An Efficient Graph-Based IOTA Tangle Generation Algorithm

†‡Fengyang Guo, †Xun Xiao, †Artur Hecker and ‡Schahram Dustdar
†*Munich Research Center, Huawei Technologies, Munich, Germany*
‡*Distributed Systems Group, TU Wien, Vienna, Austria*

***Abstract*—IOTA is a recent distributed ledger technology that relies on Directed Acyclic Graph (DAG) for its ledger organization. To improve IOTA mechanisms, the state of the art methodology employs graph analysis and, for that, heavily relies on synthetic graph generation. Herein, the most popular generation method simulates IOTA protocol execution. Although this method produces realistic IOTA ledgers, it requires too much memory and time due to repeated random walks on the DAG. In this paper, we propose an alternative Graph Generation and Refinement (GraGR) algorithm designed to generate realistic IOTA ledgers while strongly relaxing memory and timing constraints. The evaluations show that, compared to the state of the art, GraGR can generate a ledger with the same properties with only half of memory and up to 10 times faster.**

***Index Terms*—IOTA Blockchain Network, Network Modeling, Distributed Ledger System, IoT**

I. Introduction

Blockchain is a popular technology that features a decentralized and immutable data ledger [1] with a distributed consensus mechanism. It shows huge potential in areas, where several independently operating authorities work together, such as finance, supply chain management and Internet of Things (IoT). However, most current blockchain or *distributed ledger technologies* (DLT) exhibit some weaknesses [2], such as limited transaction processing speed. In the traditional blockchain it is caused, among others, by the data structure choice: since a chain offers exactly one block that new transactions (or a new block) can be "attached" to, to achieve consistency, either the number of nodes upholding this particular block has to be limited, or complex agreement/consensus between all such nodes at the moment of attachment is required. Both effects limit the transaction throughput [3].

A solution to improve the scalability of DLT is to use a Directed Acyclic Graph (DAG) data structure. Among the DAG-based DLTs, one of the most recent and prominent is IOTA [4]. Here, the incoming transactions can be attached to any leaf node (called "tip"), promising an increased transaction processing speed. However, the speed depends on several factors: an empirical analysis of an operational IOTA data structure (called "tangle") reveals that the actual processing speed is not as high as expected [5], pinpointing the relative complexity of the tangle as the main challenge area. To address this and to improve the performance, graph topology analysis becomes key, as it allows to develop better-suitable, faster algorithms for transaction processing.

For a comprehensive graph analysis, many sample tangles are required. Such samples can be obtained from either real or synthetic data. The problem with real data is limited diversity and availability. The main problem with generated tangles is realism. The common methodology is to follow IOTA protocol for arriving transactions: concretely, after an initiation to a single-vertex DAG, the generator, e.g., IOTA node binary, is subsequently fed with incoming transactions, either taken from a recorded trace or from a stochastic arrival process. Per default, IOTA employs random walk on the transpose graph of its DAG, i.e., from the root to the tips for each incoming transaction, i.e., for each step in the transpose graph, the walker calculates the transition probability for all candidate attachment points. Alas, this transition probability cannot be reused, because the cumulative weight and edge weight change, once the incoming message is attached to the tangle. In a nutshell, this method keeps the whole tangle data structure in memory, including cumulative weight of each message and, potentially, additional graphical information. We refer to these approaches as *Protocol Simulation based Generator* (ProSG). Albeit delivering realistic IOTA tangles, ProSG is not optimized for efficient research data generation and requires a lot of time and memory [6], in particular under a bursty message arrival. Hence, a more efficient tangle generator is crucial to streamline research and development activities.

In this paper, we propose a novel *Graph Generation and Refinement algorithm* (GraGR) for realistic IOTA tangle generation. GraGR does not need to calculate the transition probability at each step. Instead, with additionally provided, expected in- and out-degree distributions, GraGR can generate a representative IOTA tangle while limiting memory and timing requirements. Our main contributions are:

- We propose the first IOTA tangle generation algorithm that does not rely on costly random walk approaches.
- We conduct a comprehensive evaluation and analysis of the performance of GraGR in comparison to ProSG type of methods and show that GraGR generation is both correct and efficient.

In the following, we review existing IOTA tangle generation methods in Section II. Section III introduces IOTA preliminary as a background and presents GraGR; after that, Section IV compares ProSG-type generation to tangle generation with GraGR. Finally, Section V concludes this paper.

