have not been dealt with so far, since we do not use them in the main simulation loop to create a Tangle. However, they are used to calculate confirmation confidences and thus play an important role in the simulation framework. We mentioned in Chapter 2 that there are multiple ways of calculating a transaction's confirmation confidence and outlined the idea from [47], i.e. to run the tip-selection algorithm many times, e.g. 100 times and then calculate the proportion of how many selected tips directly or indirectly approve a specific transaction, e.g. 95 out of 100 means 95% confirmation confidence for that transaction. This approach has two drawbacks with regard to the implementation in code. First, it is computationally demanding to run 100 random walks. Second, it is imprecise, due to the random nature of the walks.

Using exit probabilities we can solve both problems. When we have a tip set $T_i(t)$, we define the exit probability distribution $f_i : T_i(t) \to [0, 1]$ to be the probability distribution induced by our tip selection algorithm [25]. This means that $f_i(v)$ is the probability that a newly incoming transaction issued by agent i would select v during tip selection. We know that our random walk tip selection algorithms are based on transition probabilities. Hence, we can use these probabilities to determine the chance of a random walk ending on a specific tip, or in general passing through any transaction in the Tangle. The following picture serves as an illustration:

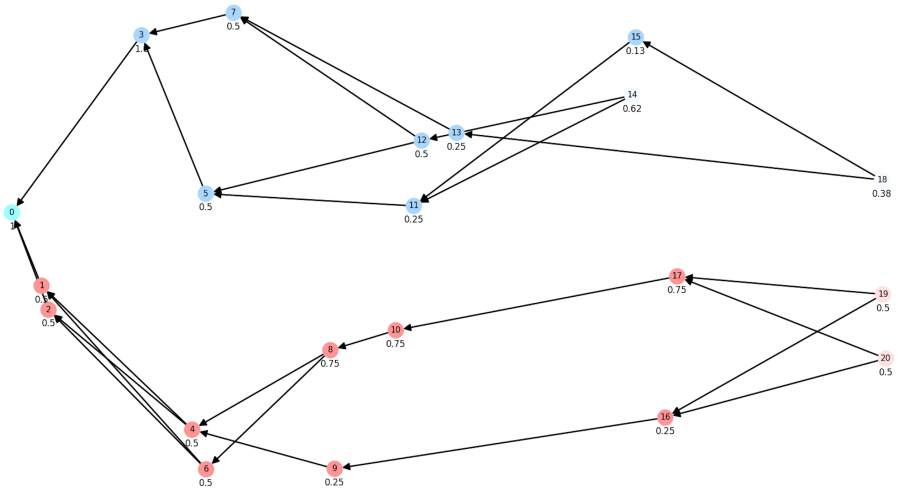

Figure 4.3: An illustration of exit probabilities, for a two-agent Tangle with $d = \infty$, i.e. two completely disconnected agents. An unweighted walk is used, transition probabilities are equal.

We see two agents, a blue and a red one. The agents just see their own sub-Tangles. If we assume a new transaction 21 comes in and is issued by the blue agent, an unweighted random walk would end with 62% certainty on tip 14 and with 38% certainty on tip 18. For the red agent, the walk would end with 50% certainty on tip 19 and with 50% certainty on tip 20. These exit probabilities can be easily confirmed by following the possible paths from the genesis to the tips and multiplying the

uniform transition probabilities of each step with the exit probability of the current transaction. For instance, transaction 3 has an exit probability (or rather probability of walking to this transaction) of 100% for the blue agent. From there, a walker could either walk to transaction 5 or 7, so the transition probabilities are [50%,50%], as well as the exit probabilities of transactions 5 and 7.

Algorithm 7: Algorithm to calculate the exit probabilities

exit_probabilities ()

Input : All transactions $V(t)$, h $(= 1)$, d globally available

Output: Exit probability $P_i(v)$ for every transaction $v \in V(t)$, for every agent $i \in A$

foreach $i \in A$ **do** // For every agent
  // Get set of visible transactions, time is arrival time of last transaction plus latency
  $V_i(t) = visible\_transactions(t, h, d, i)$, where $t = at_x + h$
  $P_i(\mathrm{G}) = 1$ // Genesis is start, initialise to 100%
  $P_i(v) = 0$ for all $v \in V(t) \backslash \{\mathrm{G}\}$ // Initialise all others to 0%
end
$sorted = reverse\_topological\_sort(V(t))$
foreach $v \in sorted$ **do** // For every transaction in sorted
  foreach $i \in A$ **do** // For every agent
    if $v \in V_i(t)$ **then** // If transaction is visible for the agent
      $approvers = predecessors(v)$
      $visible\_approvers = approvers \in V_i(t)$
      $transition\_probabilities = transition\_probabilities(visible\_approvers, i)$
      foreach $va \in visible\_approvers$ **do** // For every visible approver
        $P_i(va) = P_i(va) + (P_i(v) * transition\_probabilities[va])$
      end
  end
end

In the following, Algorithm 7 for `calc_exit_probabilities_multiple_agents()` is explained. Similar to cumulative weights, we use $P_i(v)$ to denote the exit probability of a transaction v for agent i. Note that we again deal with the visibility issue and need to make sure to include only the agents' visible transactions in our calculations. To store the values we use:

```
self.exit_probability_multiple_agents  = defaultdict(lambda: 0)
```

This means that at the start every exit probability is zero by default, except for the genesis, which we have to initialise to one. We also need to get the set of visible transactions for each agent. Note that we do this after the Tangle has been created, so the "current" time is the arrival time of the very last transaction plus the latency (h) to include all transactions in the calculations. After that, we sort the transactions in $V(t)$ in reverse topological order, since we want to start the calculations with the genesis transaction. The tips are in the end. Now a loop iterates over all sorted transactions and does the following for every agent: If a transaction v is visible for the agent, all direct approvers of the transaction are determined and the transition probabilities of walking to the visible approvers are calculated (exactly the same as for the weighted random walk algorithm). Then, the exit probability of every visible approver is updated by adding the exit probability of

transaction v times the transition probability of walking to the approver. As result of running this algorithm we know the probability of a random walk ending at any transaction, or more importantly ending at the tips, which is what we need to calculate confirmation confidences.

4.6.2 Calculation of confirmation confidence

This sections explains how exit probabilities can be used to calculate confirmation confidences, which is both computationally cheaper and more precise than running many random walks to see which tips reference a specific transaction. It might be helpful to briefly summarize the main idea of confirmation confidence in the IOTA protocol.

Essentially, confirmation confidence for a transaction v measures how many "important" tips are referencing v. Important means that tips are more likely to be selected during tip selection because they are on a path with high cumulative weights. So if many such tips reference v, we can assume that v is very likely to be deep in the main, growing branch of the Tangle and is unlikely to be orphaned.

Thanks to our calculation of exit probabilities we know exactly the chance with which any tip is selected during the next round of tip selection. So if we now sum up all exit probabilities of tips referencing v, we can measure how confident we are that v is to stay in the growing part of the Tangle. For example, in Figure 4.3 transaction 13 has a confirmation confidence of 100% (tips 16 and 19 both referencing it) for the red agent. The blue one does not know about transaction 13 at all, which we treat as 0% confirmation confidence.

We can write the confirmation confidence $c_i(v)$ formally as [25]:

$$c_i^{(t)}(v) \equiv \sum_{z \in T_i(t) \cap \mathcal{F}_i^{(t)}(v)} P_i(z) \tag{4.4}$$

where $P_i(z)$ is the exit probability of transaction z for agent i, $v \in V_i(t)$, $T_i(t)$ is a tip set, and $\mathcal{F}_i^{(t)}(v)$ is the future set of v, defined as $\mathcal{F}_i^{(t)}(v) \equiv \{z \in V_i(t) | z \text{ references } v\}$, i.e. all visible transactions that reference v. Since the exit probabilities of the tips for an agent always sum up to 100%, it is given that $c_i^{(t)}(v)$ must be between 0% and 100%. As mentioned above, if a transaction v is not in $V_i(t)$, we would record a confirmation confidence of 0%, that is $v \notin V_i(t) \Rightarrow c_i^{(t)}(v) = 0$.

This calculation has been implemented in `calc_confirmation_confidence_multiple_agents()` and is illustrated in Algorithm 8. Again, a `defaultdict` with default value 0 is used. Since every agent has his own view on the Tangle, we need to calculate confirmation confidences for every agent separately. The outer loop iterates over all agents and gets the sets of visible transactions and valid tips. As in the previous algorithm, this is done after the Tangle creation has been completed, and the arrival time of the last transaction plus latency is used as "current" time. Subsequently, we iterate over the visible transactions and for each transaction sum the exit probabilities of all tips referencing it. There is a convenient and optimised `NetworkX` function for this purpose, i.e. checking for a path between a tip and a transaction:

```
nx.has_path(self.DG,tip,transaction)
```

4.6.3 Calculation of attachment probabilities

We analyse attachment probabilities in the next Chapter and explain its calculation here. An attachment probability tells us the chance with which a new incoming transaction selects the tips issued by either agent in the Tangle. We call all current tips issued by agent i agent i's *branch* – not to confused with agent i's sub-Tangle, which is defined by visibility. There are two options to determine these, which is very similar to the concept of calculating the confirmation confidences explained

Algorithm 8: Algorithm to calculate the confirmation confidences

confirmation_confidence()

Input : All transactions $V(t)$, h $(= 1)$, d globally available

Output: Confirmation confidence $c_i(v)$ for every transaction $v \in V_i(t)$ for every agent $i \in A$

foreach $i \in A$ **do** // For every agent
  // Get set of visible transactions, time is arrival time of last transaction plus latency
  $V_i(t), I_i(t) = visible\_transactions(t, h, d, i)$, where $t = at_x + h$
  $T_i(t) = valid\_tips(V_i(t), I_i(t))$
  foreach $v \in V_i(t)$ **do** // For every visible transaction
    foreach $t \in T_i(t)$ **do** // For every valid tip
      if $t \cap \mathcal{F}_i^{(t)}(v)$ **then** // If tip references transaction
        $c_i(v) = c_i(v) + P_i(t)$
    end
  end
end

Algorithm 9: Algorithm to calculate attachment probabilities

attachment_probabilities ()

Input : All transactions $V(t)$ globally available, as well as $T_i(t)$ for all $i \in A$, where $t = at_x + h$ (arrival time of last transaction plus latency), and number of agents N

Output: An attachment probability AP_i for every agent $i \in A$

foreach $i \in A$ **do** // For every agent
  $sum = 0$
  foreach $v \in T_i(t)$ **do** // For every tip
    if $A(v) = i$ **then** // If tip was issued by the agent
      $sum \mathrel{+}= P_i(v)$
  end
  foreach $j \in A \setminus \{i\}$ **do** // For every agent other than in outer loop
    foreach $v \in T_j(t)$ **do** // For every tip of the other agent
      if $A(v) = i$ **then** // If tip was issued by the agent in outer loop
        $sum \mathrel{+}= P_j(v)$
    end
  end
  $AP_i = sum/N$
end

above. First, we could run a large number of random weighted walks, for example for two agents 100 runs started by the first agent and 100 runs started by the second agent, and for these 200 returned tips calculate the proportion of tips ending up on each branch. This gives us an estimate how likely a new incoming transaction is to select either branch in its tip selection algorithm, when the random walk is used. However, we mentioned that this is very computationally inefficient and not 100% precise (for more precision we would need a very large number of random walks). Fortunately we have an alternative using exit probabilities, analogical to the calculation of confirmation confidences.

The exit probabilities of tips are used to get a more precise measure of the likelihood of attaching to a particular agents' branch. For each agent we calculate a probability to attach to the transactions he issued, called attachment probability AP_i. To calculate these for an agent, we sum all the exit probabilities of the current tips this agent issued. In Figure 4.4 for example, we sum exit probabilities of all agents for the light-blue tips to calculate AP_i for the main-Tangle agent (and similarly for the other agent). In the end, the summed probabilities are divided by the number of agents N to get a total of 100%. See Algorithm 9 for an illustration.

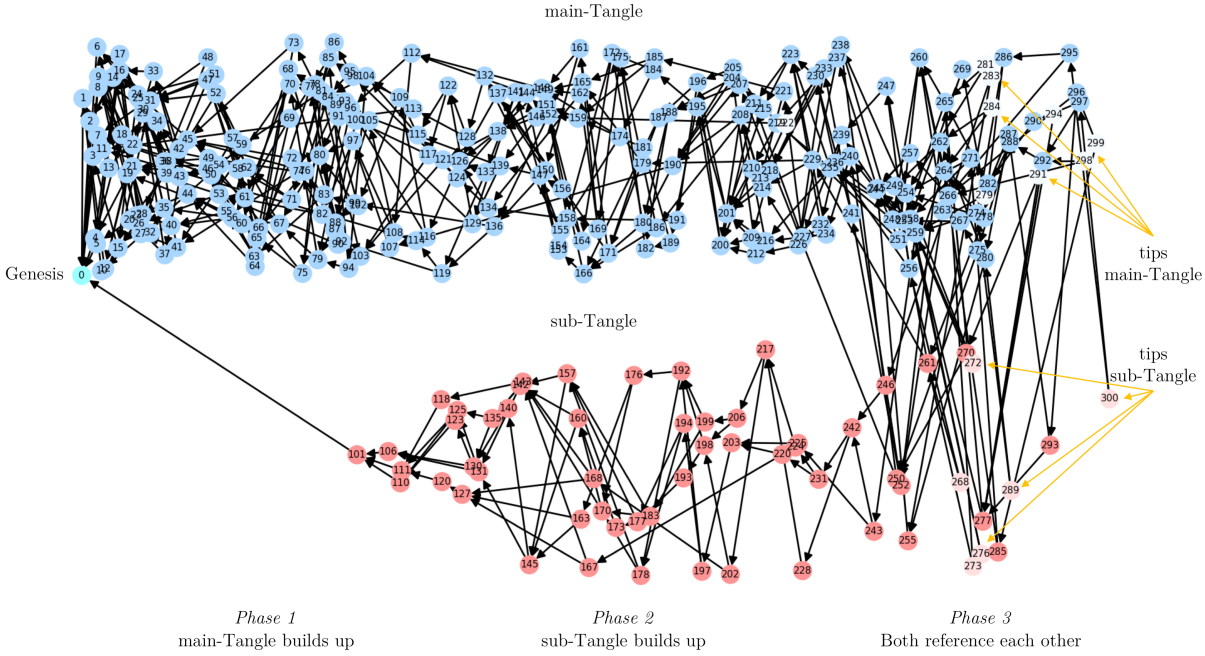

Figure 4.4: An illustration of a main-Tangle branch and a sub-Tangle branch for a multi-agent model. The blue transactions are referred to as main-Tangle branch, and the red ones as sub-Tangle branch.

4.7 Additional features: parameter changes, Tangle visualisations and graphs, and multi-processing

4.7.1 Parameter changes

There are a few additional features of the simulation framework, briefly mentioned before. First, the user can provide a `config.ini` file to specify simulation parameters and *parameter change events*. In the beginning of this chapter it was explained why a user might want to trigger such change events during the execution of the `run()` function. In the following the implementation, as well as usage will be explained.

The standard Python module `configparser` is used to provide a structured way of reading simulation parameters from a `config.ini` file. If a user provides such a file as a command line argument,

two effects can be observed: First, all instances of `class Multi_Agent_Simulation` will be initialised with the given parameters. The `config.ini` file should have a section `[PARAMETERS]` and all standard simulation parameters must be provided in the form of `no_of_agents = 2`, for instance.

The second effect is the triggering of change events, which are triggered for a given *step*, the `transaction.id` of an incoming transaction. Each change event is contained in a separate section of the configuration file, for example `[EVENT1]`. As explained earlier, two parameters can be changed: The distance matrix d, and the agent choice probabilities c. Note that the user has in general two choices to specify the distance matrix d: Either provide only one parameter, for instance `distance = 100`, so that a distance matrix (list of lists in Python) with this value as distance between all agents is created automatically. For $N = 2$ agents this would look like `[[0,100],[100,0]]`. The other option is that the user specifies a custom distance matrix with different distance between agents. For $N = 3$ agents this could look like `[[0,2,3],[2,0,1],[3,1,0]]`. Note that we assume the correctness of this matrix, i.e. that it is symmetric and contains the distances of the shortest paths between the agents, as explained in Chapter 2.

For the agent choice probabilities c the user can provide a list of probabilities, which of course need to sum up to 100% and have to match the number of N agents in the simulation. However, if the user specifies `agent_choice = None`, we assume the default uniform distribution and an appropriate probability vector is created automatically.

4.7.2 Visualisations

The second additional feature to mention are the visualisation functions. Visualisations have been both very helpful during development to do sanity checks for the Tangle algorithms, as well as for producing appropriate graphs for later chapters of this dissertation. The exact implementation is not of much interest here, but `NetworkX`'s drawing interface using `Matplotlib` has been a useful feature to illustrate the created Tangles. The user can use different methods to create illustrations:

- `print_graph()`: Creates an illustration of the DAG, with different colours for the sub-Tangles.
- `print_tips_over_time()`: Plots the total number of tips in the Tangle over time.
- `print_tips_over_time_multiple_agents()`: Plots the number of tips in the Tangle over time for each agent separately.
- `print_attachment_probabilities()`: Plots the attachment probabilities over time for each agent separately.

