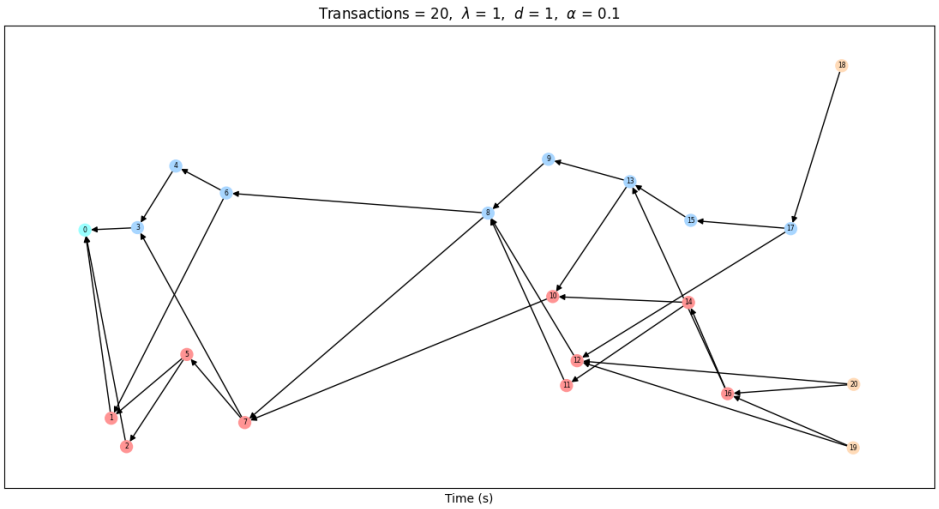
DAGsim Parameters:

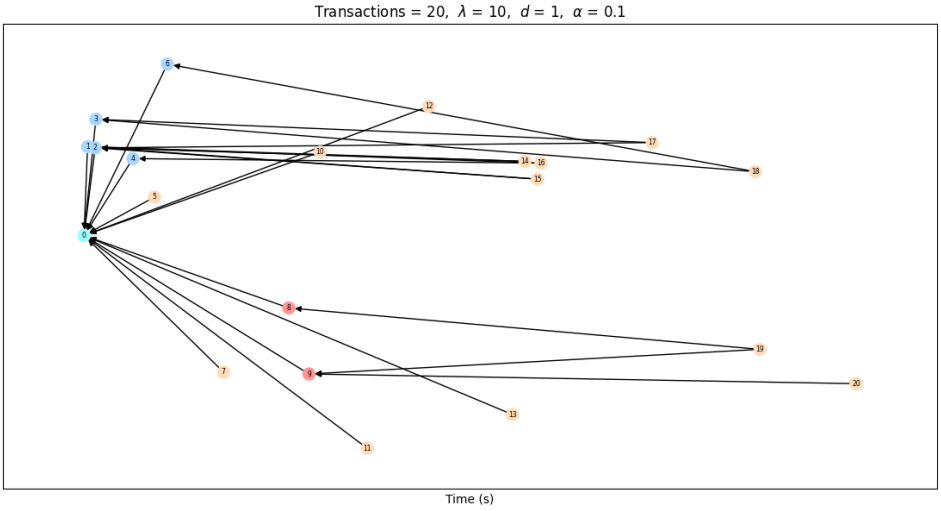
no_of_transactions: defines the number of transactions to be processed during the simulation. It will influence the size and complexity of the generated TANGLE.

no_of_agents: defines the number of nodes in the network that participate in the issuance and validation of transactions.

Lambda (λ): defines the frequency at which transactions enter the system. A high λ generates transactions faster.

Example with lambda = 1 vs lambda = 10:



