“Analysis of IOTA”

Mohammadmasoud Shabanijou, Arezoo Sajadpour
(Blockchain Data Analytics, University of Manitoba, Fall 2020)

Contents

1 What did we do?

We studied and research regarding:

- What is the IOTA network to have an overall insight on this novel cryptocurrency network
- How to collect transaction data to create the transaction graph.
- Analysing each component of IOTA address
- What is tangle graph

Following are some references that we studied:

- Video series by Robert Lie in which different topics will be explained to understand IOTA
- Chapter 12 of Blockchain by Akcora et al. [?]
- The Tangle, a paper by Popov [4]
- IOTA website to extract data and further information

In the next section, a summary of the IOTA address and graphs is provided.

2 Introduction

IOTA is the first network for exchanging value between humans and machines, and this is the first ledger that is built for the "Internet Of Things" (IOT) in 2015. Indeed, IOTA Foundation's main focus is IOT and the Machine Economy, but this technology is considered for payments between humans as well.

Once an IOTA transaction is sent, it validates two other transactions. So that, IOTA overcomes the cost and scalability limitations of blockchain.

3 IOTA Network: IOTA Address

Before going through IOTA network graph, first we explain how IOTA address is created. In IOTA network, we use trits of three values instead of bites that take values 0 and 1. Indeed, IOTA uses "seed" which each seed consists of a string of 81 characters and generated from a "tryte". Trinary numeral system has two types where we only consider the balance trinary system in which a trit has the value -1, 0, 1. Trit means trynary digit and tryte is three trits. Since a tryte has 3 trits, the maximum value will be $(3^3 - 1)/2 = 13$ and it has $3^3 = 27$ combinations. A tryte will have the following values: $-13, -12, \ldots -2, -1, 0, 1, 2, ...12, 13$. The alphabet of balance trinary system make the trytes more human readable. This alphabet consists of 26 letters in uppercase, and only number 9 like, 9ABCDEFGHIJKLMNOPQRSTUVWXYZ. So, it has 27 characters. Since one tryte has $3^3 = 27$ combinations, each tryte can be represented by one character in the tryte alphabet which makes it easy to read. Tryte alphabet is as follow [3][1]:

[1]https://www.youtube.com/watch?v=MsaPA3U4ung

Tryte	Dec	Char	Tryte	Dec	Char
0 0 0	0	9	-1 -1 -1	-13	N
1 0 0	1	A	0 -1 -1	-12	O
-1 1 0	2	B	1 -1 -1	-11	P
0 1 0	3	C	-1 0 -1	-10	Q
1 1 0	4	D	0 0 -1	-9	R
-1 -1 1	5	E	1 0 -1	-8	S
0 -1 1	6	F	-1 1 -1	-7	T
1 -1 1	7	G	0 1 -1	-6	U
-1 0 1	8	H	1 1 -1	-5	V
0 0 1	9	I	-1 -1 0	-4	W
1 0 1	10	J	0 -1 0	-3	X
-1 1 1	11	K	1 -1 0	-2	Y
0 1 1	12	L	-1 0 0	-1	Z
1 1 1	13	M			

IOTA wallet can generate addresses by using seed, and each specific seed generates addresses belonging to it. One possible way to create a seed is to use a terminal on Linux or Mac Operating System. In order to use this method enter:
cat /dev/urandom —tr -dc A-Z9—head -c$1:-81
cat /dev/urandom —LCALL=C tr -dc ’A-Z9’ — fold -w 81 — head -n 1
on Linux and Mac OS, respectively. Dev/urandom function generates cryptographically random numbers by gathering random data.

There is another way to generate IOTA seeds, and this is to use a web application. Note that you should never create an IOTA seed using a web application while you are online.

IOTA address, however, needs another component, which is called “index”. Index is a value between 0 to 9,007,199,254,740,991. For example, address 0 has index 0, and address 1 has index 1. Index leads to generate multiple addresses of the same seed. The decimal index number should be converted to trits. Then, IOTA uses the KERL hash function and creates “subseed,” which is the hash of index + seed. IOTA provides three security levels, which determine the number of rounds for hashing. The default security level is 2, which produces a private key of 4374 trytes. Using a security level of 2 and 3 increase the signature length, which leads to an inefficiency problem. Address created in security level two resulted in a signature with a double length of the size of “signatureMessageFragment” field. Therefore, an empty transaction will be created to carry the signature. These two transactions related to the same address are connected in a data structure called “bundle”.

A “checksum” is an additional component that consists of 9 trytes which can be added to the address to validate the address. An address with checksum has 90 trytes.