4.7.3 Multi-processing

The structure of every created Tangle is highly arbitrary. This is because it is based on randomly sampled data (the inter-arrival times), as well on random choices or random walk algorithms for tip selection, i.e. to decide how the Tangle grows. This is what makes it difficult to compare Tangles with identical parameter sets. Hence, to enable better comparison and more accurate analysis, one should always create a set of Tangles with the same parameters and then average the measures of interest. To make this more computationally efficient (run several simulations in parallel to optimise CPU usage), a separate script has been created, which uses the Python `multiprocessing` package. This script can run any desired number of simulations with the same parameters in parallel and collects the Tangle data. This can significantly reduce running time (see Appendix C).

4.8 Challenges

In this section the main challenges of the development of the simulation framework are highlighted. Whereas it was conceptually challenging to implement the multi-agent model in the code, especially with regard to the visibility of the different sub-Tangles, a much more important aspect is the efficient implementation of the previously presented algorithms.

4.8.1 Efficiency

The directed acyclic graph structure combined with random walk algorithms make the Tangle a complex mathematical object. As we have seen in most implementation notes above, Tangle operations are complex and therefore efficient algorithms are needed for calculations on the graph. As `NetworkX` provides the main data structure of the simulation framework, we can fortunately use many efficiently implemented algorithms, for example for finding a path between two transactions, or to get all descendants of a transactions [44]. However, when Tangles with large number of transactions are simulated, the graph operations take more time. We used the Python module `cProfile` to gather profiling statistics from the execution of large scale simulations (in particular the core Tangle creation, i.e. the `setup()` and `run()` methods). As we can see in Figure 4.5, the most time is taken by the update of cumulative weights. For a Tangle consisting of 5,000 transactions, this function takes 61.81% of the total running time of the simulation.

The algorithm we use updates all descendants of an incoming transaction by adding one to its cumulative weight, since this is less computational effort than updating a whole Tangle. We also see that the `NetworkX` function `nx.descendants` has a relatively small impact and accounts for only 7.22% of the 61.81%. Thus, the main driver of the running time of a simulation is the number of descendants iterated over. The question is whether in large Tangles most old transactions are descendants of an incoming transaction. Therefore, we plot the ratio of descendants to the total number of transactions in the Tangle over time, this time for 10,000 transactions (Figure 4.6). 5,000 or 10,000 transactions will be used as Tangle size for most simulation experiments in later chapters, just as [35].

```
Parameters:  Transactions = 5000,  Tip-Selection = WEIGHTED,  Lambda = 50,  Alpha = 0.005 | Simulation started...

Percent:  [##################################################] 100% | Number:  5000 | Simulation completed...
Simulation time: 614.142 seconds

Calculation time confirmation confidence: 0.0 seconds

         451264834 function calls (451251212 primitive calls) in 616.092 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        2    0.000    0.000  616.141  308.070 core.py:1(<module>)
   1469/1    0.039    0.000  616.094  616.094 {built-in method builtins.exec}
        1    0.160    0.160  614.142  614.142 simulation_multi_agent.py:115(run)
     5000  336.331    0.067  380.862    0.076 simulation_multi_agent.py:441(update_weights_multiple_agents)
     5000    0.009    0.000  232.263    0.046 simulation_multi_agent.py:171(tip_selection)
     5000    0.056    0.000  232.254    0.046 simulation_multi_agent.py:391(weighted_MCMC)
    10000   90.186    0.009  111.661    0.011 simulation_multi_agent.py:217(get_visible_transactions)
    10000   22.266    0.002   88.770    0.009 simulation_multi_agent.py:273(get_valid_tips_multiple_agents)
 23735890   47.598    0.000   55.651    0.000 simulation_multi_agent.py:292(all_approvers_not_visible)
     5000    0.827    0.000   44.530    0.009 dag.py:54(descendants)
     5000    0.015    0.000   42.966    0.009 generic.py:149(shortest_path_length)
     5000    1.714    0.000   42.950    0.009 unweighted.py:25(single_source_shortest_path_length)
  8931485    6.717    0.000   41.224    0.000 unweighted.py:69(_single_shortest_path_length)
    10000    3.621    0.000   31.733    0.003 simulation_multi_agent.py:416(weighted_random_walk)
  8938971   14.869    0.000   29.109    0.000 {method 'update' of 'dict' objects}
100030000   19.596    0.000   19.596    0.000 agent.py:12(__str__)
   218278   16.692    0.000   16.885    0.000 helpers.py:35(common_elements)
 48323239   12.765    0.000   16.112    0.000 digraph.py:788(predecessors)
```

Figure 4.5: Profiling statistics from the execution of a 5,000 transaction Tangle simulation with Algorithm 6.

After an initial steep growth phase, the growth slows down, but the ratio seems to keep increasing in the long run. Therefore we can conclude that this update algorithm is better than any algorithm

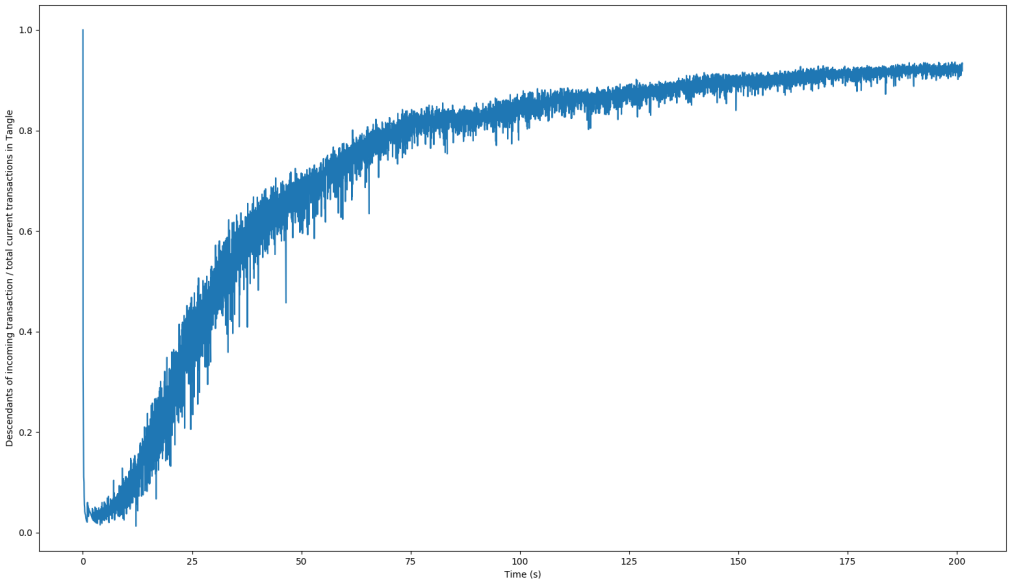

Figure 4.6: The ratio of descendants of an incoming transaction over total current transactions in the Tangle over time.

looping over the whole Tangle for simulations of less then 10,000 transactions (as long as the ratio is below 100%). Moreover, the plot shows large variations – sometimes the ratio gets very low, meaning less work for the update algorithm. The simulations show the same behaviour, instantly getting slower, but once the ratio growth slows down, the time taken to add another transaction to the Tangle stays roughly the same (but keeps increasing in the long run). This is an improvement to an alternative algorithm updating cumulative weights of all transactions, but only for "small simulations". For very large simulations, another algorithm might be more appropriate, in particular we consider a variation of the one proposed by [22], and shown in Algorithm 10. The idea of this algorithm is to generate the future sets of all transactions – as explained earlier, $\mathcal{F}^{(t)}(v)$ is the set of all direct and indirect approvers of a transaction v. The length of this future set plus one (own weight) determines the cumulative weight of a transaction. First, we sort all transactions in the DAG topologically, to make sure that every transaction is preceded by all its approvers (parents). We then iterate over the transactions and update $\mathcal{F}^{(t)}(c)$ of every child c of a transaction by set unions, adding $\mathcal{F}^{(t)}(v)$ and the transaction v itself. Note that in the multi-agent case we only need to include visible transactions in the calculations and every transaction has a separate future set for every agent.

However, for the scale of our simulations, Algorithm 6 is significantly faster (Figure 4.7). Algorithm 10 suffers heavily from the topological sorts (although its an optimized `NetworkX` method) and set unions. To summarize, the update of cumulative weights remains a challenge for an implementation, even with the optimized algorithm we use. It is the performance bottleneck and one of the main drawbacks of the developed simulation framework. Improving the computational efficiency of the simulation framework remains an area for future work. Since we outlined the difficulties with updating cumulative weights here, it is natural and interesting to think about how this issue is dealt with in the IOTA reference implementation (IRI) in practice, and if this might not help us here.

As explained in the IOTA documentation [29] and in [22], the IRI uses an algorithm very similar to Algorithm 10, but deals with a sub-graph only. In fact, the whole tip selection is done on the sub-graph. The sub-graph is a part of the Tangle, not starting from the genesis, but from closer point in the past. This solves the efficiency problem, because we do not deal with an ever

growing Tangle. In the IRI this is made possible by a concept called *milestones*, which are verified checkpoints in the Tangle past and are used to define a sub-graph (see Chapter 2 and 7 for more details). However, this is not part of the mathematical IOTA model used in this dissertation and therefore it is not possible to implement the concept in our simulation framework.

Algorithm 10: An alternative algorithm to update the cumulative weights

update_cumulative_weights (v)

Input : All transactions $V(t)$ globally available, as well as $V_i(t)$, for all $i \in A$, where $t = at_v$

Output: An updated cumulative weight $W_i(v)$ for every $v \in V(t)$, for every agent $i \in A$

1 $sorted = topological\_sort(V(t))$
2 **foreach** $v \in sorted$ **do** // For every transaction in sorted graph
3 **foreach** $i \in A$ **do** // For every agent
4 $F_i^{(t)}(v) = \emptyset$ // Empty set
  // Get all transactions approved by v - always two in the Tangle
5 $childs = successors(v)$
6 **foreach** $c \in childs$ **do** // For every child
7 **if** $c \in V_i(t)$ **then** // If child is visible for the agent
  // Update future set by set union with parent future set and transaction
8 $\mathcal{F}_i^{(t)}(c) = \mathcal{F}_i^{(t)}(c) \cup \mathcal{F}_i^{(t)}(v) \cup \{v\}$
9 **end**
10 $W_i(v) = \left|\mathcal{F}_i^{(t)}(v)\right| + 1$ // Cumulative weight is size of future set plus own weight
11 **end**
12 **end**

```
Parameters:  Transactions = 5000,  Tip-Selection = WEIGHTED,  Lambda = 50,  Alpha = 0.005 | Simulation started...

Percent:  [##################################################] 100% | Number:  5000 | Simulation completed...
Simulation time: 614.142 seconds

Calculation time confirmation confidence: 0.0 seconds

         451264834 function calls (451251212 primitive calls) in 616.092 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        2    0.000    0.000  616.141  308.070 core.py:1(<module>)
   1469/1    0.039    0.000  616.094  616.094 {built-in method builtins.exec}
        1    0.160    0.160  614.142  614.142 simulation_multi_agent.py:115(run)
     5000  336.331    0.067  380.862    0.076 simulation_multi_agent.py:441(update_weights_multiple_agents)
     5000    0.009    0.000  232.263    0.046 simulation_multi_agent.py:171(tip_selection)
     5000    0.056    0.000  232.254    0.046 simulation_multi_agent.py:391(weighted_MCMC)
    10000   90.186    0.009  111.661    0.011 simulation_multi_agent.py:217(get_visible_transactions)
    10000   22.266    0.002   88.770    0.009 simulation_multi_agent.py:273(get_valid_tips_multiple_agents)
 23735890   47.598    0.000   55.651    0.000 simulation_multi_agent.py:292(all_approvers_not_visible)
     5000    0.827    0.000   44.530    0.009 dag.py:54(descendants)
     5000    0.015    0.000   42.966    0.009 generic.py:149(shortest_path_length)
     5000    1.714    0.000   42.950    0.009 unweighted.py:25(single_source_shortest_path_length)
  8931485    6.717    0.000   41.224    0.000 unweighted.py:69(_single_shortest_path_length)
    10000    3.621    0.000   31.733    0.003 simulation_multi_agent.py:416(weighted_random_walk)
  8938971   14.869    0.000   29.109    0.000 {method 'update' of 'dict' objects}
100030000   19.596    0.000   19.596    0.000 agent.py:12(__str__)
   218278   16.692    0.000   16.885    0.000 helpers.py:35(common_elements)
 48323239   12.765    0.000   16.112    0.000 digraph.py:788(predecessors)
```

Figure 4.7: Profiling statistics from the execution of a 5,000 transaction Tangle simulation with Algorithm 10.

4.8.2 Other challenges

The conceptually challenging part of the simulation framework was to deal with the different sub-Tangles and to make sure that each agent at any time only deals with his Tangle version. This meant that the function `get_visible_transactions()` plays a central role for most algorithms, and at each time the correct $V_i(t)$ has be to be used, determined with the correct t. This is why we for example need to pass in $av_x + h$ (arrival time of last transaction plus latency) for algorithms running after Tangle creation, to make sure to include all transactions. Moreover, for all walks on the Tangle we had to make sure to only walk on the visible sections and to include only visible transactions to calculate transition probabilities (among other calculations). Another difficulty has been highlighted for the implementation of the `get_valid_tips_multiple_agents()` function in Figure 4.1, since it was not only sufficient to check for all unapproved transactions, but also to check if for all approved transactions all approvers (returned by a `NetworkX` function) are invisible. Lastly, multiple agents also means that we need to record all transaction variables (exit probability, cumulative weight, confirmation confidence) for every agent separately, which has been solved by the use of `defaultdict`.

In summary, the conceptual challenges have been solved and an appropriate framework to simulate multi-agent Tangles has been created. Performance remains an issue, but should be sufficient for our purposes. In the following chapters we will use the framework to analyse specific scenarios.

Chapter 5

Simulations: The two-agent case

This chapter deals with simulation results and analysis of two-agent Tangles. This is the minimal model in a multi-agent framework, and thus a good starting point to dive deeper into the context. Also, the distance matrix d becomes a single number and thus reduces parameter space and complexity. One of the main research questions examinable with the help of the simulations is to analyse what happens when two agents are very distant from each other (for a period of time). In particular, we will examine tip count stability and attachment probabilities for both agents.

5.1 Parameter choices

5.1.1 Interaction of transaction rate, time and alpha in simulations

Simulations of multi-agent networks are dependent on many parameters, which all have an effect on how the structure of the simulated Tangle looks like. Hence, it is important to make reasonable assumptions and choices and some of the parameter choices are discussed in the following. This section deals with two-agent Tangles, $N = 2$, and as explained earlier, we assume $h = 1$ to be fixed throughout all simulations. In other words, we assume that the difficulty of POW is fixed (POW takes a constant amount of time). The unit of time is arbitrary for our framework, so we could decide to measure it in seconds. Note that λ is measured in $1/h$, such that λ has the meaning "transactions per second". By default c is uniformly distributed, unless otherwise stated. The distance matrix d will depend on the specific scenario we are analysing, so this will be discussed later on. We are left with λ, the average incoming transactions flow, and α, the bias parameter.

The rate of incoming transactions per second λ should be chosen together with the size of the simulated Tangle, since these two measures together define the time horizon of a simulation, i.e. the total time is given by dividing the number of transactions by λ. For example, if $\lambda = 50$, 50 new transactions will be issued per second, and so the time horizon of a 50 transaction Tangle would be one second. Since $h = 1$ second, a "blowball" Tangle with only tips referencing the genesis would be created (Figure 5.2). The other extreme would be a very small λ, producing a large time horizon and as result a chain of transactions (Figure 5.3). Both of these transaction structures are not DAG-like and thus for a given number of transactions an appropriate λ should be chosen. Additionally, if a large Tangle is simulated, one could play around with different λ values to model different "stress-levels".

The last parameter not yet discussed, α, is the most important parameter regulating the Tangle creation and structure. As we have seen in the implementation section, it comes into play when the transition probabilities of the random walk are calculated (Equation 4.2) and the higher α is, the more importance is placed on high cumulative weights during the walk. A high alpha would for example result in many unreferenced (left behind) tips and a main chain of "heavy" transactions (Figure 5.4). In this illustration we can observe a "chain of blowballs", occurring every second, since within the invisible period caused by h all incoming transaction attach to just one transaction – the one with the highest cumulative weight. The right level of alpha is so important, because

the Tangle should support high confirmation rates, which is not the case in a chain-like Tangle structure with most transactions being left behind. Remember that a weighted random walk with $\alpha = 0$ is equivalent to the unweighted random walk, which is also not desired because it makes the Tangle more vulnerable to double spends and laziness (see Chapter 2 and [47]). Hence, the level of α for our simulations should be neither too high, nor too low, but earlier work has already attempted to find an "ideal level" [24], by quantifying the "probability of being left behind" (POBLB).

This measures the chance of becoming a permanent tip and can intuitively be expressed in terms of confirmation confidence: "1 - POBLB equals confirmation confidence", since a 100% probability of never being referenced should result in a confirmation confidence of 0%. Simulation results haven shown that "if one wants to keep percentage of transaction left behind to be constant, then as λ grows α has to be decreased" [24]. The following Figure 5.1 from their paper shows the POBLB as a function of α and λ and thus one can choose these values according to which confirmation confidence one is interested in. The information from the illustration thus serves as an upper bound for the α value, i.e. by defining which value would make the confirmation confidence unacceptable.