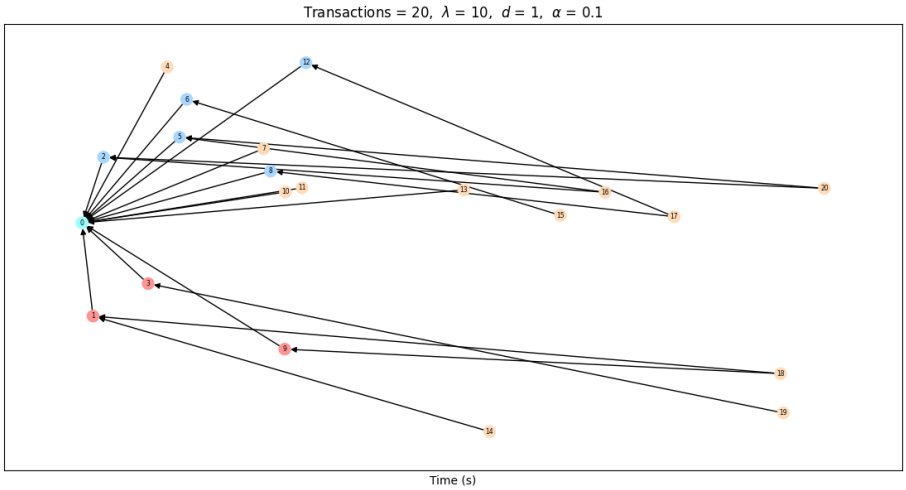


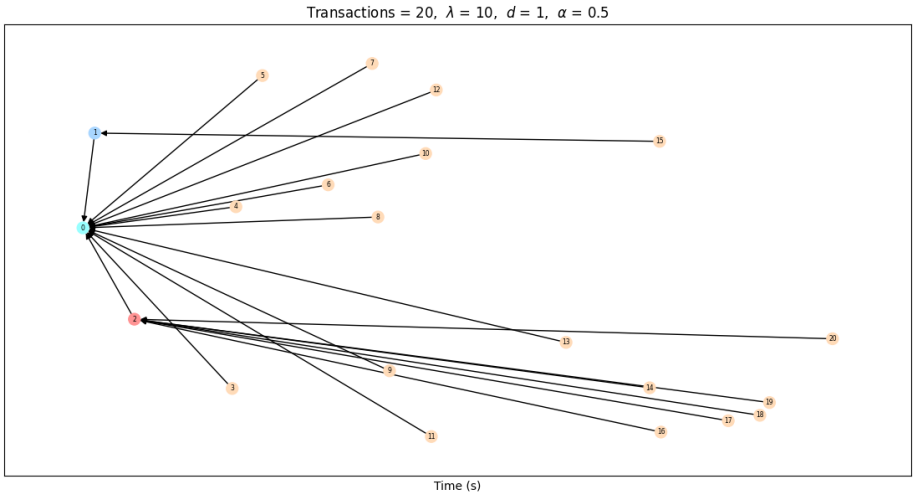
We can see that with lambda = 10, transactions are generated rapidly and have multiple opportunities to connect to various previous transactions. In the case of lambda = 1, transactions occur less frequently, allowing the Tangle to develop more slowly and with fewer options for each new transaction to connect to multiple previous transactions.

According to the article, it follows a Poisson distribution. I quote, "The arrival times of incoming transactions are sampled from the exponential distribution according to the incoming transaction rate, which follows a Poisson process." Lambda defines how many transactions are expected to arrive per unit of time. For example, if it is 10, 10 transactions are expected per unit of time. Being Poisson, sometimes more or fewer transactions than expected may arrive.

Alpha (α): It serves for randomness in the tip selection algorithm. With a high Alpha, it seems to be more deterministic, and with a low Alpha, it seems to be more random.