4 IOTA Network: IOTA Tangle Graph

There is no block or miner in the IOTA network, which is the main difference between IOTA and Bitcoin. Instead of the global blockchain, there is a “Directed Acyclic Graph” (DAG) in which nodes and edges represent transactions are created once a node is present by making a connection with two previous transactions. To be more precise, when a new transaction arrives, it creates a list of two already unconfirmed existing transactions. The unconfirmed transactions in the tangle graph are called “tips,” and this process is called tip selection, which leads to security in the IOTA network.

There is no need that any relations should exist between the new transaction and its selected tip. Note that a transaction might be chosen as a tip several times by different recent transactions, and in this way, trunk and branch are confirmed. However, there is still a chance of a double-spending attack if two transactions spend the same IOTA coins. To overcome this issue, IOTA creates "milestone transactions" in every minute, which marks double-spending nodes.

5 An Introduction to Hornet

So the next step we took was finding a good platform to connect to the IOTA tangle to create the IOTA address graph. In order to do that, we chose Hornet. Hornet is an excellent API for working with iota data because it comes with a lot of advantages. It is very lightweight, and it enables the users to run their own node. The users receive some advantages by running their own nodes:

- The users have direct access to an IOTA network instead of having to connect to and trust someone else's node.
- The users help the IOTA network become more distributed by validating the transactions in your IOTA network.

However, there are some limitations with regards to this API as well. Users cannot create or sign transactions. For such purposes, the users must use client software such as Trinity or a core client library and send the transactions to a node[2]. Hornet's source code is available on Github.
In the next section, we begin by explaining in detail how we worked with this API. For further information on Hornet, please refer to this link: `https://docs.iota.org/docs/hornet/1.1/overview\`

6 Creating the IOTA address graph

In the following, we aim to create the address graph by using extracted transactions from the previous section. Note that the order of transactions indicates how they are linked to each other. As mentioned in the introduction, once a transaction is created, it should approve two invalidated transactions and connect to them. Hence, the given text file, which includes transactions' hash address, illustrates both nodes and edges of the address graph. There are 1000 transactions in the given list, so there should be 1000 nodes, which are indicated by blue. To find the graph's edges, we create a pool of milestone transactions and extract their branch and trunk transactions in the second pool. The following code is used to receive milestone transactions:

Listing 1: Insert code directly in your document

```
import urllib2
import json

#with this command we intend to get the node information
command = {"command": "getNodeInfo"}
#in this part a connection to the API will be established with json format
stringified = json.dumps(command)

headers = {
    'content-type': 'application/json',
    'X-IOTA-API-Version': '1'
}

request = urllib2.Request(url="http://localhost:14265", data=stringified, headers=headers)
returnData = urllib2.urlopen(request).read()

jsonData = json.loads(returnData)
#we get the latestMilestone hash
a = jsonData["latestMilestone"]
#then, print it
print a
#we write it to a text file to use it in the next steps
text_file = open("Output.txt", "w")
text_file.write("%s" % a)
text_file.close()
#Refrence: https://docs.iota.org/docs/hornet/1.1/references/api-reference#findtransactions
```

As the next step, we considered the latest milestone transaction as the origin of the graph and found all its tips. Then, for each tip, we find all its adjacent nodes and store them in a file.

Listing 2: Insert code directly in your document

```
import urllib2
import json

#Reading the latest milestonetransaction from the text file
file = open("Output.txt")
ms = file.read()

file.close()


ms = ms[:81]
```

```
print("latest␣milestonetransaction:")
print ms
#Finding the tip transactions connected to the latest milestone transaction
command = {
  "command": "findTransactions",
  "approvees": [
    ms
  ]
}

stringified = json.dumps(command)

headers = {
    'content-type': 'application/json',
    'X-IOTA-API-Version': '1'
}
#Establishing connection to the API
request = urllib2.Request(url="http://localhost:14265", data=stringified, headers=headers)
returnData = urllib2.urlopen(request).read()

jsonData = json.loads(returnData)

print("\n␣tips:")

print jsonData["hashes"]

#We write it to a text file to use it in the next steps
text_file = open("tips.txt", "w")
text_file.write("\nMilestonetransaction:%s" % ms)
text_file.write("\n␣%s" % jsonData["hashes"])



length = len(jsonData["hashes"])

print length

y = length

for x in range(0, 100):

#Next step is finding tip transactions of those tips
    command = {
  "command": "findTransactions",
  "approvees": [
    jsonData["hashes"][0]
  ]
}
    stringified = json.dumps(command)

    headers = {
    'content-type': 'application/json',
    'X-IOTA-API-Version': '1'
}
 #we write it to a text file to use it in the next steps
    text_file.write("␣\n␣tip:%s" % jsonData["hashes"][0])

    s = jsonData["hashes"][0]
 #Establishing connection to the API
    request = urllib2.Request(url="http://localhost:14265", data=stringified, headers=headers)
    returnData = urllib2.urlopen(request).read()

    jsonData = json.loads(returnData)

    print jsonData["hashes"]

    length = len(jsonData["hashes"])

    for p in range(0,length-1):
        text_file.write("_\n_tip:%s" % s)
#we write it to a text file to use it in the next steps
        text_file.write("_conneceted_tips:%s" % jsonData["hashes"][p])
#Closing the file that we used for storing transactions
text_file.close()
```