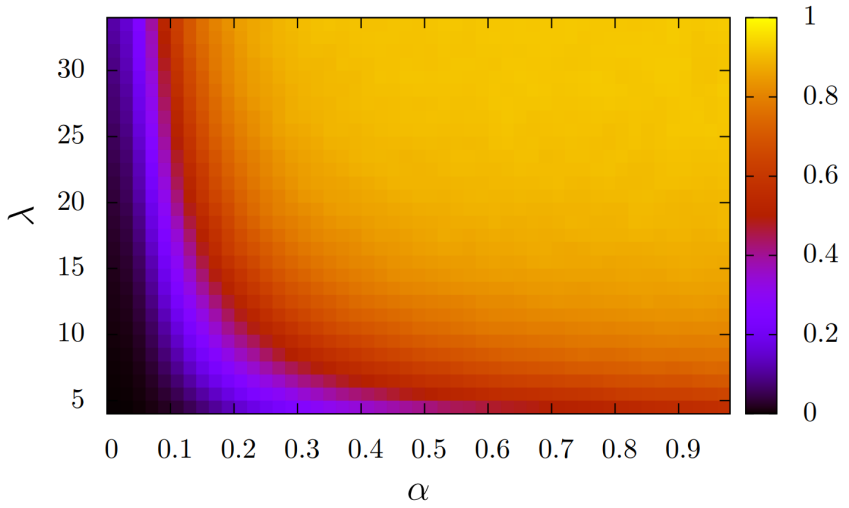

Figure 5.1: POBLB (or confirmation confidence) as a function of α and λ for $\alpha \in [0, 1]$, from [24].

In most of our simulations we use $\lambda = 25$ or $\lambda = 50$, since the number of transactions in the Tangles will be at least 5,000. This in turn means that if we do not want to leave too many transactions behind, α should be very small, and we will mostly pick values between 0.1 and 0.001.

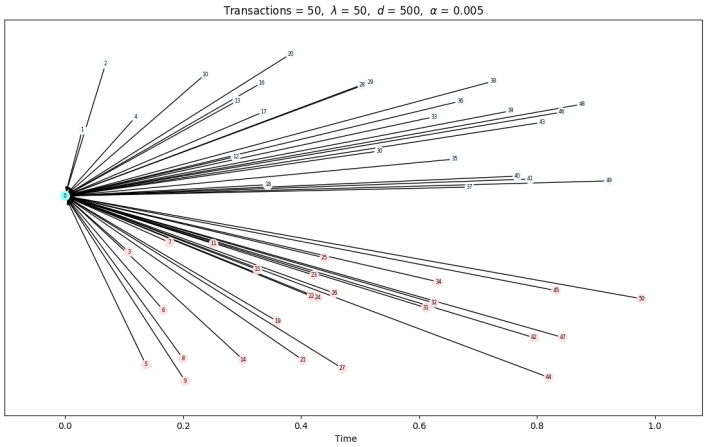

Figure 5.2: Illustration of a blowball Tangle caused by a high λ value.

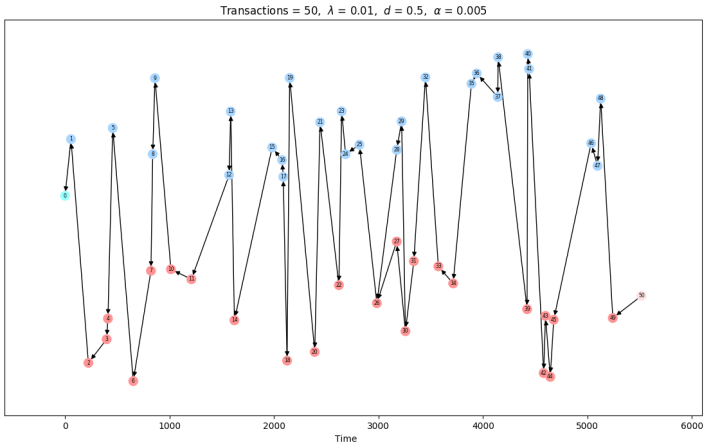

Figure 5.3: Illustration of a chain Tangle caused by a low λ value.

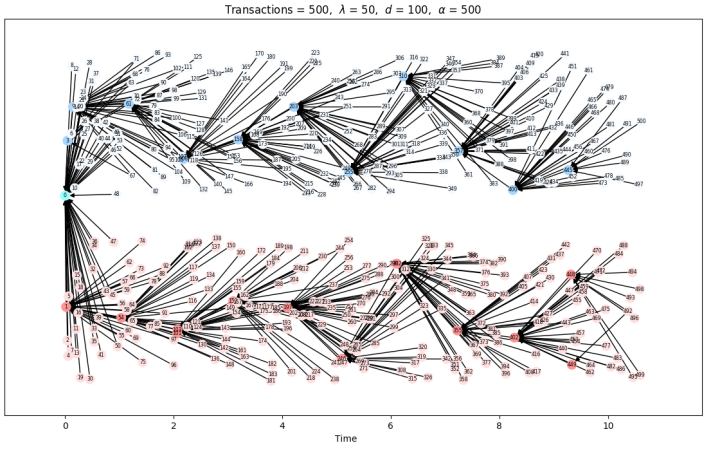

Figure 5.4: Illustration of a "chain of blowballs" Tangle caused by a high α value.

5.1.2 Simulation length

We addressed the problem of running large scale simulations in the previous chapter, but showed that we can run simulations with a total number of transactions below 10,000 transactions in a reasonable amount of time. This is due to an optimized algorithm for cumulative weights updates. If we simulate Tangles with a tip selection mechanism without any use of cumulative weights, i.e. the uniform random tip selection (URTS), we could omit the weight updates completely and easily create larger Tangles (10,000+). This is helpful because URTS will be used in the next section as a baseline and sanity check, and previous work has shown [24] and stressed [47] the usefulness of this approach:

> This strategy is the easiest to analyse, and therefore may provide some insight into the qualitative and quantitative behaviour of the Tangle.

The important question to ask is how long our simulations have to be in order to have meaningful results. Both [47, 24] show that Tangles in general undergo an "adaptation period". We explain the importance of stability in the context of the number of tips in more detail in the next section, but generally, when a Tangle builds up and we measure a certain quantity, this measure should be stable for different Tangle sizes (keeping the other parameters fixed). So it is a good idea to assure that a simulation is not "too short" and while this is difficult to analyse formally we can point out some rough, rather intuitive guidelines (thanks to Alon Gal for a discussion on this).

First, the Tangle should have many "layers". In our model transactions are arriving with a rate of λ per h, so a "layer" of transactions has the size $\lambda * h$, and we want the Tangle size to be large relative to this number, i.e. having many layers. For a multi-agent model we also need to include the latency between agents d, since transactions become visible (and available for tip selection etc.) in "chunks" of at the maximum $\lambda * (h + d_{max})$ where d_{max} is the maximum latency in the distance matrix. In addition, with multiple agents and uniform agent choice probabilities c the transaction rate λ is equally split between the agents, i.e. when $N = 2$ we have $\frac{\lambda}{2}$ for each agent. To balance this out and make sure that each agent builds up a large enough sub-Tangle, we could multiply the expression from above with the number of agents, so the final version and most suitable for our simulation framework is:

$$\lambda * (h + d_{max}) * N \tag{5.1}$$

This number should be very large compared to the total number of transactions, but we do not yet have a formal way of saying what "very large" means.

Furthermore, as explained earlier, we expect Tangle quantities to remain stable, and if this is not the case for a simulation with a smaller size (but the same other parameters), the simulation was too short. One could implement a certain threshold, for example the last 90% of the simulation time, and say that if we do not see any dramatic changes, the simulation length is sufficient. We could also run the same simulation with half the number of transactions and see if we detect any significant changes in the metrics we are interested in. However, in the future, more formal work to describe these ideas needs to be done.

5.2 Simulations to measure tip count stability

In this section, we examine different simulation scenarios with regard to tip count stability, mainly intended as a sanity-check of our model and to point out the effect the inclusion of d in the multi-agent setting has.

5.2.1 Relation to previous work

Tip count stability refers to development of the number of tips in the Tangle over time. This is an important property of the Tangle, since this value should be as stable as possible (at least fluctuate around a constant value). If the number of tips would increase, and most unapproved transactions are left behind with time, the confirmation confidence would drop significantly, i.e. transactions would not be confirmed quickly. This is the main reason why [47] derive an analytical expression for the average number of tips in the Tangle when a uniform random tip selection strategy (URTS) selecting two tips is used. The expression is as follows:

$$L_0 = 2 * \lambda * h \tag{5.2}$$

where is L_0 the average value, around which the number of tips in the Tangle should concentrate. If we translate this into our model, $L(t)$, the total number of tips in the Tangle at time t, corresponds to the sum of $T_i(t)$, summed over all agents $i \in A$.

[35] have shown that the above expression holds for results of simulations of the single agent model. They also note a very similar behaviour for the weighted random walk (or MCMC) tip selection algorithm *for small* α. This is expected, since URTS and MCMC are very similar in this case, yet they show how the average L_0 is slightly higher (even for $\alpha = 0$) and argue that this is caused by the internal structure of the Tangle not being uniform. In this section we want to examine what happens with multiple agents, that is analysing the number of tips $L(t)$ over time and thus investigating when the Tangle stabilises. This is intended to serve as an example to show how the multi-agent model and simulations can be used for various analysis of Tangle properties.

5.2.2 Simulation setup

Analysis with regard to the number of tips in (sub-)Tangles are conducted with 5,000 transactions per Tangle, sensible simulation parameters (λ, α, etc.) with regard to the discussion from above, and for two cases: Two "fully" connected agents (minimal possible latency) and two completely disconnected agents (maximum possible latency). Note that an appropriate number of transactions for simulating disconnected agents is difficult to determine with Equation 5.1, since we must have a very large number to make the sub-Tangles disjoint, however we expect to create two "separate" Tangles for each agent, so we can ignore the d_{max} for this scenario.

5.2.3 Simulation results for connected agents

First, it might be interesting to simulate two fully connected agents. We would expect a fairly similar behaviour as for the single agent model, since both agents essentially have a full view on the Tangle, yet latency plays a role. We set $d = 1$ since we fixed $h = 1$ and intuitively h and d are related: Whereas in the single agent model the network latency is included in the h parameter, we "extract" latency in the multi-agent model and let d alone describe the latency and h the POW (and tip selection computational effort). Thus, viewing d and h relative to each other, it is reasonable (and simplifying) to let both values have the same magnitude.

In Figure 5.5 we see the number of $T_i(t)$ for both agents over time for 5,000 transactions for a simulation with URST. Averaged over 50 simulations run with the multiprocessing script, the number of tips fluctuate around an average value of 66.45 for both agents, so above $2 * \lambda * h$ (which would be $= 50$ for this simulation setup). For the MCMC algorithm we get a similar result with

average values of 67.99 and 67.95 for both agents respectively (average over 50 simulations – for the result of one simulation see Figure 5.6). Note that technically the number of tips in the MCMC algorithm is always increasing as shown by [35], but our simulation is too short to show that.

Generally, in the multi-agent model a higher average number of tips than in the single agent model can be expected. The latency d causes agents to see transactions referenced by another agent as tips, when the transaction's approvers are not visible (shown in Figure 4.1). Following this logic, we would expect L_0 to increase with d, since this means that more transactions are not visible and hence regarded as tips.

The most important point is that we see how a multi-agent Tangle with fully connected agents and using URTS stabilises as well, however we are missing an analytical expression for that, i.e. a similar formula to the one above, but including d.

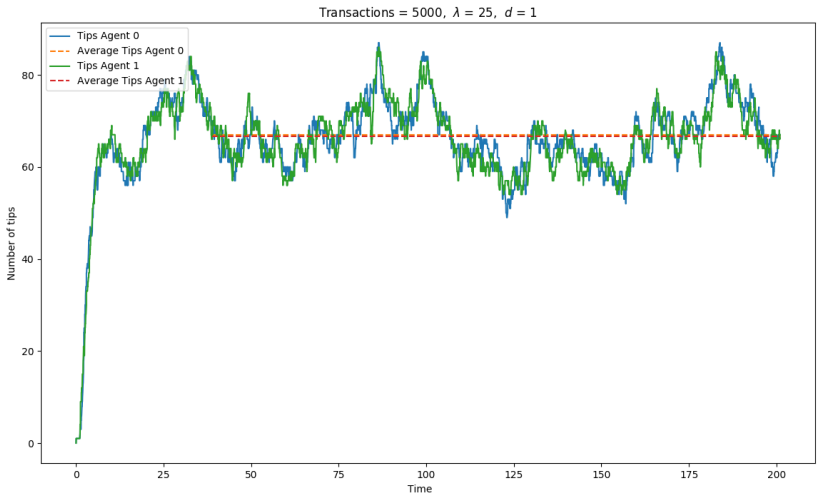

Figure 5.5: Number of tips over time for a uniform random tip selection for two fully connected agents.

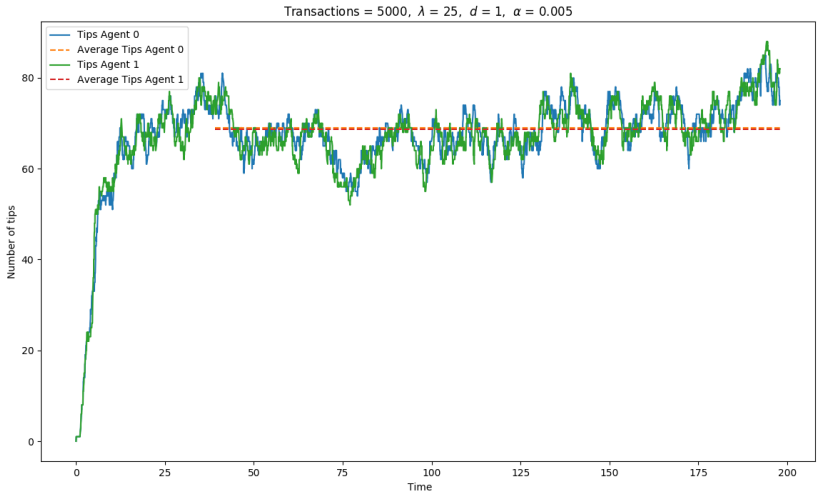

Figure 5.6: Number of tips over time for a weighted random walk tip selection for two fully connected agents.

5.2.4 Simulation results for disjoint agents

We can further examine results of the same experiment with two completely disconnected agents. For URTS, we expect the average number of $T_i(t)$ for both agents over time in their sub-Tangles to follow $\frac{2*\lambda*h}{2}$, since our "global" λ is split up between the agents equally, such that every agent has a transaction rate of $\frac{\lambda}{2}$. This is because of the default uniform agent choice probabilities, or in other words, we assume equal hashing power of both agents. As seen in Figure 5.7, this can be confirmed for disjoint agents. The average number of tips for both agents over 50 simulations is 25.59 and 25.13 respectively. This is a sanity check of our model and simulation framework compared to the single agent model simulation results by [24]. Moreover, as [24] also find, with the MCMC tip selection algorithm, L_0 is slightly higher than as given by Equation 5.2 (for our model again divided by two), for small α. Note again, for $\alpha > 0$, tip count in general increases over time, but our simulation is too short to show this. An example run is shown in Figure 5.8, and the agents' average number of tips averaged over 50 simulation runs with the same parameters are 27.12 and 27.04 respectively.

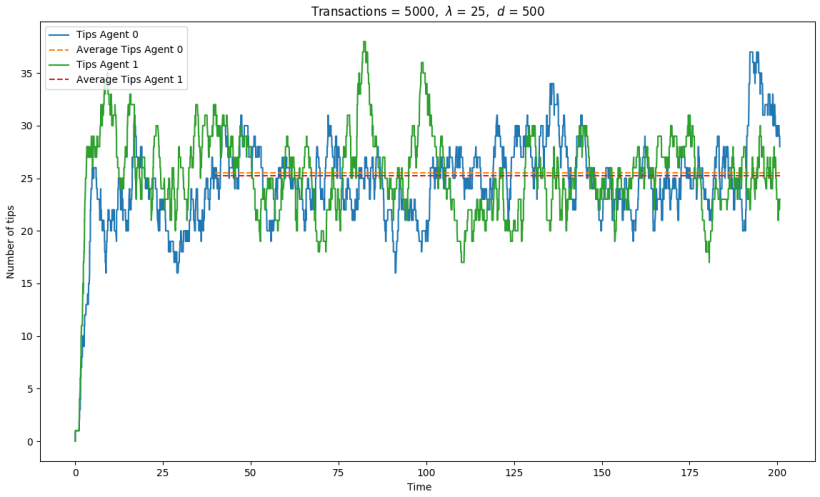

Figure 5.7: Number of tips over time for a uniform random tip selection for two disconnected agents.

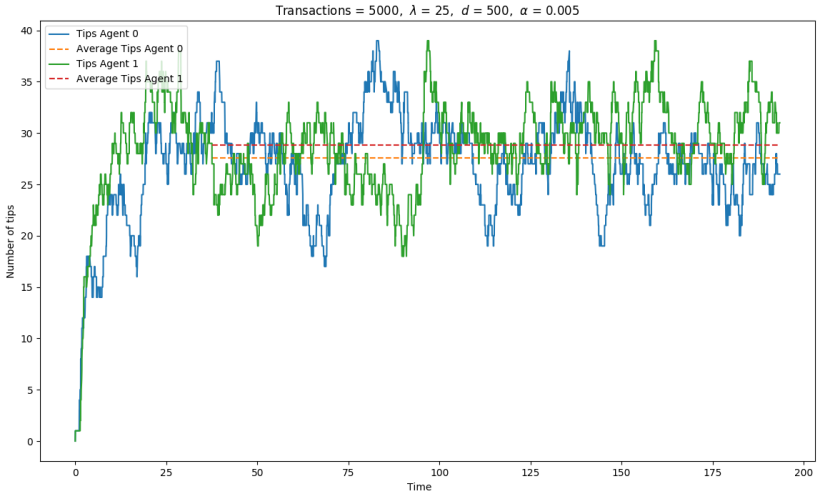

Figure 5.8: Number of tips over time for a weighted random walk tip selection for two disconnected agents.

Overall, we see that our simulation framework can be used to analyse multi-agent Tangle properties

and results are generally in line with expectations from our model. With regard to the problem of simulations being too short we show that even with just 5,000 transactions being in the simulated Tangles, results are as expected and sub-Tangles seem to stabilise.

We show that for URTS the stability of a multi-agent Tangle can no longer be expressed solely in terms of h and λ, but that we need a new expression including d. This could be used to study other Tangle quantities in a more realistic multi-agent setting and hopefully provide insights and parameters for running real IOTA nodes.

5.3 Simulations to measure heaviness of different Tangle branches

The idea dealt with in this section is closely related to what [47] calls a "parasite chain attack", in which an malicious agent builds up a secret sub-Tangle (or rather just a thin chain called parasite chain) that occasionally references the main-Tangle, and tries to trick new incoming transactions into believing that the parasite chain is the branch to reference. Consequently, the main branch would get orphaned.

5.3.1 Relevance and intuitions

Our idea is related: We want to simulate a larger main-Tangle and a smaller sub-Tangle (we will refer to a main-Tangle agent and sub-Tangle agent), and different phases in which these grow separate from each other. After merging, i.e. letting the agents see each other, we want to estimate the probability of a new incoming transaction referencing each agents' current tips (his branch). Note that when we talk about a *branch* here, this is not exactly the same as a sub-Tangle. Whereas a sub-Tangle is defined by visibility of transactions as explained in Chapter 3, a branch simply refers to all current tips (and transactions referenced by those) issued by an agent. The probabilities to attach to the agents' tips are called *attachment probabilities*, and are calculated using exit probabilities as explained in Chapter 4.

We conduct this analysis for different α-levels and show how attachment probabilities change over time. In other words, we try to quantify the "heaviness" of each agent during a simulation. This is important in many contexts of the Tangle research, since IOTA's distributed consensus mechanism is based on the idea that the main branch of the Tangle is the "heaviest".

Our approach of summing the exit probabilities of the tips of each agent measures heaviness indirectly, because the exit probabilities resemble the probabilities of random walkers walking on the heavy branch, and ending up on the tips, as if the tip selection algorithm would have been run. This is very similar to the logic behind confirmation confidence calculation in Chapter 4. Thus, we know which agent would be more likely to be attached to by an incoming transaction. We expect to show that the main-Tangle branch has a higher probability of being referenced (being "heavier"), as long as it issues the majority of transactions. To the best of our knowledge, there does not exist such an analysis so far.

First, we want to explain the simulation and scenario setup. Note that we make use of our parameter change event method (explained in the previous chapter), to change the distance matrix d and the agent choice probabilities c during the simulation.

5.3.2 Simulation setup scenario 1

There are three main phases in this simulation setup:

1. *Growth phase main-Tangle:* In this phase we have $c = [1.0, 0.0]$, so the main-Tangle is growing with a transaction rate of λ (this is equivalent of saying that its agent has 100% of the hashing power). The tip selection algorithm is the MCMC random weighted walk. The main-Tangle contains 10,000 transactions at the end of this phase.

2. *Growth phase sub-Tangle:* In the second phase, the sub-Tangle starts growing and we change the agent choice probabilities to $c = [0.7, 0.3]$ for the next 5,000 incoming transactions (so the sub-Tangle will have around 1,500 transactions and the main-Tangle around 13,500 transactions at the end of this phase). This means that the main-Tangle agent has 70% of the hashing power, and the sub-Tangle agent 30%. The sub-Tangle is hidden, so $d = \infty$ (in practice a very large value).

3. *Merging of both:* In the last phase, the agents start seeing each other for another 5,000 transactions, so we set $d = 1$ ("fully" connected, as described in the previous section). During this phase we measure attachment probabilities every 50 transactions to get 100 data points.

We use different α-levels to highlight differences in attachment probabilities it causes (0.001, 0.005, 0.01, 0.1). An example of the `config.ini` file for this scenario is shown in the Appendix D.

5.3.3 Results of scenario 1

Obtaining larger sample sizes requires substantial computational power and thus the shown results are just for illustration. Simulations rely heavily on random numbers, but the fact that we run experiments in this section with at least 5,000 to 10,000 transactions *per phase*, and at least 15,000 transactions *per simulation*, means that Tangles should stabilise.

Throughout this dissertation we highlight the importance of the weighted random walk algorithm as backbone of the IOTA protocol. With our simulation scenario and the measurement of attachment probabilities we can determine if the tip selection algorithm works as intended, i.e. builds up a heavy main-Tangle branch and hence over time makes the network more secure. We have stressed that this is under the assumption that honest agents accumulate over 50% of the hashing power and constantly create new transactions [10]. In fact, this what our first simulation scenario mirrors, since the main-Tangle issues more transactions than the sub-Tangle (higher agent choice probability) – however we do not distinguish between honest and malicious agents in our simulation framework. We expect to show that the attachment probability for the sub-Tangle branch stays below 50%. We also expect it to decrease with α, since α determines how strong the bias towards the heavy branch is.

All following graphs show the attachment probability of the sub-Tangle agent only, starting at the moment of merging (phase 3). As can be seen in Figures 5.9, 5.10, 5.11 and 5.12, our expectations can be mostly confirmed. The graphs show that the attachment probability of the sub-Tangle agent fluctuates around the agents' hashing power of 30%. This in turn means that a new incoming transaction is more likely to attach to one of the current tips issued by the main-Tangle agent.

We also show that the level of α has an effect, i.e. the higher α, the longer the attachment probability remains 50%. 50% is the default value in phase 2, when the two agents agents are completely disconnected, or partitioned. In other words, this suggest that after merging the branches stay partitioned for some time (for large α-values). This is likely to be caused by one heavy chain of transactions, which is "accumulating" most cumulative weight. With large α, the accumulating effect is stronger, so it takes a longer time until the attachment probability decreases to a level close to the hashing power. We also see that the higher α is, the faster the decrease and the higher the variance in the attachment probability is.

As explained in the beginning of this chapter, α should not be too large, since then the probability of transactions being left behind gets too high. The results presented here suggest that a too large α value has negative effects for the Tangle stability (higher variance). Thus, future work could

try use the approach presented here to further research an upper bound for the α value.

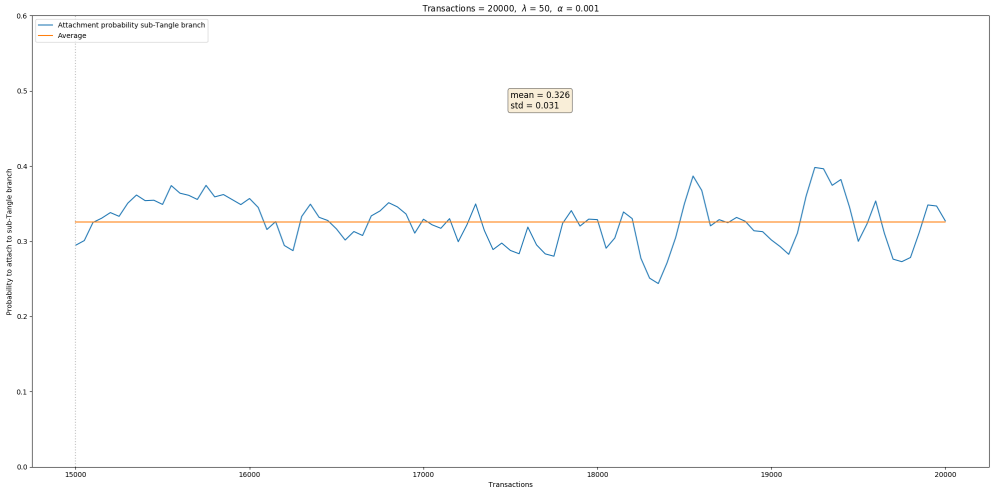

Figure 5.9: Probability to attach to the sub-Tangle branch for scenario 1 and $\alpha = 0.001$.

Figure 5.10: Probability to attach to the sub-Tangle branch for scenario 1 and $\alpha = 0.005$.

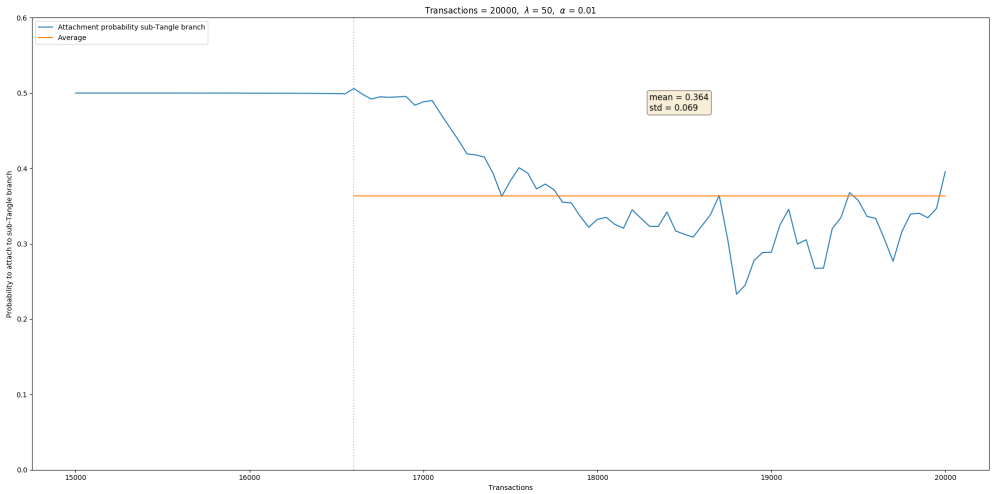

Figure 5.11: Probability to attach to the sub-Tangle branch for scenario 1 and $\alpha = 0.01$.

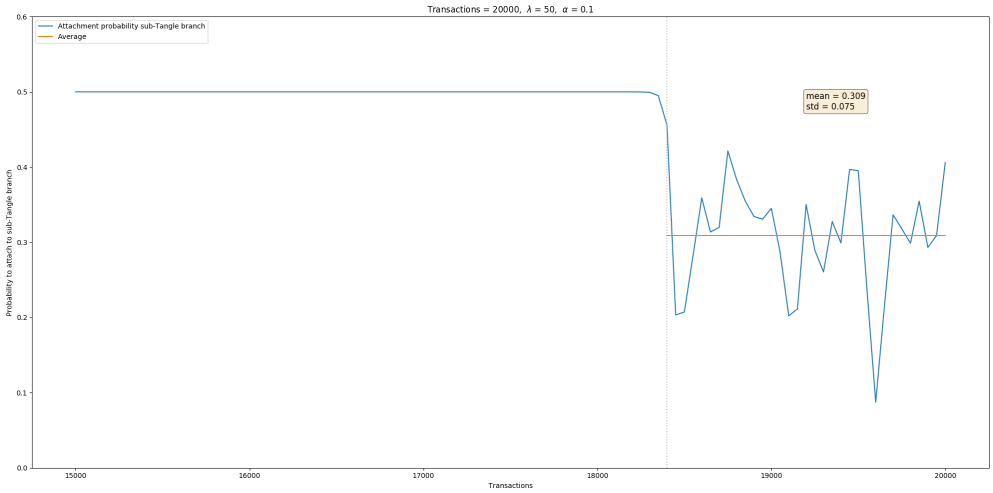

Figure 5.12: Probability to attach to the sub-Tangle branch for scenario 1 and $\alpha = 0.1$.

For comparison purposes we also want to show what happens when a large main-Tangle is not build upfront (Figure 5.13). When we simulate two fully connected agents with the same transactions rates as before, $c = [0.7, 0.3]$, and observe the attachment probability for the sub-Tangle branch, we notice that it fluctuates around the average of 0.32 and standard deviation is low. This again highlights that attachment probabilities are strongly related to the hashing power. However, we have illustrated that a heavy main-Tangle and high α-values can moderate this relation.

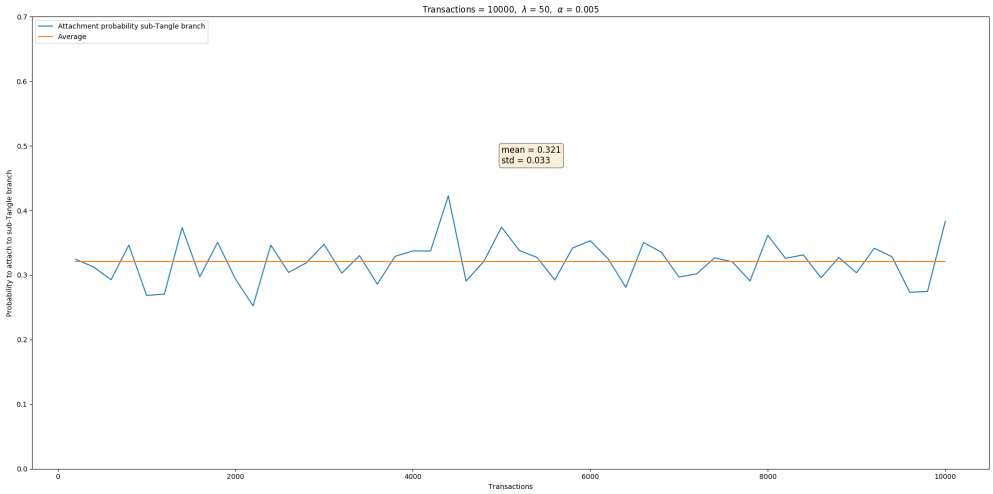

Figure 5.13: Probability to attach to the sub-Tangle branch for a simulation of 10,000 transactions with $\alpha = 0.005$.

The main-Tangle was large by design in this simulation scenario. Next, we want to compare the results of this section with a scenario with a smaller main-Tangle. We expect this to increase the attachment probability of the sub-Tangle branch.

5.3.4 Simulation setup scenario 2

We have three main phases in this simulation setup again. The phases are largely identical to the ones of the previous scenario, but we let the main-Tangle grow for only half the time.

1. *Growth phase main-Tangle:* We start with $c = [1.0, 0.0]$ and use the MCMC random weighted walk. The main-Tangle contains 5,000 transactions at the end of this phase.

2. *Growth phase sub-Tangle:* We set $c = [0.7, 0.3]$ for the next 5,000 incoming transactions and have $d = \infty$ (very large) again.

3. *Merging of both:* In the last phase, the agents start seeing each other for 5,000 transactions, i.e. $d = 1$, and we measure the attachment probabilities every 50 transactions to get 100 data points.

Different α-levels are evaluated (0.001, 0.005, 0.01, 0.1).

5.3.5 Results of scenario 2

The illustrations show the attachment probability of the sub-Tangle agent from the moment of merging (phase 3). As can be seen in Figures 5.14, 5.15, 5.16 and 5.17, the results of this scenario are very similar to the previous ones. We noted for the results of scenario 1 that the main-Tangle was heavy by design, and therefore we wanted to examine what happens with a smaller main-Tangle. We expected this to increase the attachment probability of the sub-Tangle branch, since the main-Tangle should accumulate less cumulative weight. However, the results do not show this, but rather just confirm the previously made conclusions about the influence of α and the dependence on hashing power. Future research and larger simulations should examine this matter further.

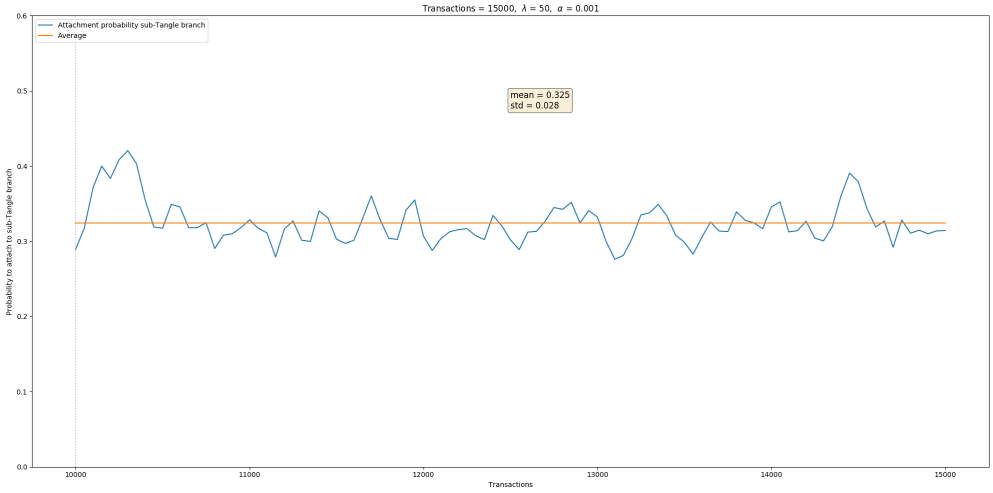

Figure 5.14: Probability to attach to the sub-Tangle branch for scenario 2 and $\alpha = 0.001$.

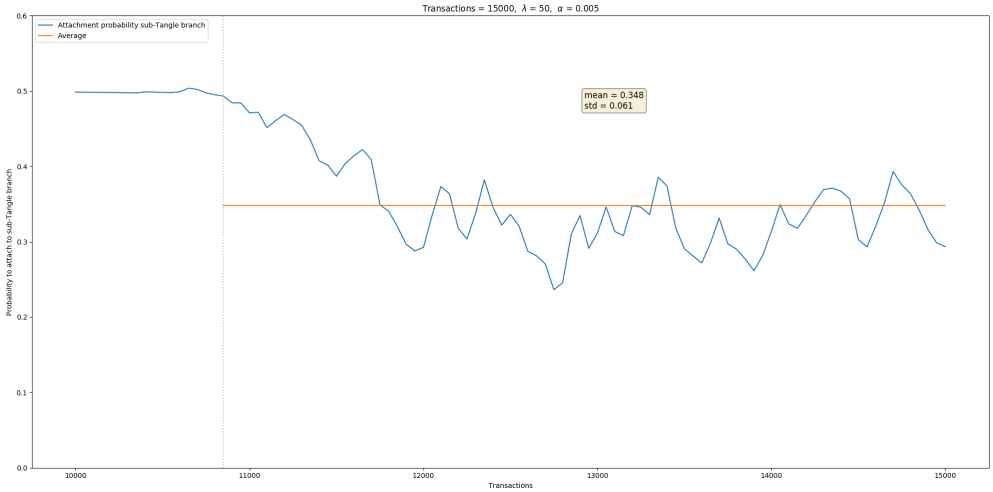

Figure 5.15: Probability to attach to the sub-Tangle branch for scenario 2 and $\alpha = 0.005$.

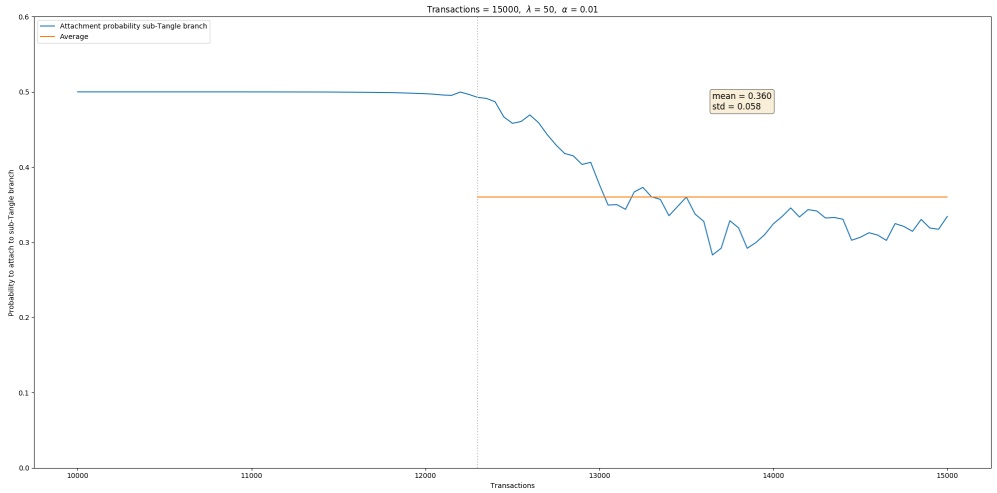

Figure 5.16: Probability to attach to the sub-Tangle branch for scenario 2 and $\alpha = 0.01$.

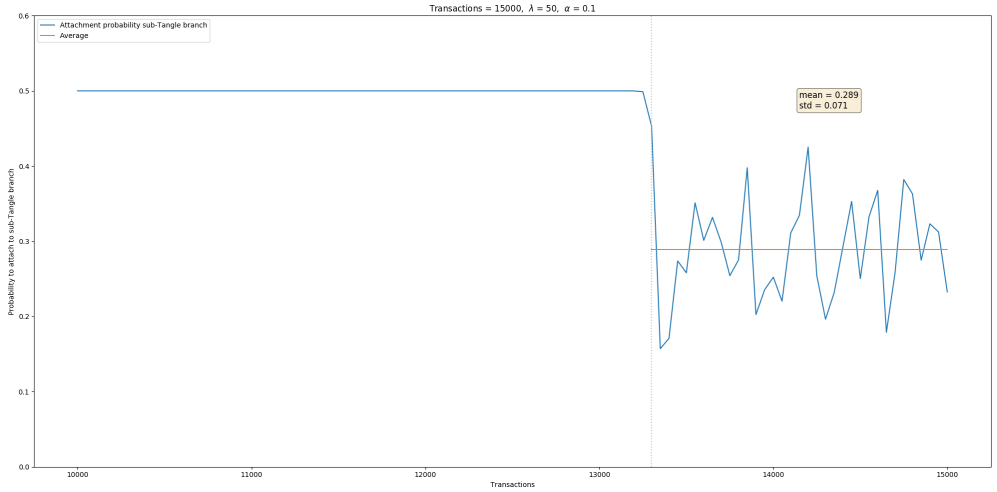

Figure 5.17: Probability to attach to the sub-Tangle branch for scenario 2 and $\alpha = 0.1$.

Next, we formulate and simulate a last scenario to see how "durable" attachment probabilities are.

5.3.6 Simulation setup scenario 3

In both of the above scenarios we have seen that the attachment probabilities are closely related to agent choice probabilities, i.e. the transaction rate of each agent (or hashing power). Hence, we want to evaluate a scenario, where we set the agent choice probabilities to $c = [0.5, 0.5]$ in the last phase to see if the sub-Tangle agent can quickly accumulate enough cumulative weight such that its attachment probability increases to around 50% as well.

1. *Growth phase main-Tangle:* We start with $c = [1.0, 0.0]$ and use the MCMC random weighted walk. The main-Tangle contains 10,000 transactions at the end of this phase.

2. *Growth phase sub-Tangle:* We set $c = [0.7, 0.3]$ for the next 5,000 incoming transactions and have $d = \infty$ (very large).

3. *Merging of both:* Agents start seeing each other for the next 5,000 transactions, i.e. $d = 1$, and attachment probabilities are measured every 100 transactions.

4. *Merging with equal transaction rate:* In the last phase, we set $c = [0.5, 0.5]$ for 2,000 transactions, so that each agent issues transactions with $\frac{\lambda}{2}$ (equal hashing power).

Different α-levels are evaluated (0.001, 0.005, 0.01, 0.1).

5.3.7 Results of scenario 3

The results of this scenario (Figures 5.18, 5.19, 5.20 and 5.21) illustrate that the attachment portability for the sub-Tangle branch quickly increases to around 50% after we set $c = [0.5, 0.5]$. This is the expected level given by the hashing power, but the important question is for how long a heavy main-Tangle would have an advantage over a smaller sub-Tangle (how long the increase takes). We would expect the superior heaviness of the main-Tangle branch to last longer, or put differently, it should take a longer time for the sub-Tangle branch to "catch-up" and accumulate enough cumulative weight to make it as heavy as the main-Tangle branch. It is difficult to confirm this with the illustrative results presented here, more advanced measures are needed. Further research should investigate this matter and try to measure the time needed to "catch-up".

In summary, all results suggest that attachment probabilities in these scenarios are primarily governed by the hashing power, and that we need larger simulations to see the advantage of a heavy main-Tangle prevail. The α-level seems to play a role as well, but we need further tests to see how exactly it influences the relationship.

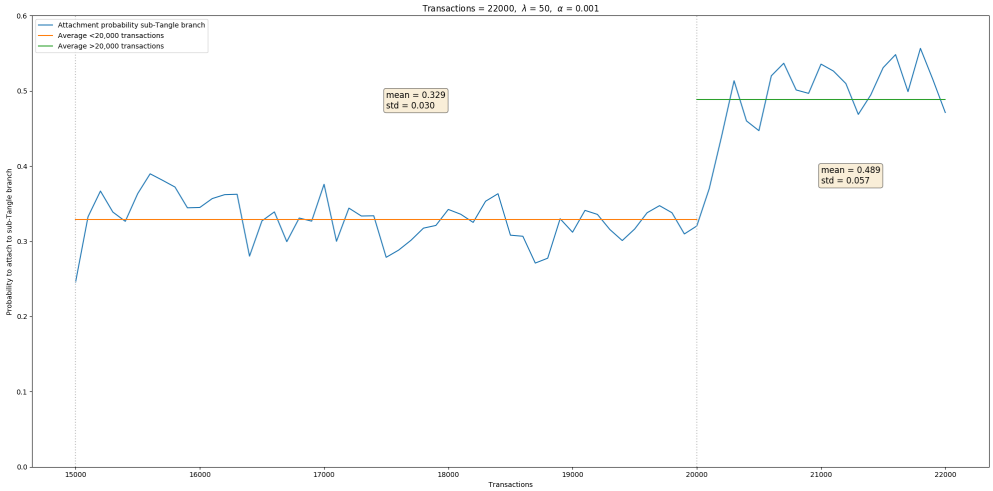

Figure 5.18: Probability to attach to the sub-Tangle branch for scenario 3 and $\alpha = 0.001$.

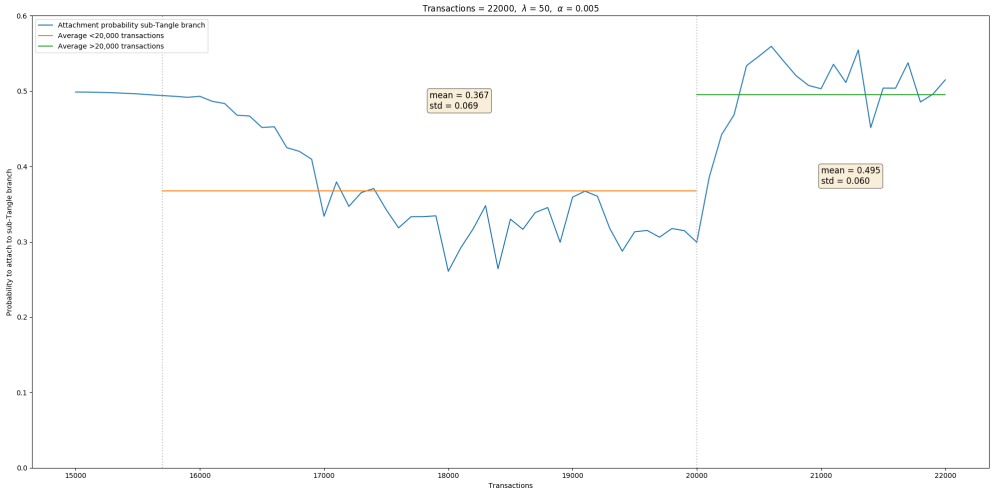

Figure 5.19: Probability to attach to the sub-Tangle branch for scenario 3 and $\alpha = 0.005$.

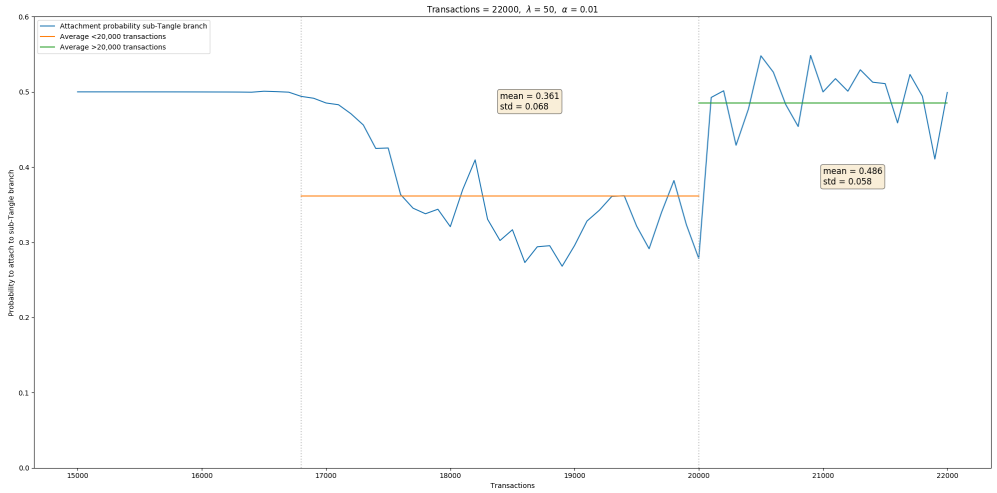

Figure 5.20: Probability to attach to the sub-Tangle branch for scenario 3 and $\alpha = 0.01$.

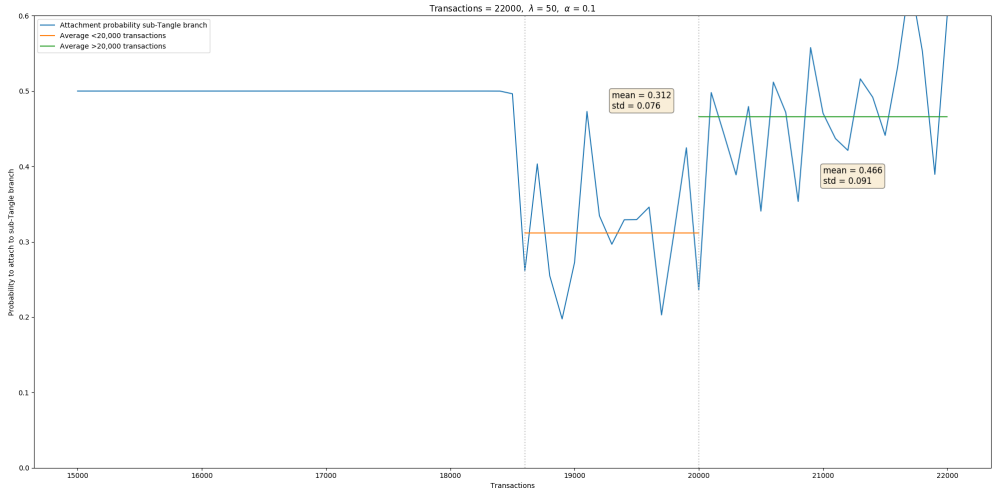

Figure 5.21: Probability to attach to the sub-Tangle branch for scenario 3 and $\alpha = 0.1$.

5.4 Evaluation

Our simulation framework is flexible enough to simulate various different scenarios and produces results that can be used to draw conclusions about the effectiveness of the protocol main puzzle piece, the MCMC tip selection algorithm. Some of these conclusions are summarised as follows:

- Assuming both agents use the recommended tip selection algorithm of the protocol, the weighted random walk, a large main-Tangle has enough cumulative weight to "outweigh" a smaller sub-Tangle, measured by attachment probability over time, assuming the main-Tangle keeps issuing the majority of incoming transactions.

- Attachment probabilities show higher variance for larger values of α, so the more the random walk is biased towards the heavy branch. Larger α values were expected to decrease attachment probabilities of the smaller sub-Tangle, but further simulations and more advanced measures are needed to investigate this.

- A smaller upfront built main-Tangle is less heavy and was thus expected to increase attachment probabilities of the smaller sub-Tangle, but the results have shown no difference to the scenario with a larger upfront built main-Tangle. Further simulations are needed to investigate this as well.

- The main-Tangle could loose its weight advantage relatively quickly, i.e. a sub-Tangle could quickly catch up when it is issuing as many transactions as the main-Tangle. This was expected to take some more time, but further simulations, more advanced measures and different scenarios have to be examined in order to derive more convincing results.

Chapter 6

Simulations: The multi-agent case

This chapter deals with simulation results and analysis of multi-agent Tangles. In Chapter 3 we introduced the notion of networks of agents, defining a distance matrix of shortest paths. The distance matrices we use are based on a randomly generated sparsely connected agent graphs. We examine the attachment probabilities and see how it is affected by the latency between agents.

6.1 Parameter choices

In the previous chapter we discussed general remarks about the choice of model and simulation parameters, which largely hold for the simulations conducted here as well. However, there is one parameter having significant relevance in a multi-agent setting – the distance matrix d. We explained that in the two-agent model this matrix is reduced to a single parameter. Here, we deal with many agents, and in particular we want to examine what happens when they are sparsely connected. *Sparse networks* are most commonly found in practice and are defined as having a number of links L between nodes N *much* smaller then the maximal possible number of links L_{max} [4], as given by:

$$L_{max} = \frac{N * (N-1)}{2} \tag{6.1}$$

In our multi-agent model, we defined the network of agents as a weighted undirected graph, which translates into a distance matrix d containing the shortest paths between all agents. The distances describe the network latencies between the agents. The matrix is of special interest in the following simulations and we follow the lead of [52], who used an *Erdős–Rényi random network* topology [18] with $N = 20$ nodes for their experiments. We create such a random graph with 20 nodes as well, and ensure that the number of edges is well below $L_{max} = 190$ (for simplicity, we use $L = N$). The initial distance between two neighbouring agents is one (we consider scaling in later experiments), and correspondingly we calculate the shortest distances between all agents. For this task, as well as for the random graph generation, `NetworkX` provides helpful functions. In Figure 6.1 we show an example of such a network of agents and the corresponding d can be found in Appendix E.

6.2 Simulations to measure heaviness of different Tangle branches

We use the same measure to quantify heaviness as before, the attachment probabilities of different sub-Tangle branches. This time, we do not have a main-Tangle and a sub-Tangle with different growth phases, but instead all agents have the same transaction rate (equal hashing power).