II. Related Work

There are two known ways to obtain IOTA tangles to analyze and improve IOTA system performance. Because they only differ in the used dataset, but rely on the IOTA protocol for message processing, we generally refer to these both methods as ProSG.

The first is to use a synthetic message sequence under IOTA typical message processing. One of such methods is *TangleSimulator*[1]. Starting from a genesis message, *TangleSimulator* generates a Poisson message sequence and adds each message to the tangle using IOTA-typical random walk. A well-known analysis of basic properties of IOTA tangles, such as cumulative weight, number of tips for different *Tip Selection Algorithm* (TSA) versions, etc., also uses this method to create different tangles [7], [8]. Similarly, security analysis of IOTA tangles [9], [10] utilizes tangles generated this way to study parasite chain attacks. Authors in [11] study tangle parameter influence on a so-called large weight attack on an IOTA tangle in a real network. They generate the tangle with a Python library and simulate the large weight attack. An analysis of the TSA properties in [6] also uses this method and states that it is rather ineffient, in particular for a bursty message arrival.

The second way is to extract ledger data from the operational IOTA ledger, called IOTA mainnet/devnet tangle. An analysis of the real transaction speed in IOTA extracts real tangle data from the IOTA raw dataset, rebuilds the tangle and finds various abnormal structures in the real IOTA tangle that limit the transaction speed [5]. However, the raw datasets of real IOTA tangles are of limited diversity and availability.

In contrast to these state of the art approaches, which all rely on the basic IOTA protocol, we aim to generate IOTA tangles based on graph construction mechanisms. Random graph generators are widely used in many fields, such as social networks, biological networks and Internet studies [12].

A DAG generation algorithm was proposed in [13]. The input is n vertices and m layers. Two vertices are selected from two adjacent layers. A random variable is generated and, if it is smaller than a predefined threshold p, an edge is added from the vertex of the previous layer to the vertex of the latter layer. Common to random graph generators is the fact that the generated graph does not follow IOTA tangle's degree distribution. To improve IOTA mechanisms, it is an important requirement to closely follow topological features of real IOTA tangles [7].

To find out what would be characteristic properties of tangles, authors in [5] analyzed and compared real and theoretical tangles in terms of in-degree distribution, longest and shortest path, diameter ratio and edge weight. The in-degree distribution of simulated tangles based on [4] follows a Poisson distribution. The length of the longest and shortest paths in tangles with 1 million messages are similar and about 10^5. The edge weights of these simulated tangles are distributed in a range from 1 to 100. Tangles should always have a single genesis message and the out-degree of all messages in all kinds of tangles is limited to 2 [4].

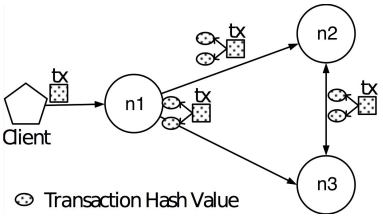


Fig. 1: Full and light node in IOTA network

In summary, the existing IOTA tangle generators can only generate tangles in an inefficient way, while the existing general random graph generation algorithms do not necessarily comply with the identified constraints of IOTA tangles. Hence, we still need an efficient IOTA tangle generator to generate tangles, which meet the in-degree, out-degree, index difference and other prescribed requirements.

III. Algorithm Design

In this section, we first introduce some basic knowledge about IOTA tangles, IOTA's main data structure. The main point is to understand, how IOTA tangle is constructed. Then, we introduce the idea of our proposal and present the details of GraGR, the new graph-based tangle generation method.

A. IOTA Preliminary

1) IOTA System: IOTA is designed and operated by IOTA Foundation (IF)[2]. IOTA network is a distributed system consisting of two types of nodes, full nodes (like n_1, n_2, n_3) and light nodes in Fig. 1. A full node participates in the IOTA network by storing, exchanging and synchronizing message data, eventually written into a local ledger, called tangle and organized as DAG. A light node collects data from users and sends messages to the IOTA network (a full node). For a full specification of an actual IOTA system (node interactions, consensus, etc.), please refer to [5].

2) Message Attachment: As mentioned, IOTA organizes all messages in a DAG. Herein, an edge between two messages indicates the order of attachment and represents approval: in IOTA, fresher messages approve older messages. Hence, an IOTA message processing consists in attaching an arriving message to a DAG leaf, i.e., tangle tip. TSA in IOTA chooses one of the tips, i.e., one yet unapproved message. After the selection of the first tip, the IOTA system selects the second tip in the same way. Once attached to a tip, the newly arrived message becomes a new tip. The selected previous tips are now considered approved and will no longer be selected. While there could be various kinds of TSAs for IOTA, the officially recommended TSA is Markov Chain Monte Carlo (MCMC). For example, in Fig. 2, m_3, m_4, m_5 are tips, and m_6 and m_7 are incoming messages. m_6 may select m_3 and m_4 with MCMC, and m_7 selects a former message to attach without using MCMC.

[1]https://github.com/minh-nghia/TangleSimulator

[2]https://www.iota.org/

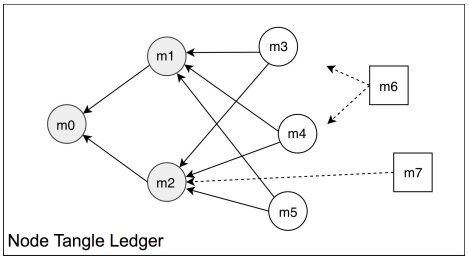


Fig. 2: Message attachment

TABLE I: Variable definition

Variable	*Definition*
G_k	The k-th DAG
n	The size of the message set
λ	Message arrival rate
v_i^t	Message i with out-degree t
V, V_k^t	Message set, Message set of G_k with out-degree t
l_i	The number of messages in the layer i
D_k	In-degree distribution of G_k
d_i	In-degree of the message i
T_k	Out-degree distribution
n_k^t	Message count with t out-degree in graph k
E	Index difference distribution
e_{ij}	Index difference between i and j

Inherently, MCMC is a random walk on a transpose DAG, i.e. in the reverse direction. In a tangle, each message has its own weight and a Cumulative Weight (CW). Own weight is set to 1, and CW is the number of child messages plus its own weight. The child messages are the messages, which point to this message directly or indirectly. In IOTA, the CW of a message indicates the number of messages, which approve this message and, hence, the confidence level of this message. A huge CW value means that the message has a high confidence. The difference between the CWs of two connected messages is the Edge Weight (EW). The EW influences the transition probability of the random walk on the transpose DAG. As per transition probability function [4], the smaller the EW is, the bigger the transition probability is. One important variable for the random walk is α, which reflects the impact of the EW value. If α is 0, it is a uniform random walk. The bigger α is, the greater the impact of EW value on MCMC behaviour has, and the tangle structure tends to narrow.

B. GraGR Algorithm Design

The basic idea of the IOTA tangle construction method in this paper is inspired from graph generation methods. In contrast to generic random graph generation, we create an algorithm that creates a DAG with some characteristic topological parameters of a simulated IOTA tangle. Specifically, we want the generated structure to exhibit the same in-degree distribution as an additionally provided input distribution, and to respect typical constraints of the IOTA tangle's out-degree, as described in Section II.

We define V is the message set, D_0 is the prescribed in-degree distribution (input to our algorithm), and E is the prescribed index difference distribution. The index difference is the difference between the indices of two connected messages. For example, in Fig. 2 the index difference of m_5 and m_1 is 4. The index difference indicates an attachment time interval between two connected messages. For all tangles, the index difference should follow a similar distribution. Otherwise the tangle structure is abnormal, and the attachment order is chaotic. The in-degree list d_i is generated by D_0, and index difference list e_{ij} is generated from E. Table I summarizes all used variable definitions.

Initially, the generated tangle starts with a genesis message, and the new vertices are attached to this genesis message.

The proposed tangle generation algorithm can be divided into two major parts: part I is the *Generation* part, where we generate a DAG following wanted in-degree and index difference distributions, as shown in Algorithm 1. Part II is the *Refinement* part, where we change output Part I using the out-degree distribution and the index difference distribution, as shown in Algorithm 2.

For *Generation* part, Algorithm 1, inputs are message arrival rate λ, number of nodes N, and the prescribed in-degree distribution D_0. The output of Algorithm 1 is DAG G_1. The major steps are as follows:

1. For the first message v_0 in the tangle, assign a random variable l_1 based on the Poisson distribution determined by arrival rate λ, and add l_1 in-degree to this message.
2. Calculate the index difference for each edge, and the index of messages connected to v_0 is the message v_0 index 0 plus an index difference value.
3. For the following vertices, associate each message v_i with an in-degree value d_i and add d_i messages to this messages, whereas d_i follows the given in-degree distribution D_0.
4. The index of the message added in step 3, which directly connects to the current message, is the current message index plus a random index difference e generated from the index distribution E.

While the output of Algorithm 1, G_1, is a DAG with basic properties of an IOTA tangle, it cannot be considered a realistic IOTA tangle, as it includes too many messages with out-degree 0, and, some of its messages have an out-degree higher than 2. Indeed, in Algorithm 1, we only enforce the prescribed in-degree and add the directly connected children to each message, while out-degree constraints are not respected yet.

The following is the *Refinement* part, Algorithm 2, starts from G_1 and the prescribed out-degree distribution T_0.

1. Calculate the out-degree distribution of G_1 and compare the percentage of each out-degree value of the out-degree distribution T_1 to the prescribed out-degree distribution T_0. Select the messages with unqualified out-degree values, for example, 0 or higher than 2.
2. For all messages with out-degree 0 - except the genesis message - generate two random index difference variables e_{ij}, e_{ih} based on the index difference distribution E. Link v_i to $v_{i-e_{ij}}$, $v_{i-e_{ih}}$.

3. In Step 3, if the index of the added out-degree message is smaller than 0, then link v_i to genesis message v_0.
4. For messages, whose out-degree is bigger than 2, delete the edges randomly, until their out-degree is exactly 2.
5. For messages with out-degree 1, if the prescribed out-degree distribution needs more messages with out-degree 2, then randomly select the messages v_i with out-degree 1, link v_i to a former message $v_{i-e_{ij}}$, the index difference e_{ij}, which is also selected from the index difference distribution E_0.

Algorithm 2 returns DAG G_2, which is a synthetic tangle with prescribed properties.

Algorithm 1 Graph Generation

Input: n, V, D_0, E, λ, T_0
Output: G_1
for v_i in V **do**
  if i == 0 **then**
    j $\sim$ Poisson(λ)
    V'=$\{v_1,v_2,...,v_j\}$
    add $v_j \in V'$ to v_0
  else
    h $\sim$ D
    generate V'=$\{v_{j_1},v_{j_2},...,v_{j_h}\}$
    each j_h = i + e, e $\sim$ E
    add V' to v_i
  end if
end for
return G_1

IV. Evaluation

In this section, we evaluate general ProSG methods and our GraGR proposal in the following aspects: out-degree, in-degree, longest path, shortest path from the genesis message to the latest tip, diameter ratio, index difference and costs, i.e., time and memory consumption. Since the realism of ProSG is not questioned, we use it as a reference for topological parameters. In contrast, we want to have a better performance.

A. Experiment Setup

We run all experiments on a computer with Intel Core i5-8265U @ 1.6Ghz CPU and 16GB RAM. All algorithms are implemented in Python 3.8. For the experimental setup, we set $\alpha = 0.01$, which is the default value in the real IOTA network. Then, we vary both λ values, and the number of vertices N in Table II. We generate a number of tangles for each set of parameters, until the statistical error of the reported results is lower than 5%. In practice, the number of generated tangles was around 10 for each parameter set.

B. Results

1) Path Length: The determined shortest and longest path lengths from the genesis message to the latest tip of the tangles generated by ProSG and GraGR are shown in Fig. 3.

Algorithm 2 Graph Refinement

Input: G_1, n, V, T_0
Output: G_2
for v_i in V_1 **do**
  if Out-degree (v_i) == 0 and $i \neq 0$ **then**
    e_1,e_2 $\sim$ E
    v_{j_1} = i - e_1
    v_{j_2} = i - e_2
    if v_{j_1} <0 **then**
      v_{j_1} = 0
    end if
    if v_{j_2} <0 **then**
      v_{j_2} = 0
    end if
    Add v_i to v_{j_1}, v_{j_2}
  else if Out-degree (v_i) >2 **then**
    Successor (v_i) = $\{v_{j_1},v_{j_2},...,v_{j_h}\}$
    random delete edges until Length(Successor(v_i)) ==2
  else if $n_0^2 > n_1^2$ **then**
    $\Delta n = n_0^2 - n_1^2$
    random select Δn vertices V' from V_1^1
    get $V' = \{v_{i_1},...,v_{i_{n'}}\}$
    each $e_{n'} \sim E$, $j_{n'} = i_{n'} + e_{n'}$
    add $v_{i_{n'}} \in V'$ to $v_{j_{n'}}$
  end if
end for
return G_2

TABLE II: Experiment parameter setup

Parameters	*Value*
α	0.01
λ	5, 10, 15, 20
N	$10^4 \sim 10^5$ with a step-size $= 10^4$

As can be seen in Fig. 3, with the number of messages going up, the lengths of the longest and shortest paths increase. For the same number of messages, for smaller λ values the paths are longer. For $\lambda = 1$ (smallest value), the tangle degenerates to a chain. Generally, for bigger λ values, the tangle becomes wider. Note that for the same value of λ, the longest and shortest paths of the tangles, generated by ProSG and GraGR respectively, are similar in length. We conclude that GraGR maintains the path length properties of real IOTA tangles.

2) Diameter Ratio: In addition, we calculate the diameter ratio of tangles generated by these two methods and the absolute value of the difference of diameter ratios. We define diameter ratio as the longest path length divided by the shortest path length. Semantically, this term is indicative of the shape of the tangle. A bigger diameter ratio indicates the tangle is wider, while a smaller diameter ratio means that the tangle becomes like a narrow band. The results are shown in Fig. 4. For diameter ratio comparison, we use $\lambda = 10$ as an example, as the results are similar for different λ. In Fig. 4a, the difference between the diameter ratio of the tangle generated

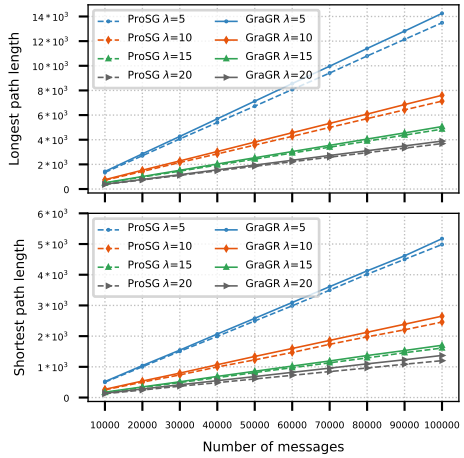
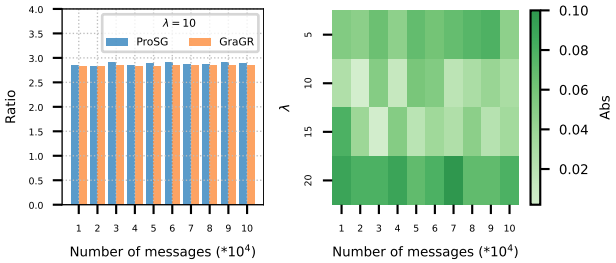


Fig. 3: Comparisons of path lengths of generated tangles



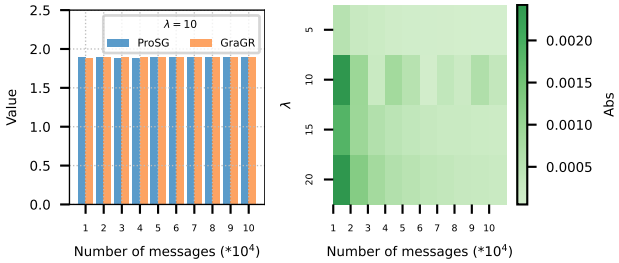
(a) (b)

Fig. 4: The comparison of the tangle diameter ratio

by the two methods is very small. Fig. 4b shows the diameter ratio difference of the experiments. Most difference values are around 0.06. This comparison indicates that the tangles generated by GraGR and ProSG have quite similar diameter ratios. Hence, we conclude that GraGR generates tangles of realistic shapes.

3) Out- and In-Degree Distributions: Just like ProSG methods, which use IOTA TSA, by the strict construction of GraGR in Algorithm 2, the out-degree of each message in the constructed tangle exactly follows the prescribed out-degree distribution, but without employing TSA.

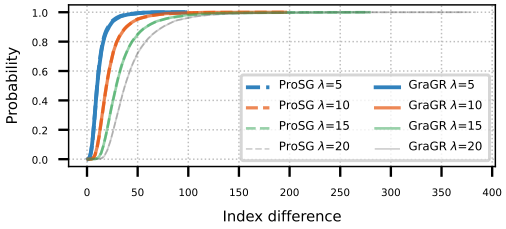
We now evaluate the in-degree distribution of the generated tangles. Specifically, we measure mean and absolute difference values of the in-degree mean of tangles generated by the two methods, as shown in Fig. 5. Using $\lambda = 10$ as an example, Fig. 5a shows that the average in-degree of tangles generated by two methods is essentially the same. We show the absolute value of difference of in-degree mean in Fig. 5b. For higher numbers of vertices (e.g. more than 10000), the difference value is slightly larger. However, the maximum difference is under 0.002, which is still small. As the number of vertices grows, the difference of the in-degree mean becomes smaller, and, when the number of vertices is 100000, the difference is still below 0.0005. Note that the in-degree mean difference remains stable for changing parameter settings. These findings confirm that the tangles generated by ProSG and GraGR have



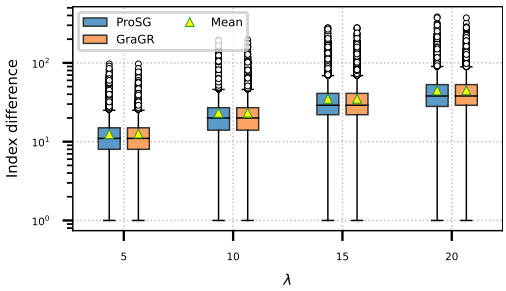
(a) (b)

Fig. 5: The comparison of the tangle in-degree mean

similar properties in terms of in-degree.



(a) CDF



(b) Boxplot

Fig. 6: The comparison of the index difference

4) Index Difference: We calculate the index difference in the generated tangles and compare their distributions.

Results are presented as Cumulative Distribution Function (CDF) in Fig. 6a. Note the good match of the index difference distribution of the tangles generated by ProSG and GraGR for the same message arrival rate respectively. Depending on λ, the proportion of high index difference values (more than 50) rapidly decreases: while for $\lambda = 5$, there is a very significant proportion of index differences underneath 25, for $\lambda = 10$, the same proportion includes values underneath 50 and for $\lambda = 15$ and $\lambda = 20$ - values under 100.

Fig. 6b shows this phenomenon more clearly. The index difference values of the tangles generated by ProSG and GraGR are almost the same. The index difference values are distributed in a small range, for example, most index difference values of the tangles with $\lambda = 5$ are mostly distributed in the range (1,20). As λ increases, the median and mean value of the index difference distribution also increase. Overall, however, the index difference in the tangles produced by ProSG and GraGR are similar.

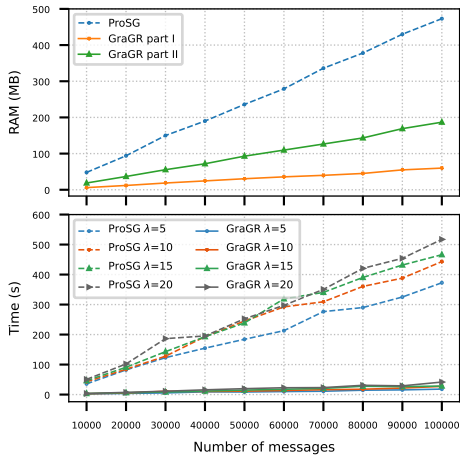


Fig. 7: The comparison of the consumed time and memory

5) Runtime and Memory Cost: We evaluate both ProSG and GraGR approaches in terms of runtime and memory consumption required when generating tangles of the respectively same size. We run experiments for different tangle sizes N and using different message arrival rates λ. We record the elapsed time and the required memory. The results are shown in Fig. 7.

First of all, we observe that λ has almost no effect on the memory consumption. Hence, we chose $\lambda = 10$ as an example. The upper plot in Fig. 7 shows the consumed memory. As expected, bigger graphs (more vertices) require more memory; the relationship is essentially linear. To better understand GraGR, we present the consumed memory separately for both of its phases. GraGR Refinement part consumes approx. 3 times more memory than its Generation part, because Refinement needs to calculate the out-degree distribution of the tangle. Comparing GraGR to ProSG, we observe that GraGR consumes only half of the memory required by ProSG for the same size of the generated IOTA tangle.

The second plot in Fig. 7 describes the runtime duration of both generators for a tangle of the same size. Again, as expected, with the increasing number of vertices, both ProSG and GraGR require more time; the relationship is, again, essentially linear. However, the runtime of ProSG grows significantly faster than that of our algorithm. Generating a tangle with the same number of vertices, GraGR is up to 10 times faster. Also note that the message arrival rate has a relatively low impact on GraGR execution time.

Overall, GraGR is way more efficient. From the observed trend, and expressing it another way around, on a platform with 1024MB of memory, to produce a tangle with 1 million vertices, the state of the art ProSG requires from 3000 to 5000 seconds, depending on the message arrival rate, while GraGR finishes the task in only 300 to 500 seconds. Given the second-only duration of tangle production for smaller size tangles, with GraGR synthetic tangles can be produced on the fly, without the need to store the tangles for latter analysis. This significantly speeds up the experimentation.

V. Conclusion

In this paper, we propose a novel IOTA tangle generation algorithm. Instead of following IOTA protocol like all existing generators (aka ProSG), our proposal, GraGR, manipulates the topology of a generated random DAG, until its properties fulfill requirements on IOTA tangles. While GraGR delivers results topologically equivalent to ProSG, GraGR consumes only about half of the memory and is up to 10 times faster.

The main difference between ProSG and GraGR is the reliance of ProSG on repetitive reverse random walks. Our evaluations suggest that such random walks are indeed the main contributor to the performance gap of ProSG.

In general, the new tangle generation algorithm provides the research community with an easier way to yield experimental tangles for DAG DLT research. For the first time, GraGR allows on-the-fly generation of realistic IOTA tangles with hundreds of thousands of messages.

We recently found that *double Pareto Lognormal* (dPLN) distribution may be a better fit [14] for tangle degree distribution. While in this paper, we still rely on the accepted state of the and require Poisson distribution, a generator using dPLN distribution will be addressed in our future work.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International journal of web and grid services*, vol. 14, no. 4, pp. 352–375, 2018.

[3] L. Hughes, Y. K. Dwivedi, S. K. Misra, N. P. Rana, V. Raghavan, and V. Akella, "Blockchain research, practice and policy: Applications, benefits, limitations, emerging research themes and research agenda," *International Journal of Information Management*, vol. 49, pp. 114–129, Dec. 2019.

[4] S. Popov, "The tangle," *IOTA Found. Whitepaper*, p. 131, 2016.

[5] F. Guo, X. Xiao, A. Hecker, and S. Dustdar, "Characterizing IOTA Tangle with Empirical Data," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[6] X. Xiao, F. Guo, A. Hecker, and S. Dustdar, "Fast Tip Selection for Burst Message Arrivals on A DAG-based Blockchain Processing Node at Edge," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, Dec. 2022, pp. 1373–1378.

[7] B. Kusmierz, P. Staupe, and A. Gal, "Extracting tangle properties in continuous time via large-scale simulations," Technical Report. Accessed: 2018-08-23, Tech. Rep., 2018. [Online]. Available: https://https://www.iota.org/foundation/research-papers

[8] B. Kusmierz, W. Sanders, A. Penzkofer, A. Capossele, and A. Gal, "Properties of the Tangle for Uniform Random and Random Walk Tip Selection," in *2019 IEEE International Conference on Blockchain (Blockchain)*, Jul. 2019, pp. 228–236.

[9] A. Penzkofer, B. Kusmierz, A. T. Capossele, W. Sanders, and O. Saa, "Parasite chain detection in the iota protocol," in *Tokenomics*, 2020.

[10] S. Ghaffaripour and A. Miri, "Parasite chain attack detection in the iota network," *2022 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 985–990, 2022.

[11] L. J. W. Vries, "IOTA Vulnerability: Large Weight Attack Performed in a Network," B.S. thesis, University of Twente, 2019.

[12] M. Drobyshevskiy and D. Turdakov, "Random graph modeling: A survey of the concepts," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–36, 2019.

[13] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *Journal of Scheduling*, vol. 5, no. 5, pp. 379–394, 2002.

[14] F. Guo, X. Xiao, A. Hecker, and S. Dustdar, "A Theoretical Model Characterizing Tangle Evolution in IOTA Blockchain Network," *IEEE Internet of Things Journal*, vol. 10, no. 2, pp. 1259–1273, Jan. 2023.