Example with Alpha = 0.1 vs Alpha = 0.5





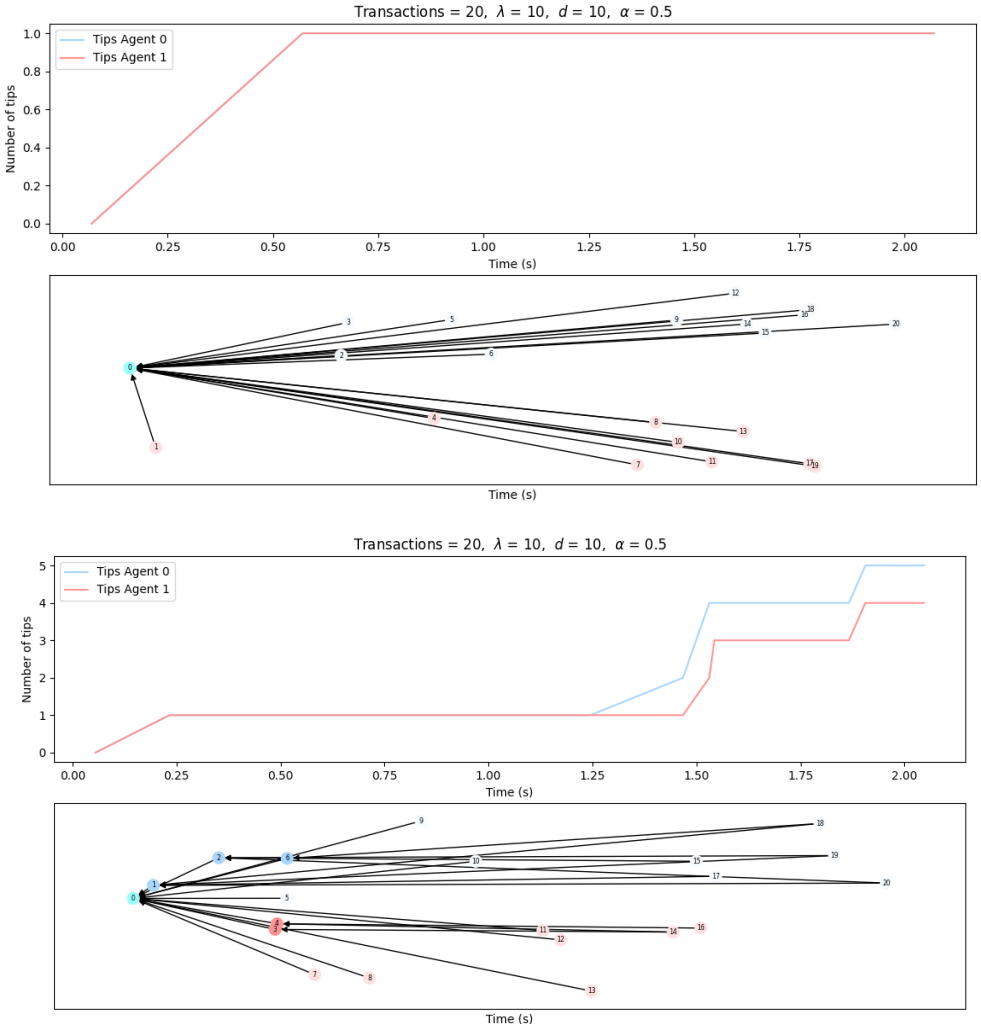
The relationship between the tip selection method and the alpha (α) parameter in the IOTA Tangle simulation is crucial, especially when the tip selection algorithm is weighted:

- **Random:** Does not use the α parameter, as it simply chooses tips randomly among visible and valid transactions.
- **Unweighted MCMC (Monte Carlo Markov Chain):** This method also ignores the α parameter because it performs random walks without weighting transactions by their cumulative weight.
- **Weighted MCMC:** This is where α comes into play. In this method, α directly affects how transactions are weighted during the random walk. Specifically, it is used to adjust the transition probability between transactions in the random walk, making the probability of selecting certain tips proportional to their cumulative weight, adjusted by α.

Therefore, Alpha is an adjustment factor that modulates the influence of a transaction's cumulative weight during tip selection in the weighted algorithm. A higher α increases the likelihood that transactions with greater weight will be selected as tips, which can favor faster consensus convergence in the network but may also increase graph centralization as heavier transactions dominate tip selection.

Latencia : It refers to the communication delay between nodes. In the article, latency is always 1.

Example latency 10 vs latency 1:



With latency 10, we can see that during most of the simulated time, Agent 1 has a very low number of tips, while Agent 0 has no tips directly. This indicates that high latency may be preventing Agent 1's transactions from being seen and selected as tips by other agents in a timely manner. If we look at the Tangle, Agent 1's transactions appear to be dominant, meaning almost all transactions confirm at least one transaction from Agent 0.

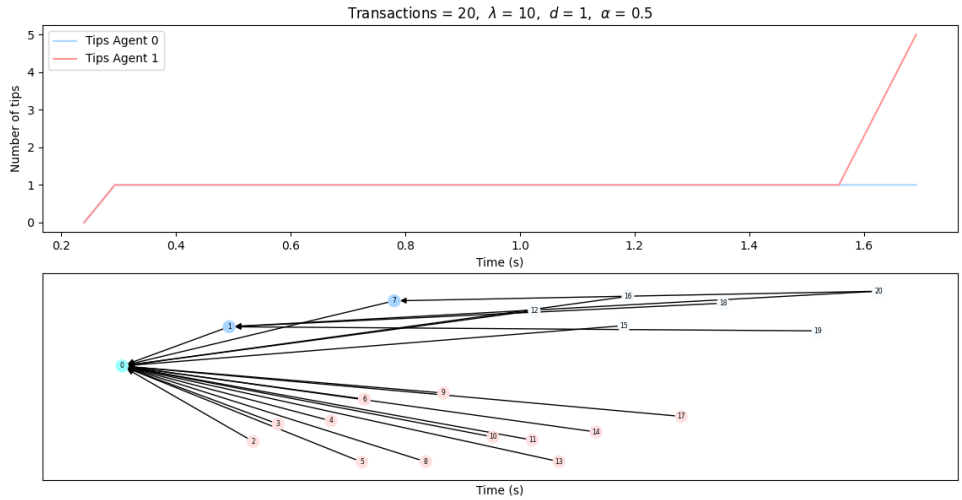
With low latency, the number of tips for both agents varies more uniformly over time. Agent 1 shows an increase in the number of tips towards the middle and end of the simulated period, suggesting that its transactions are being seen and selected more frequently by other agents, and Agent 0 also shows a more uniform increase. Now, if you look at the Tangle, it seems that both are contributing to the network and that Agent 0's transactions can confirm Agent 1's transactions and vice versa.