6.2.1 Relevance and intuitions

Our idea is to find a relation between the *centrality* of agents in the randomly created agent network and the development of attachment probabilities of their Tangle branches over time. In other words, we want to find out if we can draw conclusions about their "importance", based on how far they are away (or how large the latency is) from other agents. The rationale behind this is that if agents are

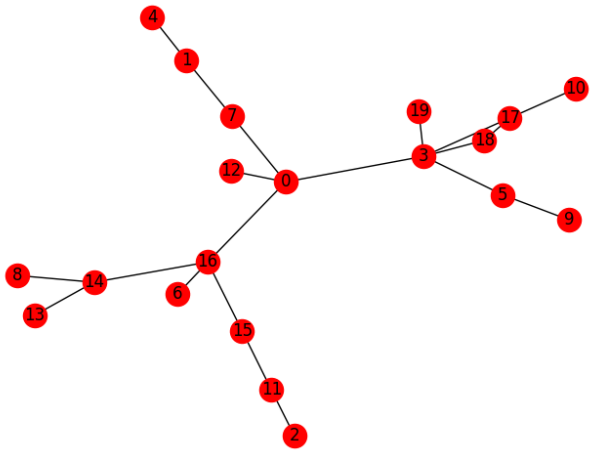

Figure 6.1: Example of a random Erdős–Rényi graph of 20 sparsely connected agents.

well connected, i.e. have small distances to (many) other agents, this means that by definition of visibility in our multi-agent model they will be available for tip selection more often, and thus accumulate more cumulative weight in their branch. To put differently, more closely connected agents see each other more often and connect to each other more frequently, thus building heavier branches.

In order to examine if this idea holds, we need a measure of centrality of agents in the agent network. One of the most intuitive ideas is that an agent is central if he is close to all other agents, i.e. if his average shortest path to all other agents is small [20, 50, 5], usually referred to as *closeness centrality*. Since our random graphs are created using `NetworkX`, we can use one of the provided functions to calculate the closeness centrality of the agents as well. Furthermore, this measure is a good choice in this context, since tip-selection in multi-agent Tangles is highly dependent on the distance matrix d, which contains the shortest paths between all agents. Hence, a centrality measure based on shortest paths is likely to be effective.

If closeness centrality and attachment probabilities are indeed related, we can examine this pattern for different α values and see if this has an effect on the relation. We expect a higher α value to make the relation stronger. Closer agents could build heavier branches together, since we increase the bias for heavier branches in the tip selection mechanism with α.

6.2.2 Simulation setup scenario 1

We start with a smaller networks of agents, $N = 10$, since simulating many agents is computationally intensive. The simulation duration is 10,000 transactions. In Chapter 5 we pointed out several heuristics with regard to simulation length for multi-agent models, but we would need substantially more computational power to realise larger simulations. Node distance is not scaled, that is the distance between two neighbouring agents is 1 by default. In further scenarios we scale the distances to see if this has any effect. Since we are interested in the heaviness of branches, the weighted random walk tip selection algorithm is default in these experiments. Remember that we keep agent choice probabilities uniformly distributed, i.e. all agents have the same transaction rate (the "global" λ, the total hashing power, is equally split).

Since measuring attachment probabilities is computationally demanding (for every agent we need to loop over all other agents), we collect this data every 200 transactions and get 50 data points in total per agent.

6.2.3 Results of scenario 1

In this scenario we did not scale d, meaning that the distances between the agents are relatively small compared to the time horizon of the simulation, which is $\frac{10{,}000}{50} = 200$ (time units). As explained above, we expect an effect caused by visibility of agents in the network and if the visible sub-Tangles diverge heavily (much latency between agents), the effect should be stronger. In this scenario agents 0 and 4 are the most central (closeness centrality values not shown, see next scenarios for more detailed analysis), and in fact have the highest average attachment probability (see Figure 6.2). However, as a result of the relatively small distances in this scenario, the attachment probabilities of the agents fluctuate a lot and the effect we expect to observe is not easy to determine, even for a large $\alpha = 0.1$. Hence, we conclude that scaling of d might help.

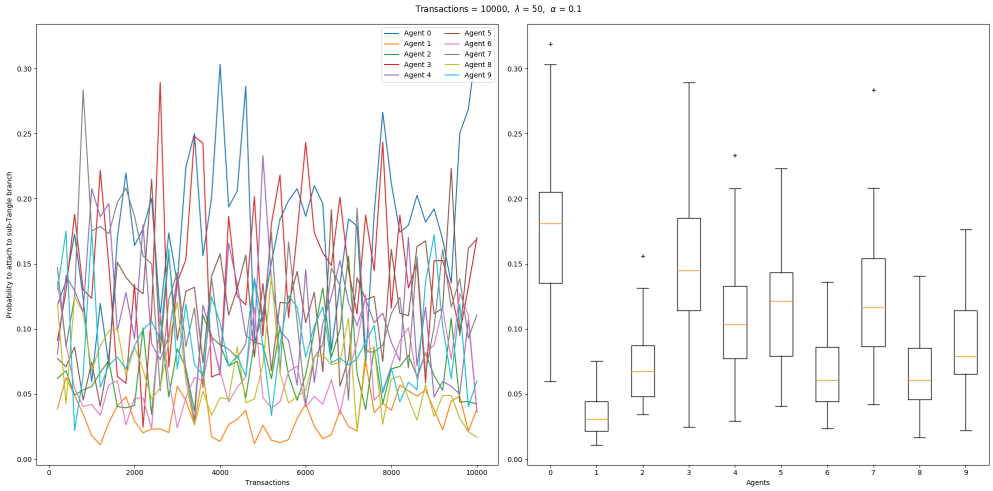

Figure 6.2: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with no scaling of d and $\alpha = 0.1$.

6.2.4 Simulation setup scenario 2

This scenario is almost identical to scenario 1, but we scale the distances in the agent network by a factor 10. This scaling should make it easier for closely connected agents to build heavier branches, since the disconnected agents are even further away and less involved in the tip selection. Note that this scenario uses a new randomly generated agent graph with $N = 10$ agents, which is kept fixed, but we test several α-levels again.

6.2.5 Results of scenario 2

We observe a clear trend, i.e. higher closeness centrality corresponds to larger average attachment probability in almost all cases. We can also see that this effect becomes more pronounced with higher α-values, where the most central, or best connected agent, is able to build heavier branches. However, we cannot show what happens in the long-run, and whether the values eventually converge to 0.1 (the hashing power split), or stay stable at their respective level. Larger simulations are needed to examine this matter.

As in the previous chapter, obtaining larger sample sizes requires substantial computational power and thus the graphs below are just for illustration. However, in order to get more statistically significant results, we used the multiprocessing script to run the exact simulation configuration 20 times per α-value (0.001, 0.005 and 0.01) and averaged the attachment probabilities per agent over the 20 runs. These are in line with the illustration below and shown in Appendix F. Randomness in the multi-agent Tangles is cancelled out and the trends are easier to recognise. Overall, this is a strong indication of a relation between closeness centrality and attachment probability.

The closeness centrality of the agents in the random graph is shown in the following table, in descending order, together with the average attachment probability for the different α-levels:

Agent	**Closeness centrality** (descending order)	**Average attachment probability**			
		$\alpha = 0.001$	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.1$
5	0.6	0.131	0.134	0.136	0.287
0	0.53	0.115	0.129	0.129	0.135
9	0.47	0.111	0.120	0.110	0.107
8	0.43	0.111	0.096	0.101	0.087
3	0.39	0.084	0.092	0.092	0.072
7	0.39	0.089	0.093	0.095	0.055
1	0.36	0.089	0.085	0.081	0.061
2	0.36	0.107	0.089	0.102	0.085
4	0.36	0.086	0.086	0.083	0.062
6	0.27	0.084	0.077	0.072	0.062

Table 6.1: Average attachment probabilities for 10 agents over 10,000 transactions with scaling of d (factor 10) and different α-levels. Closeness centrality and average attachment probability per agent rank the same for most agents.

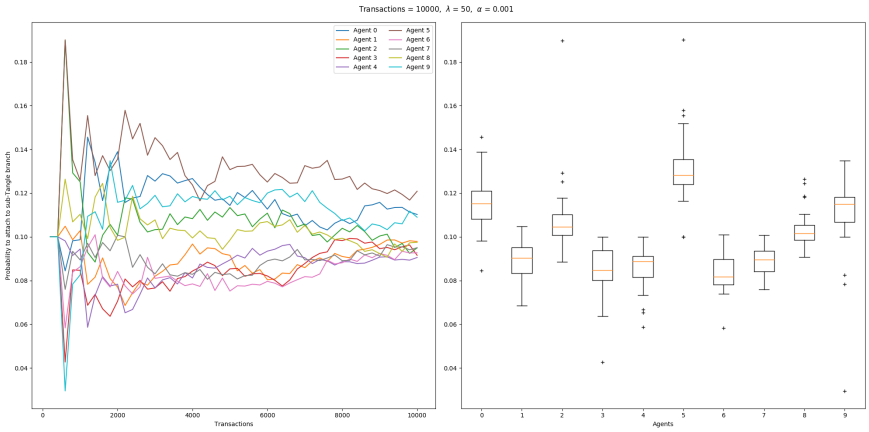

Figure 6.3: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 10) and $\alpha = 0.001$.

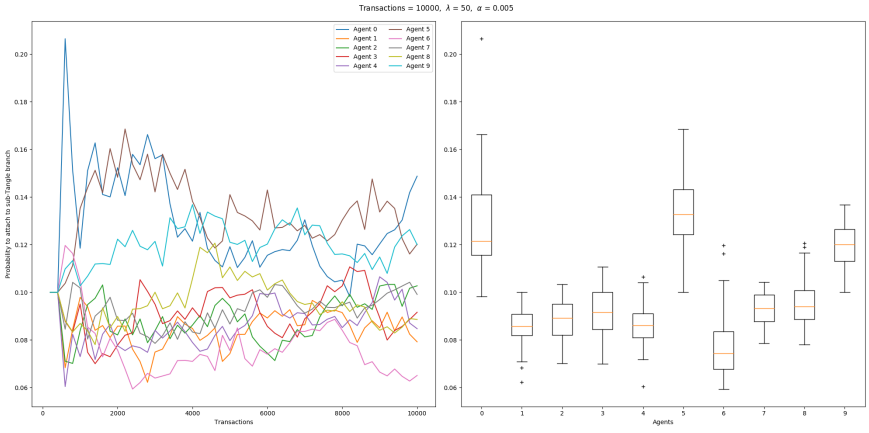

Figure 6.4: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 10) and $\alpha = 0.005$.

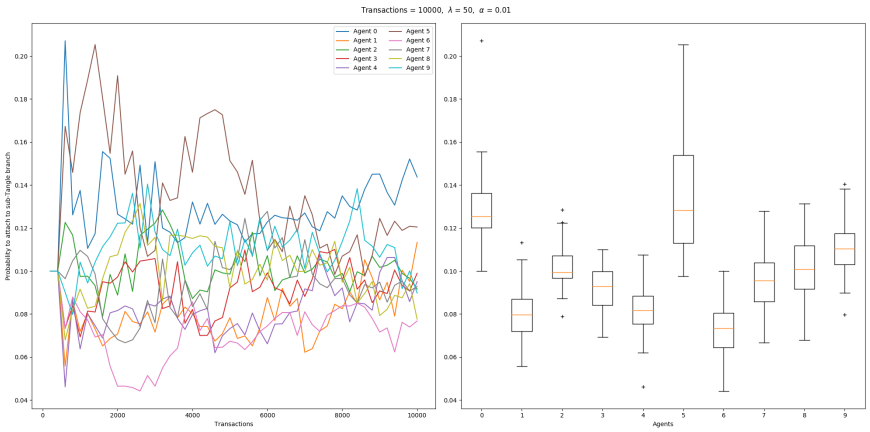

Figure 6.5: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 10) and $\alpha = 0.01$.

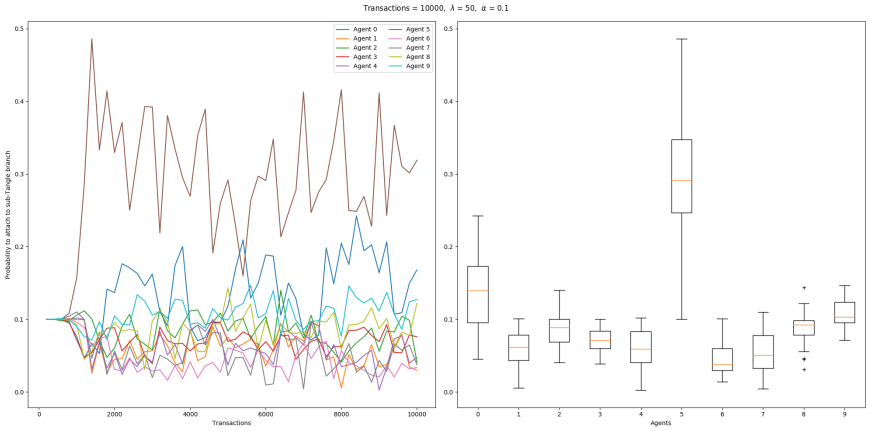

Figure 6.6: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 10) and $\alpha = 0.1$.

6.2.6 Simulation setup scenario 3

In this scenario we keep all previous simulation parameters fixed, but use a larger scaling factor, 20, to see if this makes the observed effect even stronger. We also use a new randomly generated agent network to test for a slightly different network topology.

6.2.7 Results of scenario 3

With a higher scaling factor, i.e. an agent network where latencies are higher relative to the fixed simulation time horizon, the observed trend becomes not necessarily clearer. Once again, higher closeness centrality corresponds to a higher average attachment probability in almost all cases. Higher α-values increase the differences between agents and the values fluctuate stronger. Nevertheless questions remain, for example why with $\alpha = 0.01$ and $\alpha = 0.1$ agents 7 and 9 show differences in attachment probabilities, despite having the same closeness centrality value. This might be due to chance and longer simulations as well as larger sample sizes are needed to investigate this matter further. Nevertheless the relationship between the two metrics becomes clear again.

The closeness centrality of the agents in the random graph is shown in the following table, in descending order, together with the average attachment probability for the different α-levels:

Agent	Closeness centrality (descending order)	Average attachment probability			
		$\alpha = 0.001$	$\alpha = 0.005$	$\alpha = 0.01$	$\alpha = 0.1$
7	0.6	0.139	0.137	0.223	0.123
9	0.6	0.132	0.144	0.128	0.253
8	0.47	0.114	0.120	0.123	0.100
6	0.41	0.091	0.092	0.078	0.076
2	0.41	0.094	0.086	0.075	0.074
3	0.39	0.087	0.088	0.080	0.076
5	0.39	0.090	0.089	0.076	0.074
0	0.39	0.087	0.087	0.076	0.077
1	0.33	0.082	0.079	0.070	0.075
4	0.33	0.085	0.077	0.072	0.073

Table 6.2: Average attachment probabilities for 10 agents over 10,000 transactions with scaling of d (factor 20) and different α-levels. Closeness centrality and average attachment probability per agent rank the same for most agents.

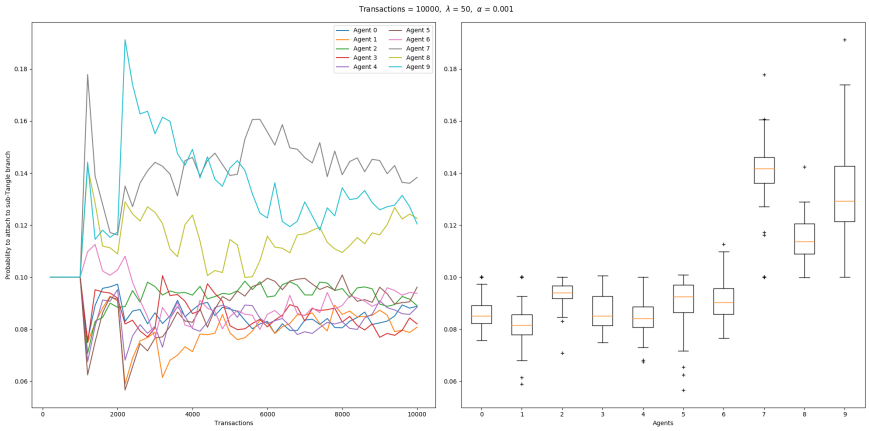

Figure 6.7: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 20) and $\alpha = 0.001$.

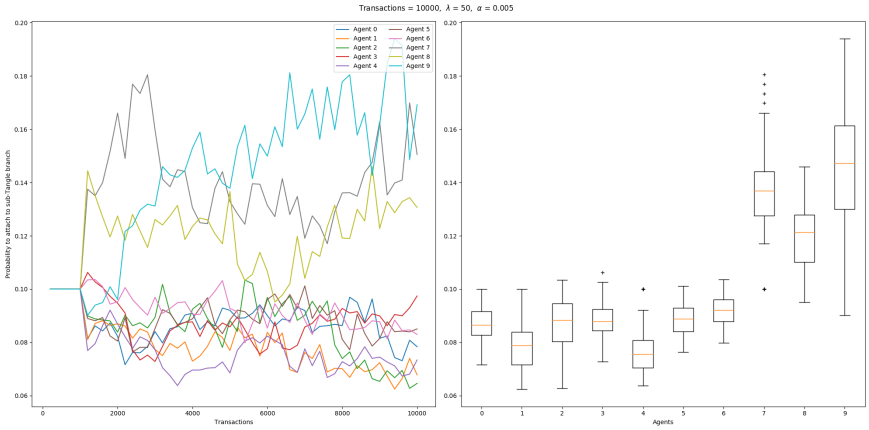

Figure 6.8: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 20) and $\alpha = 0.005$.

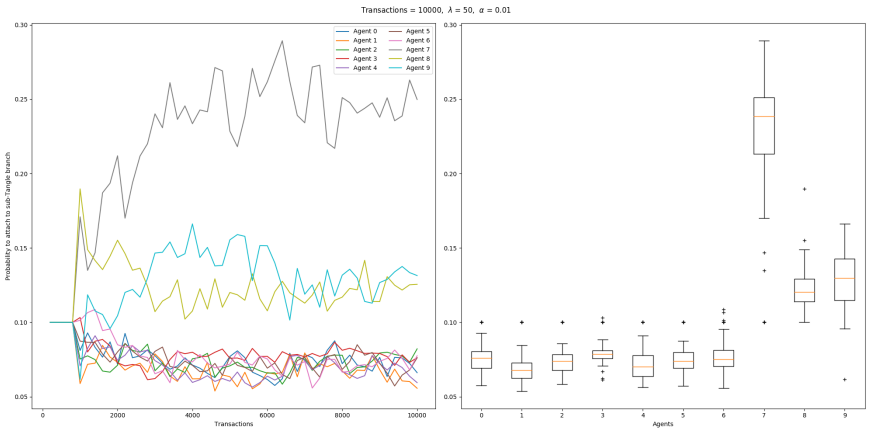

Figure 6.9: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 20) and $\alpha = 0.01$.

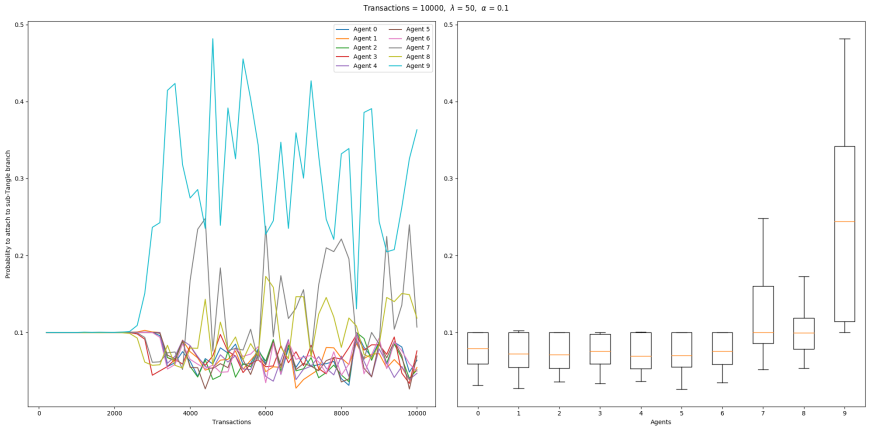

Figure 6.10: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 20) and $\alpha = 0.1$.

6.3 Evaluation

As previously mentioned, the simulation framework developed as part of this dissertation proves to be flexible enough to easily simulate different scenarios and evaluate the Tangle structure with respect to the chance of attaching to the branches of different agents. To the best of our knowledge, the approach presented here is novel and there is no previous work on evaluating how attachment probabilities of sparsely connected agents in multi-agent Tangles evolve over time. Our findings suggest the following:

- Generally, simulations of multi-agent Tangles require much computational power, and experiments with larger sample sizes need to be conducted to further validate our findings.
- Assuming all agents have the same hashing power and use the weighted random walk tip selection, closely connected agents "team up" and build heavier Tangle branches. This is shown by the fact that these agents have high combined attachment probabilities. The reason should be examined further, but it seems plausible that these agents are visible to each other more often during tip selection and hence start to accumulate more cumulative weight in their transactions (branches). This, over time, is reinforced by the bias in the random walk, which is why the observed effect is stronger for larger α values.
- The previous point suggests that the tip selection mechanism over time favours heavier branches (as it is intended by design), which is also likely to have a positive effect on the

confirmation rate of all transactions in that branch (referenced by the tips issued by the "heavier" agent), although we do not provide a specific measure in this context (the confirmation confidence measure in Chapter 4 is sub-Tangle specific, not branch specific). Further simulations and more advanced measures are needed to investigate this.

- Badly connected agents are "left-behind" and are not able to accumulate as much cumulative weight as the well connected agents. This in turns might have negative consequences for their confirmation rate as well, since new incoming transactions are less likely to connect to tips belonging to these agents. Further simulations and more advanced measures are needed to investigate this as well.

Chapter 7

Conclusion

This section summarises the main contributions of this project, as well as areas for applications and future research to overcome the limitations and possibly add further findings to the ones presented in this dissertation.

7.1 Summary of achievements

The aim of this dissertation was to provide a realistic model for analysis and simulations of the Tangle, the distributed ledger structure at the heart of the IOTA protocol. Thus, we extended the single agent model described in the IOTA white paper to a multi-agent model. We believe that this model represents the reality of P2P networks more accurately, since it is based on an asynchronous network of agents, having arbitrary network latencies between each other. Besides, a multi-agent model could make it easier and more intuitive to model various scenarios involving multiple agents, such as side Tangle attacks, or just in general a coordinator-less protocol implementation. Hence, we formalised the multi-agent model and the implemented algorithms. We pointed out why a continuous time model was chosen over a discrete time model, and aspire to help in the establishment of a common language for further Tangle research.

The multi-agent model has been implemented in a simulation framework in Python, along with all required algorithms used in the IOTA consensus mechanism, most importantly the tip selection algorithm. We also implemented two other basic tip selection algorithms to show differences and make simulation results comparable to previous ones. The simulation framework takes all model parameters as input and allows for the specification of various simulation scenarios. These can be provided over the command line in a designated file. Furthermore, algorithms to calculate and measure specific Tangle-metrics such as exit probabilities, attachment probabilities and conformation confidences have been implemented. The user can collect this data and examine how multi-agent Tangles evolve. Visualisations can be produced to help with this task. Furthermore, the framework can serve as a basis to implement algorithms to simulate other DAG-based protocols, which has already started as part of other research projects.

During the implementation of the multi-agent simulator it has become apparent that certain calculations require substantial computational power. We have outlined two different algorithms we could use in our framework to update cumulative weights of transactions (essential during the MCMC tip selection algorithm), but both have shown significant difficulty to run in acceptable time, since it is necessary to iterate over the complete DAG (if the number of transactions in the simulation is very large). This highlights a problem for the protocol in practice as well, hence we help justifying why the concept of milestones (or a different way of saving a snapshot of the Tangle) is needed. However, as described in the future work section below, this solution is not perfect for various reasons, and there are alternative ways of ensuring fast finality of transactions.

Furthermore, the multi-agent simulation framework has been used to run experiments with the goal to examine the behaviour of multi-agent Tangles with respect to stability and effectiveness of the MCMC tip-selection algorithm. To the best of our knowledge, there is no previous work and no simulations on multi-agent Tangles in an asynchronous setting. Different scenarios have been formulated and evaluated with respect to the influence of hashing power and latency on Tangle structure, as well as the effect of α, the random walk bias parameter. For the minimum multi-agent model with just two agents we examined the development of tips over time and showed how sub-Tangles stabilise for a uniform random tip selection, and that the predictions for a single agent model just hold for fully disconnected agents. Furthermore, we model agents with different hashing-power and show how this influences the attachment probabilities, i.e. the chance of a new incoming transactions selecting the current tips issued by each agent. Our results suggest that the MCMC algorithm is effective, that is a "heavy" Tangle branch is more likely to be attached to by a newly incoming transaction. This is one of the fundamental ideas of the protocol and the consensus algorithm. Our results suggest that hashing power influences the attachment probabilities strongly. Finally, we conduct experiments with many agents, based on randomly generated sparsely connected network graphs. The simulation results have led to the conclusion that tips issued by agents which are centrally located in the network are more likely to be referenced by an incoming transaction.

7.2 Applications

With the increasing public and academic interest in distributed ledger applications and protocols, and recently with increased popularity of DAG-based solutions, it becomes even more pivotal to research these protocols and understand benefits and shortcomings. In order to make this research fruitful, appropriate models and simulations should be used. These models need to be close to reality, and thus the presented multi-agent model and simulation framework can be used to conduct further research on multi-agent Tangles.

First, research on parameters for running real IOTA nodes can be conducted, most importantly the α-parameter, and how it should be adjusted in an asynchronous setting. Our results illustrate that α influences the probability with which tips issued by agents are selected, but it is still unclear how exactly latency comes into play. Our results suggest that well connected agents gain an advantage, thus adjustments of α might be needed for less well connected agents. Second, the model and framework might be of use for the simulation of specific attacks against the network. With a model for multi-agent settings, malicious agents and even networks of malicious agents could be simulated and evaluated.

7.3 Limitations and future work

Based on the formal description of the multi-agent model, more formal analytic work would be beneficial. Compared to blockchain-based protocols, DAG-based ones and the IOTA protocol are a relatively novel concept, and would benefit from research on its benefits and drawbacks. As discussed throughout this dissertation, Tangles are a complex mathematical structure and many parameters collude (as shown by the simulation framework as well). There are many phenomena of (multi-agent) Tangles with respect to the MCMC random walk algorithm, parameter choices, security and stability (among others), which lack formal analysis, possibly using tools including matrix or graph theory. Furthermore, a multi-agent Tangle model could be formally examined with regard to an asynchronous communication model [45], or as an extension of the formal framework by Kiayias and Panagiotakos [31]. Moreover, intentions to model Tangles as thermodynamical objects and to use concepts from statistical physics to examine its properties have been mentioned [35].

We showed that calculations on DAGs are very time-consuming, which is why in practice milestones are used. However, milestones are in fact very similar to blocks in regular blockchain-based protocols (a collection of transactions, which are deemed confirmed once they are in an accepted

block or milestone), but the disadvantages of regular block-chain based systems have been stressed earlier. We also described how milestones in IOTA make the network essentially centralised. On the other hand, milestones mean that finality of transactions is reached instantly (when they are approved by the coordinator). To make this centralisation obsolete and finally use the distributed consensus mechanism based on the MCMC algorithm, an extension of the protocol might be required. This extension could still ensure trust-less and instant finality – something similar has for instance been proposed for Ethereum [12], a finality system as overlay for proof of work block-chain protocols. As an alternative, a concept similar to the Meshcash framework [7] could be needed, where ideas from the byzantine agreement literature are used to develop a consensus model with two features: A combination of a consensus protocol enabling fast consensus and finality in the short-term, together with a long-term focused protocol ensuring robust, irreversible consensus on transactions (essentially creating milestones). In the background section of this dissertation we already mentioned the work done around GHOST, SPECTRE and PHANTOM [54, 38, 52, 51], which use a DAG-based approach to build scalable protocols with fast transaction irreversibility. The consensus algorithms used in these approaches should be evaluated and compared to the IOTA consensus mechanism.

We also believe that an in-depth analysis of the algorithms used to simulate multi-agents would be useful to detect further efficiency bottlenecks in the simulation framework and possible implications for the real-world IOTA implementation. Furthermore, this could make large scale simulations more feasible, which is one of the main drawbacks of this project. Simulations of more than 30,000 transactions currently require substantial computational power. However, simulations with large number of transactions and also larger sample sizes are needed in order to validate the findings we presented, which are of rather illustrative nature. This would also be useful to verify analytical calculations, or would help if these are hard or infeasible because of strong and unrealistic assumptions. Hence, we believe that more simulations are desirable for further quantitative analysis of (multi-agent) Tangles. Different parameter sets and algorithms should be used, and newly designed tip-selection algorithms could be examined with regard to their effects on Tangle stability and security.

Security in specific should be one main future research focus, and the multi-agent model could be used to formalise and simulate attacks against the network, such as side Tangle attacks we have already experienced in reality. Hence, the framework should distinguish between honest and malicious agents. Overall, a well balanced mix of simulation experiments as well as predictions derived from formal analysis will be useful to make Tangle research more robust and allows for better comparisons to research on other DAG-based protocols. This field will remain an active research area in the coming years and it will certainly be interesting to see how distributed ledger applications and research evolve in the future.

Appendix A

UML diagram of the multi-agent simulation framework

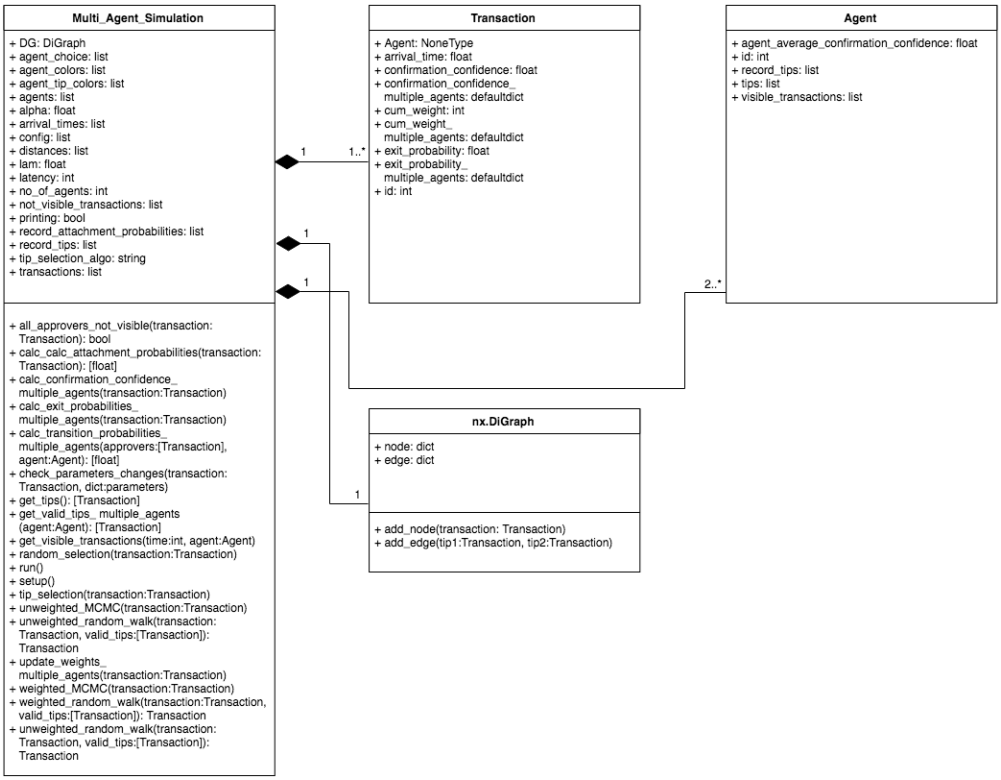

Figure A.1: A UML diagram showing the main classes of the multi-agent simulation framework. Note that we also show the nx.DiGraph class to stress that it is the main data structure for the Tangle-DAG (although it is part of the NetworkX package). Note that some other helper scripts, for instance for plotting and data analysis are not shown in this UML for simplicity.

Appendix B

Simplification of the transition probability function

As described in Chapter 2, the IOTA white paper [47] suggests a weighted random walk algorithm for tip selection, which uses a transition probability function to determine the chance by which the walker steps onto one of the approving transactions. We showed the suggested formula in 2.1, but noted that for the practical implementation a simplified version is needed, shown in 4.2.

The simplification works as follows:

$$P_{xy} = \frac{e^{-\alpha(W_x - W_y)}}{\sum_{z:z \rightsquigarrow x} e^{-\alpha(W_x - W_z)}} = \frac{e^{-\alpha W_x} e^{\alpha W_y}}{\sum_{z:z \rightsquigarrow x} e^{-\alpha W_x} e^{\alpha W_z}} = \frac{e^{-\alpha W_x} e^{\alpha W_y}}{e^{-\alpha W_x} \sum_{z:z \rightsquigarrow x} e^{\alpha W_z}} = \frac{e^{\alpha W_y}}{\sum_{z:z \rightsquigarrow x} e^{\alpha W_z}} \quad \text{(B.1)}$$

The result is both easier to understand and more convenient for the practical implementation and re-normalization. We do not deal with weight differences any more, but just with the weights of the approvers.

Appendix C

Effect of multiprocessing for parallelizing simulations

We used the Python `timeit` module and the following simulation parameters for a comparison of the runtime for 50 simulations:

```
Multi_Agent_Simulation(500, 25, 2, 0.005, 1, "weighted")
```

Running every simulation one after another resulted in a total running time of 439.056 seconds.

Using our multi-processing script this was significantly reduced to 56.543 seconds only, almost 8 times faster.

Appendix D

Two-agent simulation configuration file example

This is the content of an example configuration file `config.ini` for a two-agent simulation for the evaluation of attachment probabilities. The scenario is explained in Chapter 5.3.1. We use parameter change events as explained in Chapter 4.2.2.

```
[PARAMETERS]
no_of_transactions = 20000
lambda = 50
no_of_agents = 2
alpha = 0.001
latency = 1
distance = 50000
tip_selection_algo = weighted
agent_choice = [1.0,0.0]
printing = True


[EVENT1]
step = 10000
agent_choice = [0.7,0.3]


[EVENT2]
step = 15000
distance = 1
```

Appendix E

Distance matrix of sparsely connected agent network

This is an example of d for a network of sparsely connected agents, in particular the one illustrated in Figure 6.1. The matrix can be used as input parameter for a large multi-agent simulation. Scaling of the matrix has not been used for this illustration, but for most experiments presented in Chapter 6.

```
[[0, 2, 4, 1, 3, 2, 2, 1, 3, 3, 3, 3, 1, 3, 2, 2, 1, 2, 2, 2],
 [2, 0, 6, 3, 1, 4, 4, 1, 5, 5, 5, 5, 3, 5, 4, 4, 3, 4, 4, 4],
 [4, 6, 0, 5, 7, 6, 4, 5, 5, 7, 7, 1, 5, 5, 4, 2, 3, 6, 6, 6],
 [1, 3, 5, 0, 4, 1, 3, 2, 4, 2, 2, 4, 2, 4, 3, 3, 2, 1, 1, 1],
 [3, 1, 7, 4, 0, 5, 5, 2, 6, 6, 6, 6, 4, 6, 5, 5, 4, 5, 5, 5],
 [2, 4, 6, 1, 5, 0, 4, 3, 5, 1, 3, 5, 3, 5, 4, 4, 3, 2, 2, 2],
 [2, 4, 4, 3, 5, 4, 0, 3, 3, 5, 5, 3, 3, 3, 2, 2, 1, 4, 4, 4],
 [1, 1, 5, 2, 2, 3, 3, 0, 4, 4, 4, 4, 2, 4, 3, 3, 2, 3, 3, 3],
 [3, 5, 5, 4, 6, 5, 3, 4, 0, 6, 6, 4, 4, 2, 1, 3, 2, 5, 5, 5],
 [3, 5, 7, 2, 6, 1, 5, 4, 6, 0, 4, 6, 4, 6, 5, 5, 4, 3, 3, 3],
 [3, 5, 7, 2, 6, 3, 5, 4, 6, 4, 0, 6, 4, 6, 5, 5, 4, 1, 2, 3],
 [3, 5, 1, 4, 6, 5, 3, 4, 4, 6, 6, 0, 4, 4, 3, 1, 2, 5, 5, 5],
 [1, 3, 5, 2, 4, 3, 3, 2, 4, 4, 4, 4, 0, 4, 3, 3, 2, 3, 3, 3],
 [3, 5, 5, 4, 6, 5, 3, 4, 2, 6, 6, 4, 4, 0, 1, 3, 2, 5, 5, 5],
 [2, 4, 4, 3, 5, 4, 2, 3, 1, 5, 5, 3, 3, 1, 0, 2, 1, 4, 4, 4],
 [2, 4, 2, 3, 5, 4, 2, 3, 3, 5, 5, 1, 3, 3, 2, 0, 1, 4, 4, 4],
 [1, 3, 3, 2, 4, 3, 1, 2, 2, 4, 4, 2, 2, 2, 1, 1, 0, 3, 3, 3],
 [2, 4, 6, 1, 5, 2, 4, 3, 5, 3, 1, 5, 3, 5, 4, 4, 3, 0, 1, 2],
 [2, 4, 6, 1, 5, 2, 4, 3, 5, 3, 2, 5, 3, 5, 4, 4, 3, 1, 0, 2],
 [2, 4, 6, 1, 5, 2, 4, 3, 5, 3, 3, 5, 3, 5, 4, 4, 3, 2, 2, 0]]
```

Appendix F

Further results for scenario 2, Chapter 6

We run the same simulation configuration 20 times per α-value (0.001, 0.005 and 0.01).

Agent	**Closeness centrality** (descending order)	**Average attachment probability**		
		$\alpha = 0.001$	$\alpha = 0.005$	$\alpha = 0.01$
5	0.6	0.127	0.127	0.145
0	0.53	0.123	0.129	0.13
9	0.47	0.109	0.107	0.116
8	0.43	0.099	0.094	0.101
3	0.39	0.092	0.088	0.094
7	0.39	0.095	0.097	0.084
1	0.36	0.090	0.094	0.081
2	0.36	0.098	0.093	0.098
4	0.36	0.089	0.092	0.081
6	0.27	0.078	0.078	0.07

Table F.1: Results for 20 simulations per α-level: Average attachment probabilities for 10 agents over 10,000 transactions with scaling of d (factor 10). Closeness centrality and average attachment probability per agent rank the same for most agents.

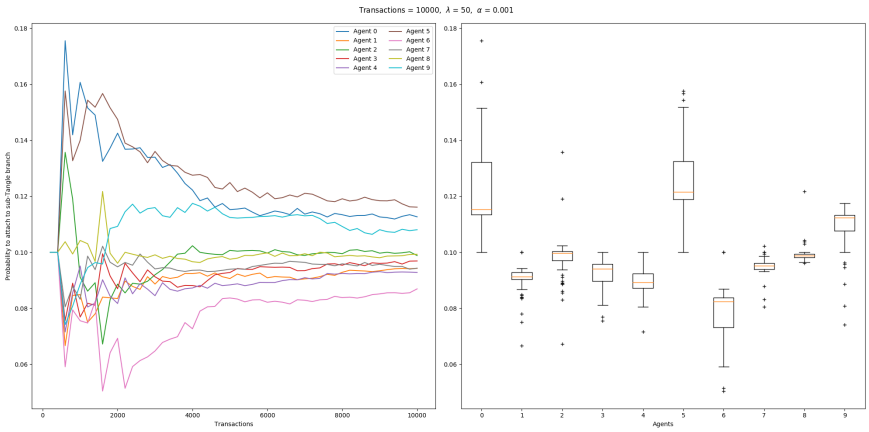

Figure F.1: Results for 20 simulations: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 10) and $\alpha = 0.001$.

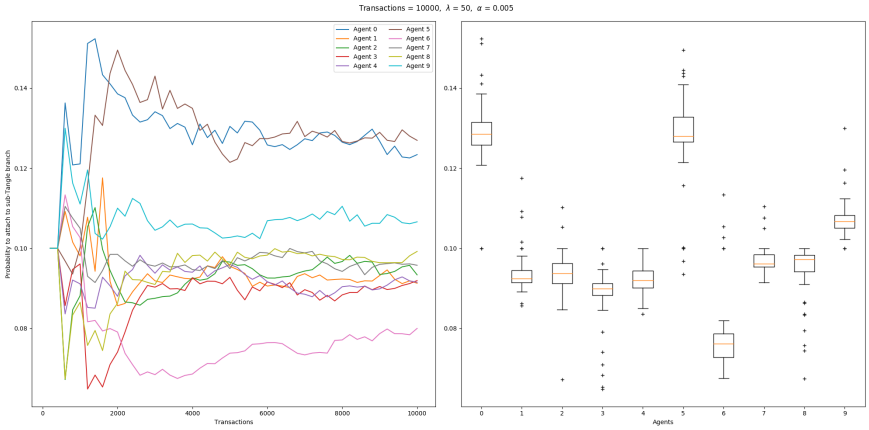

Figure F.2: Results for 20 simulations: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 10) and $\alpha = 0.005$.

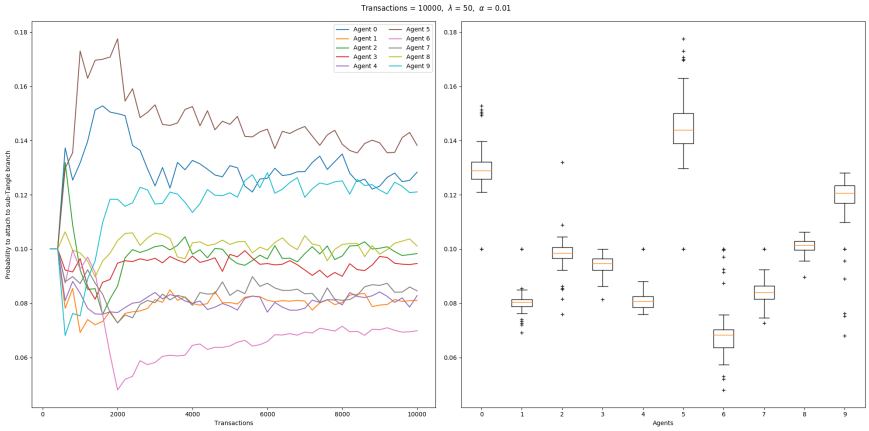

Figure F.3: Results for 20 simulations: Probabilities to attach to the sub-Tangle branches of 10 agents over 10,000 transactions with scaling of d (factor 10) and $\alpha = 0.01$.

Appendix G

Legal and ethical considerations

This dissertation does not involve any problematic legal and ethical considerations – consequently all questions in the following checklist have been ticked "no". This is mainly due to the fact that all software has been written from scratch and all data analysed has been created by this simulation software. The analyses are based on a theoretical distributed ledger model and do not have any (foreseeable) implications in practice.

LSEPI Checklist:

	Yes	**No**
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve human embryonic stem cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from "human embryos/foetuses" i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		✓
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓

If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?		✓
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
Section 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
Section 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

Appendix H

Glossary

Abbreviations and symbols for the multi-agent Tangle model and algorithms described in this dissertation, ordered by first appearance:

α	Alpha, factor used in calculation of transition probabilities, determines bias/randomness of MCMC algorithm
W_v	Cumulative weight of transaction v
$z \rightsquigarrow x$	Transaction z directly approves transaction x
λ	Lambda, average incoming transactions rate
h	Latency, average computational cost for tip selection, verification and POW plus network latency, assumed to be fixed to 1
$\mathcal{G}$	Set of all DAGs $G = (V, E)$
G	Genesis transaction
$\deg_{\text{out}}(v)$	Number of approved transactions by transaction v
$\mathcal{T}(t)$	Tangle at time t
$V(t)$	Set of all transactions (vertices) at time t
$E(t)$	Set of all edges at time t
A	Set of agents
d	Distance matrix, contains the shortest paths (distances) between all agents, represents network latency in multi-agent model
d_{ij}	Distance (shortest path) between agent i and agent j
$A(v)$	Issuing agent for transaction v
$T(v)$	Issuance time for transaction v
N	Number of agents in the multi-agent model
c	Probability vector for choosing an issuing agent for an incoming transaction (agent choice probabilities)
$V_i(t)$	Set of visible transactions for agent i at time t
$E_i(t)$	Set of visible edges for agent i at time t
$\mathcal{T}_i(t)$	sub-Tangle of agent i at time t
τ_1	Value of inter-arrival time for first transaction
at_v	Arrival time of transaction v

$I_i(t)$	Set of invisible transactions for agent i at time t
$T_i(t)$	Set of valid tips for agent i at time t
w	Walker (current transaction) in a random walk algorithm
P_{xy}	Transition probability to walk from transaction x to y
$W_i(x)$	Cumulative weight of transaction x for agent i
$f_i(v)$	Probability that a newly incoming transaction issued by agent i would select a *tip* v during tip selection
$P_i(v)$	Exit probability of a transaction v for agent i, that is the chance of walking to transaction v during a random walk (generalisation of $f_i(v)$ for *all* transactions)
$c_i^{(t)}(v)$	Confirmation confidence of a transaction v for agent i at time t
$\mathcal{F}_i^{(t)}(v)$	All transactions that directly or indirectly reference transaction v for agent i at time t (future set of v)
d_{max}	Maximum value in the distance matrix d
$L(t)$	Number of tips in a Tangle at time t, see [47], Section 3
L_0	Predictive value, around which the number of tips $L(t)$ fluctuates over time, see [47], Section 3
AP_i	Attachment probability, the chance to attach to (or to select during tip selection) a current tip issued by agent i

The following symbols represent set operations:

$\emptyset$	The empty set
$\in$	Is member of
$\notin$	Is not member of
$\subseteq$	Is subset of
$\cup$	Set union
$\setminus$	Set difference

Bibliography

[1] A. Back, "Hashcash - a denial of service counter-measure," 2002, accessed: 2018-09-04. [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf

[2] L. Baird, "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance," *Swirlds, Inc.Technical Report SWIRLDS-TR-2016*, vol. 1, 2016.

[3] T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten, "Have a snack, pay with bitcoins," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–5.

[4] A.-L. Barabási, *Network science*. Cambridge university press, 2016.

[5] A. Bavelas, "Communication patterns in task-oriented groups," *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[6] I. Bentov, A. Gabizon, and A. Mizrahi, "Cryptocurrencies without proof of work," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 142–157.

[7] I. Bentov, P. Hubácek, T. Moran, and A. Nadler, "Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies." *IACR Cryptology ePrint Archive*, vol. 2017, p. 300, 2017.

[8] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake." *IACR Cryptology ePrint Archive*, vol. 2016, p. 919, 2016.

[9] X. Boyen, C. Carr, and T. Haines, "Blockchain-free cryptocurrencies: A framework for truly decentralised fast transactions," *Blockchain-free cryptocurrencies: A framework for truly decentralised fast transactions*, 2016.

[10] Q. Bramas, "The stability and the security of the tangle," 2018.

[11] V. Buterin, "A next-generation smart contract and decentralized application platform," *White Paper*, 2014. [Online]. Available: http://www.the-blockchain.com/docs/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf

[12] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[13] A. Churyumov, "Byteball: A decentralized system for storage and transfer of value," 2016, accessed: 2018-09-04. [Online]. Available: https://byteball.org/Byteball.pdf

[14] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. G. Sirer, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[15] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[16] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–10.

[17] ——, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[18] P. Erdos and A. Rényi, "On the evolution of random graphs," *Publ.Math.Inst.Hung.Acad.Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[19] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.

[20] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social networks*, vol. 1, no. 3, pp. 215–239, 1978.

[21] L. Freiberg, "What's up with the Tangle?" 2018, accessed: 2018-09-04. [Online]. Available: https://medium.com/@lewisfreiberg/whats-up-with-the-tangle-d825c692e7a8

[22] A. Gal, "Algorithm for calculating cumulative weights," 2018, accessed: 2018-09-04. [Online]. Available: https://github.com/alongalky/iota-docs/blob/master/cumulative.md

[23] ——, "Alpha: playing with randomness," 2018, accessed: 2018-09-04. [Online]. Available: https://blog.iota.org/alpha-d176d7601f1c

[24] A. Gal and B. Kuśmierz, "Probability of being left behind and probability of becoming permanent tip in the tangle v0.2," 2018, accessed: 2018-09-04. [Online]. Available: https://www.iota.org/research/academic-papers

[25] A. Gal and C. Shikhelman, "Partitioning in the Tangle: a Multi-Agent Extension," 2018, unprinted, accessed: 2018-09-04.

[26] Gartner Inc., "Leading the IoT: Gartner Insights on How to Lead in a Connected World," 2017, accessed: 2018-09-04. [Online]. Available: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf

[27] E. Greve, "What's Next: Current IOTA R&D Projects ," 2018, accessed: 2018-09-04. [Online]. Available: https://blog.iota.org/whats-next-current-iota-r-d-projects-eeb8cc03adb5

[28] IOTA Foundation, "Iota docs: Consensus," 2018, accessed: 2018-09-04. [Online]. Available: https://docs.iota.org/introduction/tangle/consensus

[29] ——, "Iota docs: Tip selection," 2018, accessed: 2018-09-04. [Online]. Available: https://docs.iota.org/introduction/tangle/tip-selection

[30] A. Kiayias and G. Panagiotakos, "Speed-security tradeoffs in blockchain protocols." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1019, 2015.

[31] ——, "On trees, chains and fast transactions in the blockchain." *IACR Cryptology ePrint Archive*, vol. 2016, p. 545, 2016.

[32] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[33] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 279–296.

[34] B. Kusmierz, "The first glance at the simulation of the tangle: discrete model," 2017, accessed: 2018-09-04. [Online]. Available: https://www.iota.org/research/academic-papers

[35] B. Kusmierz, P. Staupe, and A. Gal, "Extracting tangle properties in continuous time via large-scale simulations," 2018, accessed: 2018-09-04. [Online]. Available: https://www.iota.org/research/academic-papers

[36] S. Kutten and D. Peleg, "Asynchronous resource discovery in peer to peer networks," in *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on.* IEEE, 2002, pp. 224–231.

[37] S. D. Lerner, "Dagcoin: a cryptocurrency without blocks," 2015.

[38] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *International Conference on Financial Cryptography and Data Security.* Springer, 2015, pp. 528–547.

[39] J. D. Licciardello, "The first cohort of ecosystem development fund grantees," 2018, accessed: 2018-09-04. [Online]. Available: https://blog.iota.org/the-first-cohort-of-ecosystem-development-fund-grantees-e9da89ecfb56

[40] J. Lind, I. Eyal, P. Pietzuch, and E. G. Sirer, "Teechan: Payment channels using trusted execution environments," *arXiv preprint arXiv:1612.07766*, 2016.

[41] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2016, pp. 17–30.

[42] Matplotlib developers, "Matplotlib documentation," 2018, accessed: 2018-09-04. [Online]. Available: https://matplotlib.org/

[43] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, accessed: 2018-09-04. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[44] NetworkX developers, "Networkx documentation," 2018, accessed: 2018-09-04. [Online]. Available: https://networkx.github.io/

[45] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2017, pp. 643–673.

[46] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," *draft version 0.5*, vol. 9, p. 14, 2016.

[47] S. Popov, "The tangle," 2016, accessed: 2018-09-04. [Online]. Available: https://www.iota.org/research/academic-papers

[48] S. Popov, O. Saa, and P. Finardi, "Equilibria in the tangle," *arXiv preprint arXiv:1712.05385*, 2017.

[49] S. Ross, *A First Course in Probability 8th Edition.* Pearson, 2009.

[50] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, no. 4, pp. 581–603, 1966.

[51] Y. Sompolinsky and A. Zohar, "Phantom: A scalable blockdag protocol," 2018.

[52] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: Serialization of proof-of-work events: confirming transactions via recursive elections," 2016.

[53] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing: Fast money grows on trees, not chains," 2013.

[54] ——, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[55] Team Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies," 2018.

[56] Twitter, accessed: 2018-09-04. [Online]. Available: https://twitter.com/JoshuaJBouw/status/1015586862335815680

[57] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014. [Online]. Available: http://yellowpaper.io