The output of the given node creates a list of edges, which is stored in a ".txt" file. Note that to have a presentable graph, we consider only 100 transactions, not the entire transactions (list of edges is attached). However, to make the output proper as an input of the "readedgelist" function, we prepossessed the edge list and converted it to a 2-column ".tex" file. Then, by using the "Networkx" package to visualize the address graph. The output of this code is saved as a "PNG". The image of a graph is in the appendix (Figure 1).

Listing 3: Insert code directly in your document

```
#References: https://deparkes.co.uk/2018/04/09/python-networkx-load-graphs-from-file/
#Here the list of edges is given to read_edgelist function to create a graph.
import networkx as nx
import csv
from matplotlib import pyplot as plt
g=nx.read_edgelist(r"C:\Users\sajadpoa\Documents\Education\3\Blockchain\Project\sample.txt",
create_using=nx.Graph(), nodetype=None)
print (nx.info(g))
nx.draw(g, node_size=100)
nx.draw(g)
#This part is aded to print the output as png file
plt.savefig(r"C:\Users\sajadpoa\Documents\Education\3\Blockchain\Project\Graph.png", format="PNG")
plt.show()
```

7 Address Graph Analysis

To analysis the address graph, we considered various metrics used in [1]. This address graph is based on a sample of 100 nodes, and the number of edges in this sample of the graph is 174. The average degree which is simply the average number of edges per node in the graph, which is approximately 2. To calculate the average degree, we took advantage of "nx.info()" in "Networkx".

8 Analysis of UTXO graph

In this part of the project, we intend to construct the UTXO graph of the IOTA transactions that we got from the API. In order to do that, we needed to access the balance of each transaction. So what we did in the next step was that we used this code to access the transaction balances. So we wrote the transactions on a text file. Below you could see the code for this part.

Listing 4: Code for writing transactions on a file

```python
#References:https://docs.iota.org/docs/hornet/1.1/references/api-reference#findtransactions
import urllib2  #Adding the necessary libraries
import json
file = open("Output.txt")  #Reading the latest milestonetransaction from the text file
ms = file.read()
file.close()
text_file = open("balances.txt", "w")
def listToString(s):    #Function for converting list to string
#https://www.geeksforgeeks.org/python-program-to-convert-a-list-to-string/
    # initialize an empty string
    str1 = ""
    # traverse in the string
    for ele in s:
        str1 += ele
    # return string
    return str1
ms = ms[:81]
print("latest milestonetransaction:")
print ms
command = {
 #Finding the tip transactions connected to the latest milestone transaction
  "command": "findTransactions",
  "approvees": [
    ms
  ]
}
stringified = json.dumps(command)
headers = {
    'content-type': 'application/json',
    'X-IOTA-API-Version': '1'
}
request = urllib2.Request(url="http://localhost:14265", data=stringified, headers=headers)
#Establishing connection to the API
returnData = urllib2.urlopen(request).read()
jsonData = json.loads(returnData)
print("\n tips:")
print jsonData["hashes"]
text_file = open("balances.txt", "w")
#We write it to a text file to use it in the next steps
length = len(jsonData["hashes"])
print length
y = length
for x in range(0, 100):

    command = {
  "command": "findTransactions",    #Next step is finding tip transactions of those tips
  "approvees": [
    jsonData["hashes"][0]
  ]
}
    stringified = json.dumps(command)
    headers = {
    'content-type': 'application/json',
    'X-IOTA-API-Version': '1'
}
 #we write it to a text file to use it in the next steps
    text_file.write("␣\n␣%s" % jsonData["hashes"][0])
    request = urllib2.Request(url="http://localhost:14265", data=stringified, headers=headers)
#Establishing connection to the API
    returnData = urllib2.urlopen(request).read()
    jsonData = json.loads(returnData)
text_file.close()
```

After that, we started to get the balances of each transaction in the next part. One problem that we faced in this section was that the command that the API provided was not compatible with loops, so we had to manually enter all the transactions in the command and receive the result.

Listing 5: Code for accessing the balances

```
#References:https://docs.iota.org/docs/hornet/1.1/references/api-reference#findtransactions
#Here the list of edges is given to read_edgelist function to create a graph.
import urllib2                    # Adding the necessary libraries
import json
command = {
  "command": "getBalances",            # Command for getting the balances
  "addresses": [
    "DE9DVSOWIIIKEBAAHCKBWNXGXTOKVLZPLRAGKZG9GXKFRFWERKBFYMPRLAGVZTRVYPEPHBMUPDMRQ9DPZ"
#
  ]
}
stringified = json.dumps(command)

headers = {
    'content-type': 'application/json',  # Establishing connection to the API
    'X-IOTA-API-Version': '1'
}
request = urllib2.Request(url="http://localhost:14265", data=stringified, headers=headers)
returnData = urllib2.urlopen(request).read()

jsonData = json.loads(returnData)

print jsonData                    # Printing the balance
```

The results that we received after doing these steps were very interesting. All of the transactions had a balance of zero value. We still do not know what might be the reason behind this fact, but this was one of the most interesting observations that we made during the project's implementation.

9 Appendix

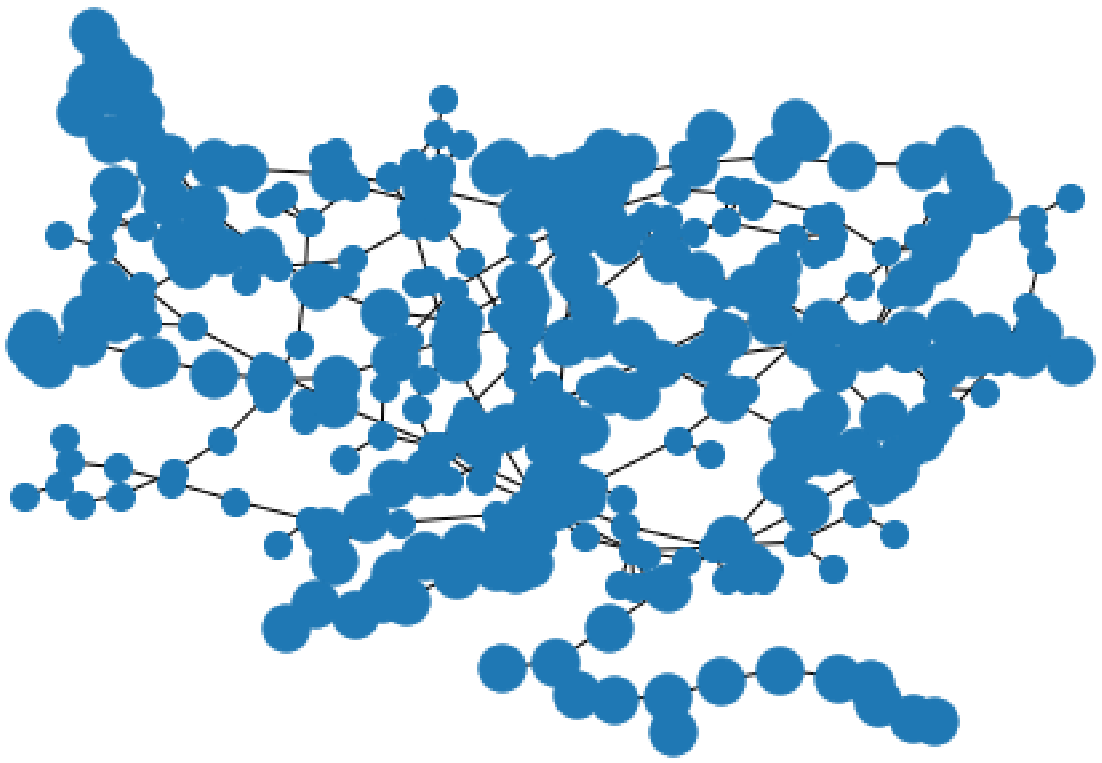

Figure 1: Address IOTA Graph of 100 nodes

References

[1] T. Chen, Y. Zhu, Z. Li, J. Chen, X. Li, X. Luo, X. Lin, and X. Zhange. Understanding ethereum via graph analysis. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1484–1492, 2018.

[2] I. documentation. Iota documentation.

[3] R. Lie. Iota tutorial.

[4] S. Popov. The tangle. *cit. on*, page 131, 2016.