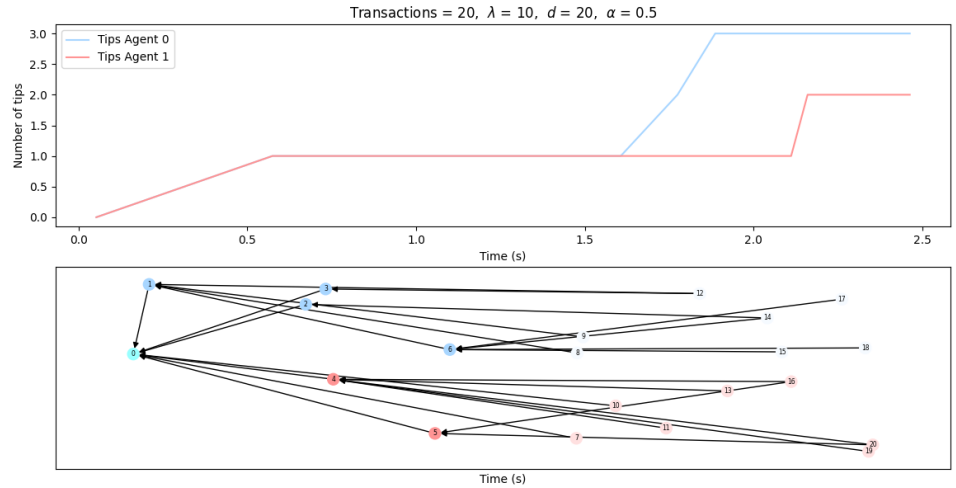
Latency represents the network delay time for a transaction to be visible to other nodes or agents. Latency affects when a transaction is considered "visible" to other nodes, meaning when it has arrived and thus can be referenced or confirmed by other transactions. With lambda, arrival times are generated for each transaction. During the processing of each transaction, it appears that the simulator checks if the transaction is visible to other nodes at a specific moment in time. It compares the arrival time of a new transaction (adjusted for latency) with the arrival times of previous transactions. If the arrival time of the previous transaction plus the latency is <= the arrival time of the new transaction, then the previous transaction is visible and can be "referenced" or "validated," so to speak.

Latency is measured in the same time units as arrival times, in seconds. This latency is added to the arrival times, as I mentioned earlier, to determine when a transaction is considered visible. For example, if a transaction arrives at the 5th second and the latency is 1 second, this transaction will not be visible to other transactions arriving before the 6th second.

Distance (d): distance between agents.

Example with distance = 1 y distance = 10





We can observe the tips that agents manage to have active over a period of time. With shorter distances between agents, we see a stable growth in tips and a more stable TANGLE. With longer distances, the dominance of one agent over another can reoccur, affecting the Tangle.

A distance matrix is used to represent the latency between each pair of agents based on the distance. In the code, it seems that the distance simply determines which transactions are visible to a given agent by manipulating that latency. The matrix establishes how the different agents are connected to each other and how long it takes for information from one agent to reach another. Each entry in the matrix indicates the delay (or latency) between a pair of agents. It also seems that the matrix is used to choose the optimal path between two agents. This is how I understand it from reading the article.

```
else:
    #Get distance from agent to agent of transaction from distance matrix
    distance = self.distances[agent.id][transaction.agent.id]

    #Determine if the transaction is visible (incoming_transaction.arrival_time determines current time
    if (transaction.arrival_time + self.latency + distance <= incoming_transaction_time):

        agent.visible_transactions.append(transaction)

    #Record not visible transactions for 'current agent' only (reduces overhead)
    elif(incoming_transaction_agent == agent):
        self.not_visible_transactions.append(transaction)
```

tip_selection_algo: Define the method used to select which previous transactions a new transaction will approve. We can choose between "random", "weighted", and "unweighted" (random, considering their weights, or not considering the weights).

Example weighted vs random: