Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Graph Generation with Preserved Properties

Valentin Marianov

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Kurt Rothermel
Supervisor:	Michael Schramm, M.Sc.
Commenced:	November 20, 2020
Completed:	May 20, 2021

Abstract

The increasing size of real-world networks and the lack of publicly available datasets due to security and user privacy concerns has increased the demand for graph generators. Developing graph generators could facilitate researchers in analysing network's properties and underlying structure. More accurately, a request towards designing synthetic generator applicable to arbitrary graphs, which also scales up to massive datasets, is made. Such model could eventually be further extended to produce graphs of multiple size aiming to predict how those will conceivably evolve in the future. Having acquired synthetic networks we could then measure multiple metrics in order to understand a graph's structure and properties. However, current graph generators are only capable of matching a fraction of real-world graph properties, are not suitable for different types of networks, do not scale up for large datasets or a mixture of the preceding.

The main goal of this thesis is to take a deeper look in how well state-of-the-art graph generators preserve real-world graphs and their properties. Answers to the following questions are to be investigated: How well can a generation model capture a certain property? Is the model reliable in generating real-world networks from different domains? Can any similarities be found between the properties of synthetically generated and real-world graphs? Can a generator potentially produce graphs with adaptable properties, e.g. multiple times larger graph? Detailed analyses of existing graph generators and approach to try to combine or extend them to be able to generate graphs with similar to real-world networks properties are the head focus of this work.

After investigating several generation models we focused our attention on *Darwini*. Because to the best of our knowledge, it is only the second approach making an effort to concomitantly match the explicitly provided degree and clustering properties. In addition, no work aiming at questioning the model's ability has been found motivating us to thoroughly analyse the algorithm. Besides assessments we contributed to this topic by extending the algorithm with two methods aiming to predict how the structure of a certain network would look like in the future. Darwini along with our proposed extensions have been implemented in the *Graph Analysis Measurement Environment (GAME)* framework to analyse and compare the properties of arbitrary graphs. As we have shown in Chapter 6, Darwini lacks the ability of reproducing graphs with notable degree difference of neighbour elements in the distribution series and missing vital links connecting multiple components with one another. However, distributing vertices across several groups could address the issue and particularly improve the overall results.

Contents

List of Figures

List of Tables

Acronyms

AS Autonomous Systems. 50

BTER Block-Two-Level-Erdós-Renyi. 18

CGA Community Guided Attachment. 19

CL Chung Lu. 18

CSR Compressed Sparse Row. 32

DBC Database Connectivity. 31

ER Erdós-Renyi Model. 27

FF Forest Fire. 19

FRD Fast Reciprocal Degree. 18

GAME Graph Analysis Measurement Environment. 3

LPA Linear Preferential Attachment. 7

MVC Model-View-Controller. 31

R-MAT Recursive Matrix. 18

RID Random Increase of Degree. 7

SKG Stochastic Kronecker Graphs. 19

1 Introduction

By studying the evolution of real-world networks, we can identify various properties of a graph such as important nodes known as hubs, numerous communities and many others. Numerous studies show that real-world networks like online social graphs exhibit continues growth and shrinking average diameter. For example, in 2011 the Facebook graph [UKBM11] consisted of about 721 million active users and well over 68 billion friendships (edges) between the previous, while as of the end of 2014 the social service network had grown up to having 1.39B active users and composed of more than 400B links between the last [CEK+15]. According to the Facebook third quarter reports of 2017, the amount of active users has increased with approximately 16.66%, since the preceding results, reaching 2.07B by the end of September 2017. So, one could ask himself the following question: Do online social networks get more dense as the graph grows overtime? Well, as of 2011 every Facebook user was connected to around 95 other people. Three years later, measurements show that on average every user had established roughly 287 friendships.The conclusion that social graphs do *densify* during their evolvement is pretty obvious, as stated in [LKF05].

To be able to come to such conclusions we would need huge public datasets. Even when such large collection of data exist, there are a couple of factors that influence their usage. First, researchers must respect user privacy and security, thus most of the large datasets are not publicly available. On other hand, if we assume that such enormous data is shared with the public, the effort, the time and the resources that need to be invested makes the work of researchers even more challenging and unpleasant. Countless hours and many efforts have been put in the development and research of graph generators for power law distribution graphs. However, the majority of them are incapable of producing synthetic graphs with similar properties [ELW+16]. Vital role for the lack of accuracy to a great extend holding for nearly all generative models are some conclusions made specifically for online networks. For example, it was believed that the average degree of a given network would remain persistent over an observed period of time, while in reality a monotone function could display the slowly increasing diameter of the network during its evolution. The most well known inference is probably that the degree distribution of real-world graphs always matches a power law function, which was already rejected by numerous studies [ELW+16], [KPPS14], [UKBM11], [KNT10]. Those inaccurate conclusions made by previous studies are derived from the research and evolution of static graphs or in some case of small set of snapshots of networks.

Before [LKF05], no one had ever observed the structure and growth evolution of real-world networks, which rejected the statements about the constant average diameter and the increasing average distance between pairs of vertices. The study of [LKF05] also discovered that Stanley Milgram's experiment *six degree of separation* holds for a variety of networks. The authors of [LKF05] have also proven that the amount of edges is growing super-linearly in the number of vertices. In order to address the above mentioned issues, synthetic graph generators were designed to reproduce huge networks based on published graph metrics retrieved from source graphs. Given a publicly available degree distribution, synthetic graph generators can reproduce graphs with similar distribution while simultaneously trying to capture multiple supplementary characteristics. Using

that type of generators we can also try to predict how the structure of the network will look like over some time as it evolves and further measure a variety of metrics. Important metrics that describe the structure of the graphs include degree distribution, number of connected components, clustering coefficient distribution, diameter, degree correlation, eigenvalues and others. Additionally, the model must ensure efficient memory usage and close to linear time complexity thus producing networks at scale, since system bottlenecks normally occur only by working with a broad range of data. The synthetic generation approach does not only omit processing the connections between millions or billions of pairs of vertices, but moreover ensures that user sensitive data is by no means publicly shared.

State-of-the-art generation models match several properties of real-world graphs but at the same time are unable to replicate at least one of the of three crucial aspects, as stated in [ELW+16]. Famous social networks such as Facebook and Instagram limit the amount of possible friends up to 5000 for the Facebook social network or the amount of people one can follow up to 7500 for the latter one. YouTube is another famous example of a big real-world dataset, where each user is limited to subscribing to a maximum of 2000 channels. In contrast one of the most popular generation models Kronecker [LKF05] produces power law based synthetic graphs, thereby it does not have the ability to meet the actual degree distribution of graphs, having such limitations like the previously mentioned once. Another issue worth mentioning is that some existing approaches could not be used from organizations due to several factors. One being that some generators are unable to scale therefore making them impracticable for industry use. Further important argument is that some models require manual tuning of parameters, which in case of fault configuration could lead to inaccurate graph structure and in general inaccurate metric results [ELW+16]. The performance of graph processing systems such as $GraphX$, $Giraph$ and $Pregel$ [KLEC16], [XGFS13], [LT16], [MAB+10], [IHN+16] can be heavily influenced by incorrect local clustering properties. The partitioning of a graph across individual machines inside the cluster is then processed not as effectively as one would like it to be. The generation of such large datasets is also needed in order to test the capabilities of graphs algorithms and to measure the performance of graph processing frameworks and for producing detailed benchmarks [KLEC16], where the strong and weak sides of each processing system can be displayed.

The idea of generating bigger graphs than the source one is to find a model that can predict how real-world graphs would naturally grow over time, is also observed by this thesis. Such growth predicting model can help identify problems, which may occur in the future. For example, by generating a multiple of a router network one can potentially spot future data flow bottlenecks where a set of routers send information to a common router. The last having the role of a bridge to connect two end points from both side, e.g. users. While this type of structure would not cause any issues if the data traffic is low, in scenario where we assume that many users from the left side of the bridge router want to send message to users located on the opposite side of the bridge and the other way around, then one of the following scenarios can occur: Each or most of the users will probably receive and/or send its messages with delay due to huge amount of computation that the bridge router has to deal with in order to find the end-target for each received message. Or in the worst case scenario, the router will shut down due to huge data traffic and thus no longer will users across sides be able to communicate with each other.

In this thesis we focus our attention to different state-of-the-art graph generators. The goal is to understand how well they are capable of reproducing single properties of real large data. In the next chapter, we will briefly introduce what a graph is and other related to this work graph theory based

terms. Then we summarize previous works in the domain of (synthetic) graph generators. Next, in the third chapter we introduce what a metric is and how is it related to graph generation models followed by a detailed presentation of every model observed in Section 2.2. Before providing details about our custom implementation of the **Darwini** algorithm together with our extensions part of the *synthetic graph generator* to produce multiple graphs, we will introduce the *GAME* framework. *GAME* standing short for *Graph Analytic Measurement Environment*, as the name suggests is a graph processing framework. Written in the *Java programming language* it is used to generate, import, export and analyse existing graphs by measuring diverse metrics and was developed at the University of Stuttgart, Germany. Additional information about the framework can be found in Chapter 4, where we point out the set of libraries used, what graph types the framework can generate and others features. The fifth chapter will be composed of a comprehensive description of our custom implementation of the Darwini algorithm [ELW+16]. Within this chapter we include explanation about the data structures used, our approach to speed-up the execution by parallelizing all possible parts but at the same time ensuring accurate results. And lastly, we take a deeper look at the growth prediction models we developed to produce networks of multiple size implemented as extension to the already discussed synthetic graph generator. Next, in Chapter 6, we provide a thorough evaluation of the results provided after generating divergent types of, but not only limited to those, online social graphs. The Darwini approach [ELW+16] will be tested in detail also on online social networks but not only restricted to that category of real-world graphs. The aim of testing being understanding the full capabilities of this quite interesting approach, which on paper should be able to capture local clustering coefficient at fine granularity, unlike other existing state-of-the-art models. Questions that need to be answered include: How well can a generation model reproduce the structure of a graph and thus preserve its properties. Is a given generator capable of producing accurate results only for real-world networks? Is it adaptable of generating graphs without fixed property? And finally, in Chapter 7, conclusion of our thesis and explicit hints about future work in the direction of improving *Darwini* and our proposed extension models are provided.

2 Background and Related Work

The demand for reliable and scalable graph generators is demanded by the speed of which real-world networks continuously expand their domains. The computation power of regular machines is also unable to keep pace with the evolution of networks like Facebook. Therefore, many researchers have made every endeavour to design a generation algorithm accurately and efficiently matching the underlying structures of original graphs and also preserving one or more key characteristics. However, to the time of writing no graph generator is apt to contemporaneously match real graph properties in a quick fashion. Before going through a variety of generation models and discussing their potential usage based on previous studies, we would like to introduce basic graph theory terms used in the rest of this thesis. People without any experience in graph theory are encouraged to read the incoming section, while experienced readers can feel free to skip the forthcoming section.

2.1 Background

A graph is a tuple $G = (V, E)$ composed by two distinct sets: a set V containing n nodes, and a set E containing m edges. An edge $e_{i,j}$ or (v_i, v_j) is a connection between two vertices v_i, v_j with $v_i, v_j \in V$. Edges can be either undirected, like in the networks of Internet routers and citation networks or directed found in road infrastructure networks, online social networks like Instagram and Epinions. In directed graphs, reciprocal edge means that a directed edge exists in both directions between nodes v_i and v_j. Additionally, edges could also have weights. The logic behind the weight can differ dependent on the domain of use. In social networks, bigger weight values typically describe stronger relationship between two people. However, in road infrastructure graphs bigger weights would describe longer distance from point A to point B. In case of a traffic jam, a hybrid metric could also be represented by an edge weight indicating the time needed to travel though a single street or boulevard. Graphs can also be either dense or sparse. A dense graph contains n nodes and nearly or exactly n^2 edges, i.e. the number of edges is close to the maximum number of permitted links. Sparse graph on the other side usually contains less than n edges, i.e. nodes have few links and possible several disconnected components and/or isolated vertices exist. An isolated vertex v_i is a vertex not incident to a single edge or with other words having 0-degree.

Most real-world networks are distinguished from random graphs by having many communities. A community is subgraph of G, where most of the nodes are connected with one another. Therefore, such communities also tend to be very dense. According to [KNT10], social networks can be divided into three groups: singletons, also known as isolated vertices, middle region consisting of small isolated communities and the giant component within which each vertex can find a path to the remainder of vertices in the component. The giant component can further be divided into two layers: the core, consisted of all active and strongly connected users and an outer-layer of numerous stars. After structural investigations of a variety of social networks, the authors of [KNT10] also conclude that the majority of the isolated groups are *stars*. A star is composed of a central node

connected to all remaining vertices in a community, whereas the remaining nodes have either none or very few additional edges. Another typical feature of many real-world networks, particularly for online social platforms, is the occurrence of small amount of high degree vertices (e.g. famous people followed by the majority of members in the network), while huge amount of the remaining vertices link to only several nodes. Possible reasons for this in social networks include: lack of interest after joining the network, only a small portion of users friends can be found on the social platform or users only participate in some groups which may be connected with personal hobbies and interests or related to the user's education and/or work domain.

The high amount of low degree vertices and occasional occurrence of high degree nodes has been observed in broad diversity of domains leading to heavily-tailed degree distributions [DKPS13], [KPPS14]. The presence of such degree distribution curves is considered as a critical feature for distinguishing real-world networks from arbitrary sparse networks. This, however is not the only feature of real-world networks. Shrinking diameter, high average local clustering coefficients and hierarchical community structure, i.e communities within communities, along with many others commonly emphasize the existence of a real-world network. To measure each of the previously mentioned graph characteristics, we would need to use a variety of metrics, which are presented in the upcoming chapter. But before that, in the section below, we will summarize previous contributions in the domain of graph generation.

2.2 Related Work

History of generation models dates back to the late 50's and to the early 60's of the last century with the introduction of *Erdös-Renyi* [ER+60] and *Gilbert* [Gil59] models. Their usage in the real-world is limited because they return a single graph from a set of graphs with pre-defined order and size with equal probability. Due to random link creation, those models are also ill-suitable for preserving any properties of real-world graphs. However they serve as a good null model for other generators like [KPPS14].

Chung Lu [ACL01] is another null-based model. It also generates random graphs but unlike *Erdös-Renyi* and *Gilbert* methods can capture heavily-tailed degree distributions common for the majority of realistic networks. Outside of providing more precise degree distribution and inexpensive cost per edge creation, Chung Lu (CL) does not build upon *Erdös-Renyi* and *Gilbert*. Used as an extension from Fast Reciprocal Degree (FRD) [DKPS13] or as part of more sophisticated models like Block-Two-Level-Erdós-Renyi (BTER) [KPPS14].

The World Wide Web, peer-to-peer networks and the Internet topology follow surprising law degree distributions and exhibit 'bow-tie' and 'jellyfish' structures with also rather small diameter, as stated in [CZF04]. Therefore, a recursive graph model Recursive Matrix (R-MAT) was developed to quickly generate realistic graphs, preserving power-law behaviours and deviations from them. R-MAT is suitable for modelling bipartite and weighted graphs unlike most state-of-the-art generators. Vertices are distributed in two groups and can only connect with nodes from the opposite group. Described as a procedural method, R-MAT can generate any type of graph while satisfying criteria of real networks like community structure (in addition to the degree distribution). Barabási-Albert model and its modifications, other approaches including geometric layout of nodes in their model and the BRITE model composed of components taken from the previous mentioned models, fail to meet at least one characteristic captured by R-MAT. However, generating such graphs can only be

possible by correctly setting all four probability fields. Probability fields a and d represent separate groups of nodes, i.e. communities or even nested communities. Ratios of 75:25 for $a : b$ and $a : c$ are also approximated for the majority of real-world networks [CZF04]. Although generating weighted, directed and even bipartite graphs, R-MAT generally does not suit reproduction of realistic networks because it only targets in maintaining the degree distribution of the original graph by implicitly forming nested communities through assignment of high field probabilities for a and/or d.

Since the seminal work of Barabási, Albert and Faloutsos, the degree distribution of graphs is considered as a key feature in distinguishing between random and realistic networks [DKPS13]. Heavily-tailed degree distributions, as we also mentioned, can be found in wide variety of domains. The authors of the *Fast Reciprocal Degree algorithm (FRD)* further conclude that most web, communication and interaction networks are directed and exhibit high reciprocity. Because many generation algorithms do not support directed graph generation, a common approach is to make the last undirected. Studies have additionally concluded that the fast spread of news or viruses occurs in graphs exhibiting high reciprocity. Reciprocity is also usually significantly higher in social networks than in information graphs and even have a vital role in interactions during gameplay of massive online games. All of the above summarised observations lead to the development of the Fast Reciprocal Degree model. Serving as an extension to the CL method, FRD almost perfectly matches the in-, out- and reciprocal degree of original graphs. It can be applied to both directed and undirected graphs while preserving the high reciprocity in real-world graphs unlike other state-of-the-art generation models like Stochastic Kronecker Graphs (SKG), Forest-Fire (FF) [DKPS13]. FRD can also produce directed graphs, while matching the in- and outdegree quite good. However, a directed comparison of FRD against other generation models is not appropriate since the previous is not a realistic graph generator, i.e. it does not take into consideration other metrics apart from degree distribution. It also requires manual parameter tuning of β for 1-degree nodes as being an extension of the Chung Lu model. Therefore, even by matching the degree distribution of real-world graphs at fine granularity and producing networks with millions of nodes and edges in couple of minutes, FRD can only be used as a base model for the development of further generation models.

Rejecting the long believed assumptions that the average degree remains constant and a slow increase in the diameter can be observed in real-world graphs as they grow can be seen in [LKF05]. Achieved by studying the evolution of various datasets in huge time span, however rose the following two questions: What underlying (natural) process causes a graph to sweepingly densify with having a fixed exponent α? The value of α can be used to estimate the size of larger graph at time t' by computing the proportionality of the size and order at time t. This leads to the following question: Why do we see ceaseless shrink in the effective diameter as network's domains further extend? To answer this questions, the authors in [LKF05] came up with the design of the *Community Guided Attachment (CGA)* and *Forest-Fire (FF)* models. Forest Fire (FF) in general preserves more properties than Community Guided Attachment (CGA), while the later also requires manual tuning of parameter producing divergent in- and outdegree distributions dependent on the value set. Therefore, we can straightforward conclude that CGA approach is not sufficient enough and continue with the deliberation of the more complex graph generator. Forest Fire to a great extent is based on CGA by producing a version of the Densification Power Law [LKF05]. On top of that heavily-tailed indegree distributions can be reproduced based on the well known *rich get richer* process. The composition of communities with the help of a component sharing flavour of the $copying$ model while addressing the issue of CGA with the constant value decrease of the diameter

are part of FF. There are however, a couple of aspects that need to be pointed out. First, FF links a fresh joining node to the graph using a forward and backward burning parameters p_f and p_b. The last two of course stick up with the necessary tuning. There is also no concrete explanation provided in [LKF05], on why the generation of graph with FF would lead to a similar synthetic graph with shrinking effective diameter. Densification can be one of the reasons since new nodes tend to created many links in the ambassador's neighbourhood and fewer beyond its scope. However, this would not be enough to make such general conclusion. Apart from that, Forest Fire also creates a negligible amount of reciprocal edges, which represent quite large percentage in some real-world networks.

Next, we would like to take a look at generation model taking into consideration not only the degree but also the local clustering coefficients of vertices. *Block-Two-Level Erdös-Renyi (BTER)* model was developed to scale up to large graphs like online social networks, collaborations, telecommunications and computer-to-computer networks while trying to maintain several properties. By taking as input the degree and clustering coefficient distributions all parameters are being computed directly using the previous two (except the blow-up factor β). Independent edge creation in arbitrary order and the absence of iterative optimization to fit BTER belong to the list with positive features about the algorithm. On top of that its also only the second technique mentioned here making the effort of forming communities based on provided distributions, hence leading to more accurate results than SKG and CL models. The way communities are formed, however leads to inaccurate results in the clustering coefficient distribution. Since, each affinity block (i.e. community) holds vertices sharing common target degree, BTER implicitly presumes that this set of nodes also should match the same clustering coefficient, which should not be necessary true for the majority of them. Thus by trying to directly match the desired degree of each vertex in phase 2, clustering coefficients could not be fixed in the later phase. As it has been concluded in both [KPPS14] and [ELW+16], BTER uses a blow-up factor in order to generate enough degree-1 vertices. Beside that, BTER comes short in reproducing high degree vertices, unlike Darwini addressing the raised issues by forming communities joined by vertices with identical expected number of closed wedges. By establishing groups with approximated amount of triangles for each vertex, Darwini indirectly tries to match both the degree and clustering coefficient for each node. After second phase, the values of the last are corrected while connecting smaller communities with one another and thus forming the final version of the output graph. Darwini produces notably better results than other state-of-the-art models such as BTER, FF, Random Walk and DK2. Remarkable is the deviation observed in clustering coefficient distribution results between BTER and Darwini, whereas the latter comes a lot more closer to the distribution curve of the original graph. Drawback for both models is that they currently observe only simple unweighted and undirected graphs. They also assign each vertex to a single group and ignore complex hierarchical structures, which can be found in models like CGA and R-MAT. Based on the above summary we have chosen to deeply test the capabilities of Darwini because there are yet no detailed researches in this direction. Even though, it brings some limitations with itself, Darwini could possibly be extended to eliminate those issues.

This thesis was inspired by [KPPS14], the synthetic graph generator Darwini [ELW+16] and its capabilities of capturing several properties of real world graphs while trying to form an identical to the original graph structure. As we show in our evaluation, Darwini is not capable of reproducing those properties mainly affected by the way it picks only one group for each vertex, contradicting with structures of online social networks where users are usually members of multiple communities. As a consequence not only does the model struggle to generate graphs with hierarchical structure but also could lead to some isolated communities and singletons due unmatched score for high degree

vertices. This is a crucial aspect resulting in diverging from the original graph structure impacting to some extent the remaining metric results. Darwini manages to reproduce degree distribution only when no significant difference in the value of two subsequent elements in the series is present. The way it distributes vertices along with the design decisions taken in maintaining the degree correlation by firstly matching the expected number of triangles each node should participate at, restrict the amount of options we have to connect high degree vertices. Even though that after certain amount of iterations in the third stage we can search across the entire graph spectrum for possible candidates, each of the remaining vertices with non-matching target degrees tend to already posses links between each other or at least the majority of them. As a evidence to our argumentation, we have removed the probability formula in the second half of phase 3, which tries to maintain the degree correlation. However, little to none improvements have been observed. The output graphs still exhibited absence of high degree nodes found in original datasets. During the implementation of Darwini, we introduced two extensions to the algorithm trying to predict how different types of networks would grow over time. In *Chapter 6* we depict that our approaches can in general follow the abrupt changes in the original curve. Both models however, do not adjust by any way the clustering coefficient of each vertex, whose degree is increased. *Random Increase of Degree (RID)* is more suitable for graphs, where we do not expect major changes in the size or density of communities while *Linear Preferential Attachment (LPA)* is more appropriate for online social networks, where people with raising popularity accumulate numerous links from fans and followers.

3 Metrics and Generation Models

During the last few decades a lot of time, resources and energy have been oriented to the development of graph generators capable of reproducing real-world graphs. The design of such universal model is not a trivial job due to a couple of reasons. First and foremost, it should be able to capture important features of real-world networks such as heavily-skewed degree distribution, short distance between an arbitrary pair of nodes and many others. This approach should also be applicable to directed and undirected, weighted and unweighted graphs, eventually with multiple links between vertices and self-edges. Synthetic graph generators have to rely on the least as possible amount of data provided by the original graph because most real-world datasets are not publicly available due to privacy and security concerns. Also some generation models require manual parameter tuning making them extremely sophisticated for use. Wrong estimation of a parameter could lead to the generation of a graph having a structure significantly diverging from the one of the original graph. And lastly, even if the algorithm generates accurate graphs while preserving one or more key features of real-world networks, it should be designed and implemented in such a fashion that makes it scale up to large graphs with trillions of edges and thus applied in different industry domains. But before we outline several generation models we would like to present a set of metrics and their relation to graphs and generation models.

3.1 Metrics

What is a metric? How are metrics related to graph generation? Metrics are basically computed characteristics of graphs such as degree distribution. They are vital in understanding the structure of graphs and are sometimes even used as input from generation models such as BTER [KPPS14] and Darwini [ELW+16]. Measuring metrics can also give us a hint about the type of a graph - random sparse graph versus real-world dense network. For example, real-world graphs exhibit low effective diameter as per [LKF05]. They can also be used to detect abnormality and thus manage the network by taking the required measures. In many real-world network settings, normal behaviour is described by small dense subgraphs and some additional properties regarding network growth. So, theoretically by detecting activities producing abnormal structures, one can label them as a potential risk of e.g. fraud, viruses, fishing, spam or even Distributed Denial of Services (DDoS). If we are incapable of detecting such vulnerabilities quickly, even a DDoS attack temporary shutting down the services for couple of hours of huge companies like Amazon or Facebook would lead to hundreds of millions of loses for the latter. Metrics also come in handy for testing the correctness and performance of algorithms and furthermore for benchmarking graph processing systems. In the subsections below we will begin with the basic once such as order, size and will steadily turn our attention to more complex metrics like betweenness centrality and k-Core. Each of the below depicted metrics is supported by the GAME framework. Also most metrics will be used in *Chapter 6* in order to investigate how capable Darwini is in preserving graph properties.

The *order* of a graph is simply the number of nodes $n \in V$. The cardinality of V can be arbitrary positive integer which however, restricts the size of the graph. The *size* of a graph implies the number of connection between the nodes. The maximum amount of edges a graph can contain must not exceed n^2. As by the order, the size of a graph is either explicitly provided or it can be derived from the degree distribution during generation or import.

Density of a graph measures the relation between edges and vertices. A graph with number of edges close to the maximum number of permitted edges is considered to be dense, while on the other side a graph with very few connections is called a sparse graph. The density can be computed with the following formula:

$$D = \frac{(2)E}{V(V-1)}$$

for directed and undirected graphs. The factor of two is only for undirected graphs because an edge between (v_i, v_j) is construed as a bidirectional connection, i.e. $(v_i, v_j) \neq (v_j, v_i)$, in directed graphs.

Degree distribution of a graph is normally depicted in a plot, where on the x-axis we can find the distinct degrees in a graph G and on the y-axis the number of nodes sharing a common degree d. If we observe only undirected graphs, then the *outdegree* and *indegree distributions* are identical to the degree distribution because each edge is considered as bidirectional. In case of directed networks, the indegree distribution corresponds to a sequence of pair of numbers (k, d_{in}), where k vertices share the same indegree d_{in}. Outdegree can be retrieved the same way by observing only the outgoing edges in a graph.

The *largest degree* in a graph G is presented by the vertex v_i with highest number of edges, to which v_i is incident. In case of an directed graph the *largest indegree* and *largest outdegree* are computed from taking the maximum of ingoing and outgoing edges for each vertex, respectively.

The *average degree* is computed by dividing the size by the order from the original graph with the result being multiplied by two and can be expressed by the following equation:

$$Degree_{avg} = \frac{2E}{V}$$

The number of edges is multiplied by two because every edge having a start and end point increases the total degree of a network by two. Applying the same method for undirected graphs we can compute the *average indegree* and the *average outdegree*. However, if in case of an directed graph the multiplication by two is omitted since each edge is directed and thus enlarges the total outdegree and indegree by one.

Closeness centrality [New18] measures the closeness of a vertex v_i, defined as the reciprocal of the farness, that is

$$C(v_i) = \frac{1}{\sum_{j, v_j \neq v_i} d(v_i, v_j)}$$

with $d(v_i, v_j)$ being the shortest path distance from v_i to v_j between an arbitrary pair of vertices. The closeness score of all vertices are equal to zero when the input graph is disconnected. A disconnected graph is a graph, where no path exists between certain vertices. So, in the absence of a path between nodes v_i and v_j the shortest path $d(v_i, v_j)$ is equal to infinity "∞". In case of a

weakly connected graph, the closeness centrality of a subset of vertices will be zero. A graph is considered to be weakly connected when there is at least one pair of vertices (v_i, v_j) with, e.g. a path from v_i to v_j but no path in the opposite direction.

Harmonic centrality [New18] was designed to improve the closeness distribution for disconnected graphs, where closeness centrality comes short and thus unpractical for use. The harmonic centrality of a vertex v_i is defined as

$$H(v_i) = \frac{1}{\sum_{j, v_j \neq v_i} d(v_i, v_j)}$$

where $d(v_i, v_j)$ corresponds once again to the geodesic path (shortest path) distance between two nodes. However, in this case if $d(v_i, v_j) = \infty$, then we set the distance $d(v_i, v_j) = 0$, i.e. no weight is being added for missing paths. This way, one can measure the closeness of any node inside the domain of the connected component it participates in. To make the results more convenient each $H(v_i)$, scores can be normalized. So, a non-zero value rather than indicating a connection of a vertex to every other vertex in the network, denotes that a given index has some connectivity.

Betweenness centrality [New18] is the third and last discussed all-pairs shortest path based metric. Unlike the previous two centrality measures, betweenness centrality shows importance in terms of how much *in-between* certain vertex is. Betweenness centrality can be expressed using the following formula

$$B(v_i) = \sum_{v_j, v_k} \frac{N^i_{v_j, v_k}}{g_{v_j, v_k}}$$

with $N^i_{v_j, v_k}$ indicating the amount of shortest paths that go from v_j to v_k through v_i and g_{v_j, v_k} being the number of distinct geodesic paths going through arbitrary sequence of vertices.

The *(hop) radius* and *(hop) diameter* of any graph are determined by the below provided formulas

$$\min_{v_i \in V}(d(v_i, v_j)) \quad and \quad \max_{v_i \in V}(d(v_i, v_j))$$

as $min(d(v_i, v_j))$ corresponds to the eccentricity of vertex v_i, i.e. computing the shortest geodesic path for each vertex and choosing the shortest among those. As opposed to radius, the diameter is computed by retrieving the eccentricity of each vertex $v_i \in V$ and choosing the longest path among the set of shortest paths. If we observe an unweighted graph, then we measure the hop radius and hop diameter, i.e. the number of hops (number of edges) from vertex v_i to vertex v_j. In case of an empty graph, the diameter and radius are zero and equal to ∞ for disconnected graphs.

In graph theory, a *component* is considered to be *connected* when a subset of vertices $V_k \subseteq V$ can reach each other by paths and no other vertices can be added to this set. For example, an isolated node is itself a component as well as a strongly connected graph. Using a breadth-first search based iterator, the algorithm begins from a given node and explores all neighbours at the current depth before iteratively continuing with the nodes at the next depth.

Transitivity of a graph is based on the number of triangles (closed triplets) in the graph, i.e. for vertices v_i, v_j, v_k the edges (v_i, v_j), (v_j, v_k) and (v_k, v_i) exist, measured against the total number of connected triplets of nodes (closed or open wedges) and corresponds to the below equation:

$$T = \frac{3 \quad * \quad \#closedtriplets}{\#opentriplets \quad + \quad \#closedtriplets}$$

which is also referred to as the global clustering coefficient $C(G)$ since it can reveal the actual existence of tightly connected communities.

Clustering Coefficient, more commonly known as the local clustering coefficient $CC(v_i)$ measures the number of closed triplets a given node participates in against the maximum amount of possible triangles that could exist for a given node using

$$CC(v_i) = \frac{\|e_{jk} : v_j, v_k \in N_i, e_{j,k} \in E\|}{k(k-1)}$$

with N_i denoting the neighbourhood of vertex v_i and $k(k-1)$ denoting the maximum size within the neighbourhood matching the biggest amount of possible closed triplets that v_i could form if being connected to every node in G. For undirected graphs, each $CC(v_i)$ should be multiplied by two due to the absence of direction of edges.

PageRank (PR) [New18] is a famous algorithm for ranking webpages inside Google search engine by measuring the importance of website pages through iterative count of the number and quality of links. By using the following equation:

$$PR(v) = \sum_{v_j \in B_v} \frac{PR(v_j)}{L(v_j)}$$

where the page rank for website v_i is predetermined by the PageRank (PR) value of each page $v_j \in B_v$ linking to page v, divided by the cardinality of the set of outgoing links of page v_j. In terms of weighted graphs, a special version for handling weighted edges is used.

Finally, we turn our attention to the *k-Core* metric [New18]. A k-core consist of a maximal subset of vertices such that every single node is adjacent to at least k other vertices within the k-core's domain. It takes the set of degrees for a given graph and begins by removing the edges of nodes with degree less than k. In the following iterations the algorithm further checks for vertices with degree less than k. Since after we have removed a subset of vertices with degree less than k, there could be some more vertices with degree fewer than k because we have reduced the degree of the last by removing nodes that were connected to them in the previous iteration. Similarly the iterative approach continues until all remaining vertices have degree greater than or equal to k.

3.2 Generation Models

Graph generators give us the opportunity to create various real-world and non-real-world graphs using different set of input metrics. While some of the generation models like the Erdös-Renyi [ER+60] and Gilbert [Gil59] generate random graphs, others like Small-World [WS98] and Darwini [ELW+16] produce graphs matching important properties of real-world networks. The last is even capable of producing industry sized graphs without requiring the explicit structure of the original network. In the upcoming pages we will present a series of generation models. Those include two random graph generators Erdös-Renyi and Gilbert, R-MAT [CZF04], Small-World, Chung Lu model, Fast Reciprocal Degree, Darwini and others, which are discussed in length below. Direct comparison between the observed generators is provided at the end of *Chapter 2* delineating the strong sides and drawbacks of each model.

3.2.1 Erdös-Renyi Model

Erdós-Renyi Model (ER) [ER+60], also known as $G(n, m)$, is a random graph generator taking as input two natural numbers, corresponding to the amount of nodes n and edges m that the target graph should consist of, where m should not exceed $n(n-1)$. The algorithm returns any of the possible graphs with order n and size m with equal probability. During each iteration two vertices v_i and v_j are picked at random. In case an edge (v_i, v_j) already exist, the algorithm simply picks a new pair. This is done until there are no more remaining edges to be added resulting in a runtime of $O(E + max(V, E))$. Although ER random graphs are not considered as realistic models for reproducing real-world data, they are still very useful baseline for the development of generation models.

3.2.2 Gilbert Model

Gilbert [Gil59] or $G(n, p)$ model is another popular random generation method. The inputs to the model are a natural number indicating the number of nodes n and a real number p, known as the *density* parameter, with $0 \leq p \leq 1$, defining the probability of an edge between an arbitrary pair of vertices v_i and v_j. Therefore, the *density* parameter p is associated with the expectation value for the density of the target graph. The time complexity of the algorithm yields O(n + m) using [BB05], where m is the expected number of edges to be created.

3.2.3 R-MAT Model

R-MAT [CZF04], i.e. recursive Matrix, is a Kronecker-style [LCK+10] generation model capable of creating graphs with a power-law degree distribution. This approach takes as input the order, size and four additional parameters , a, b, c, d with cumulative value $a + b + c + d = 1$, known as probability parameter of selecting a field. It is worth mentioning that the structure of the output graph can dramatically change depending on the values assigned to the field probability parameters. Upon generation the adjacency matrix is split into four square fields of equal size and each field gets assigned a probability. Once a field is selected randomly, the adjacency matrix gets zoomed in and split once again into four distinct fields having the same probabilities. The above mentioned process is recursively repeated until we reach a single field, which corresponds to the addition of an edge between vertices v_i and v_j. The whole procedure is then being repeated for all remaining edges. In terms of execution time the complexity of R-MAT is in $O(E + max(V, E))$.

3.2.4 Small-World Model

Small-World [WS98] is a random graph generator that builds upon Erdös-Renyi graphs [ER+60] by producing graphs capturing average-short path and the high local clustering of small real-world graphs. The set of inputs include the number of vertices, an even mean degree value and a parameter p strictly laying in $0 \leq p \leq 1$. It starts with a fixed graph exhibiting high clustering and large path length. The output graph is formed by randomly rewiring fraction of the edges serving as shortcuts between vertices. The main disadvantage of the algorithm is impossibility of producing directed and dense networks.

3.2.5 Chung Lu Model

The *Chung Lu (CL)* [ACL01] model is a random graph generator used as a base in various other generators including in [KPPS14], [DKPS13]. It takes as an input a fixed degree sequence and creates edges between pair of vertices with probability p equal to their join degrees d_i, d_j divided by the total sum of degrees in G, rather than the constant value of ER graphs. The probability of an edge $e_{i,j}$ can be described using the following formula: $p(e_{i,j}) = \frac{d_i * d_j}{\sum_{k \in V} d_k}$. The Chung Lu configuration model produces graphs more closer to realistic networks because it can capture the heavily-tailed degree distribution nature of most real-world networks. The algorithm can also scale up to larger graphs since it required only O(m) execution time for adding all m links to G.

3.2.6 Fast Reciprocal Degree Model

Fast Reciprocal Degree (FRD) [DKPS13] is a generation algorithm able to match the in-, out- and reciprocal degree of arbitrary graph types. It is considered as a special variant of preferential attachment [AB02] by allowing edges between existing nodes to be created and thus introducing reciprocity. FRD starts with reciprocal distribution and selects k vertices that will form edges in both directions using a special algorithm for vertex selection. In the next step, it does exactly the same in order to match the indegree and outdegree distributions. Because Fast Reciprocal Degree algorithm is based on the Chung Lu model [ACL01] it manages to reproduce huge graphs very quickly results in total execution time of $O(m)$. However, its main disadvantage is the inapplicability to real-world network by trying to match only the degree distribution.

3.2.7 Community Guided Attachment Model

Community Guided Attachment (CGA) model was proposed as a solution to the surprisingly decreasing diameter of real-world networks in [LKF05]. CGA begins by constructing a tree with the leaves corresponding to the nodes of the original graph. The constructed tree has a depth of h and as we climb up the hierarchy the more unlikely it becomes that two nodes will form a link or even be part of the same community. Following this observation, the probability of generating an edge dramatically decreases as we further head to the top of the hierarchy to find the first common ancestor for both vertices part of distinct groups. Dynamic CGA builds on top of the original algorithm by enabling the addition of b new children to each current leaf at snap shot t. CGA argues that densification of real-world graphs can be explained through the decomposition of nodes into nested set of communities. However, this approach does not capture all observations from [LKF05], so a more sophisticated model known as *Forest Fire* was introduced by the same authors, addressing the missing observations.

3.2.8 Forest Fire Model

Forest Fire (FF) [LKF05] model was designed to exhibit both densification and decrease of the effective diameter as a graph continuously grows. It is based on having new nodes attached to the network by 'burning' through existing edges. The aim of FF, in addition to the properties already preserved by CGA, is to produce networks with heavy-tailed outdegree. The algorithm processes

nodes joining the network over time. Each node has its own 'center of gravity' in a specific region of the graph. As a node v_i joins the network, it creates a connection to a randomly chosen vertex v_j. Next, we generate a random number k and link v_i k-times with neighbours of v_j by choosing outgoing edges of v_j with higher probability compared to incoming links to v_j. Vertex v_i then continues the process recursively with the neighbours of v_j but with significantly reduced edge probability. Potentially, several newly introduced nodes will form multiple links and therefore serve as bridges between smaller communities across the network. This would lead to decrease in the diameter and a heavily-tailed outdegree distribution. The termination of the algorithm is secured by prohibiting visits of nodes multiple times.

3.2.9 Block-Two-Level Erdös-Renyi Model

Block-Two-Level Erdös-Renyi (BTER) model [KPPS14] is a state-of-the-art approach for capturing both degree and clustering coefficient distributions. BTER is developed to generate graphs with community structure. By accurately reproducing the local clustering coefficient of vertices, BTER is capable of matching real-world networks exhibiting community structure. The algorithm consists of three consecutive stages: In the first stage, BTER assigns target degree and clustering coefficient to each node, which should be matched in the final version of the graph. Following nodes get distributed across affinity blocks. Affinity blocks contain nodes sharing the same degree while the participants of each group should not exceed pre-defined upper bound denoted by $deg(v_i) + 1$. After assigning each vertex to an appropriate block, the algorithm proceeds with internal edge creation using a fixed probability for each possible edge, matching the ER model. Finally, the third phase takes care for linking the newly formed communities using the CL method and thus building the output graph. BTER can be used to generate large scale graphs due to its efficient edge insertion process - $O(log(d_{max})$ work per edge. However, a more promising model based on BTER presented in [ELW+16] is discussed below.

3.2.10 Darwini Model

Darwini [ELW+16] is a synthetic graph generator used to reproduce huge real-world networks while preserving some of their properties. It takes as an input the degree distribution and clustering coefficient distribution, from which the order and size of the graph can be derived. The Darwini approach is separated in three phases. Phase 1 processes the degree and clustering coefficient distributions. Then in stage 2, vertices are being grouped inside the so called "buckets", where vertices belonging to the same bucket should participate at the same amount of closed triplets. Following, Darwini has to merge the set of incomplete buckets in order to be able to closely approximate the number of closed wedges for each vertex in the newly merged bucket. In the final part of phase two, Darwini creates edges within the domain of a bucket while monitoring that no node exceeds its target degree. Otherwise, the clustering coefficient would be infeasible to correct in the final stage. And lastly, phase three manages the interconnection of vertices and thus establishing the connected target graph. The process of generating edges in both second and third phase is based on the same approach of finding pair of vertices, who do not match their target degree yet. More details about the custom implementation of Darwini and our growth prediction models as extension to the algorithm inside the GAME framework can be found in *Chapter 5*.

4 GAME Framework

Graph Analyses Measurement Environment (GAME) framework is a graph processing system developed at the University of Stuttgart, Germany. The application provides numerous ways to either import or export a graph. Those include the well known edge list and vertex list, but it also supports custom made formats, giving the user the opportunity to dynamically add custom parameters. A couple of graph generators are already implemented while guaranteeing fast execution and memory efficient usage. Those include random generators, R-MAT, Small-World. The most recent addition being our implementation of the synthetic graph generator Darwini including both extensions over which we deliberate in Section 5.2. Additional regular graph generators like the Star and Circle graph are expected to be embedded in the application in the incoming months. Computed metrics, outlined later in this chapter, can be exported as raw data facilitating the evaluation and visualization process found in Chapter 6. GAME can further be executed on remote machine benefiting of more computational power than regular desktop computers. In the following sub-sections we will set our focus on how exactly are graphs represented and which algorithms are parallelized. We will also point out the custom implemented queue and job management system. But before all that we would like to mention the used libraries and frameworks, which helped making this graph processing system more convenient for usage.

4.1 Libraries, Frameworks and Plug-Ins

Employment of external libraries and plug-ins for the development of large frameworks such as a graph processing system can be described as mandatory in order to produce an overall well build software. Plug-ins like Maven [MVM10] facilitate developers of building uniform build systems or introduce a new feature to an already existing system. A library on the other side is a set of classes with already provided functions ready to be attached to existing program using a well defined interface. OpenJFX [CVW19] is an open source client application platform where JavaFX was developed. Using the latter enables developers to design their program using the popular Model-View-Controller (MVC) software design pattern. OpenJFX provides drag and drop layout tool for designing rich JavaFX application UI called Scene Builder [VCG+18b] separately stored from logic and thus eliminating boiler code. JGraphT [MKNS20] is a library composed of numerous data structures and efficient algorithms used for graph processing and metric computations. A logging framework such as Apache Log4j 2 [Gup03] is also mandatory in large systems in order to log arbitrary type of exceptions or events that occur during program execution. Database Connectivity (DBC) Driver [Joh14] supporting major operation systems for SQLite was also required by GAME so that graphs and computed metrics can be stored for future comparisons and analyses. A light-weight and convenient to use library like XChart [WPHV15] is used to create diagrams of different type and display raw data. ControlsFX library [VCG+18a] offers a wide variety of UI controls addressing the lack of validation support in JavaFX. To make the user-program

interaction more easy, users receive hints about invalid input. The library OpenCSV [New18] facilitates parsing of comma separated values occurring for example during import of a graph represented by an arbitrary list format. When developing large programs key aspect is to make sure that every new feature integrated in a project works as expected. Therefore, a testing framework as JUnit 5 [Gar17] is required to test processes like generation, importation, exportation of graphs, metric calculations and others. Its features cover different test styles and provide flexibility during testing. The set of above mentioned libraries, framework and plug-ins perfectly fit into this project providing modern and convenient design for the user while facilitating developers mainly focusing on the backend logic, in this case the variety of generation and metric algorithms.

4.2 Custom Data Structure

GAME uses a data structure called *Compressed Sparse Row (CSR)* [BFF+09] because it is more memory efficient than the standard adjacency matrix. CSR consists of a *RowPtr* and *ColInd*, where the first contains the amount of connections for a given node while the later stores information about links to other nodes. The major disadvantage of this data structure is the fact that edge insertions are very costly when they are not inserted in ascending manner, i.e. edge (1,3) is added before the edge (1,7) to CSR. If that is not the case, it takes long time to shift the positions of each ColInd element and also no further edges can be inserted during this process. To address the above mentioned issues, a dynamic CSR was designed by splitting the RowPtr and ColInd into smaller lists. After all edges have been added merging threads take care for the accurate assembling of the partial RowPtr and ColInd lists thus forming the original CSR object.

4.3 Jobs and Queue Management

As we previously mentioned, users can import, export, generate graph or compute metrics. Each of these requests corresponds to a specific job. For example, if user wants to create a new graph using arbitrary generation model, a new GenerationJob instance is being created and handed to the CalculationManager, which adds it to the queue. When a job is attached to the end of the queue, a manually implemented JobRunner is notified for its presence. This job gets either directly executed in the absence of other jobs waiting in the queue or when the currently executing job does not utilize all available cores and both share the same CSR object. Otherwise, the job is put into waiting state until all remaining jobs at forward positions are executed. Because the Job class is actually a subclass of Thread, the Main JavaFX Thread (updating user interface) and jobs can run in parallel without interfering with one another. Once a job terminates, its corresponding JobResultHandler displays the results in the UI or informs the user about one or more exceptions caught during execution. It also removes the associated to the given job entries from the queue and thread structures. However, the user has not only the opportunity to add multiple jobs to the queue but also can abort them at any given time. On the other side this feature has a trade off. Although the user could cancel an accidentally started job during any time, this type of responsiveness comes with overall performance drawbacks. The special usage of a heart beat system demands each job to send a total number of progress steps called at the equal time intervals. During each heart beat can be determined if a cancel request for the job is available.

4.4 Importer and Exporter

The importer and exporter were not only implemented to be efficient but also exhibit flexibility by allowing the user to choose between edge list, vertex list and a custom format adaptable to a arbitrary input or output format. Directed and weighted graph are also supported. However, there is a difference in the underlying structure of the importer and exporter. The importer is designed as a parser, which corresponds to Deterministic Finite Automate (DFA) having different states and transitions from one state to another represented by dividers and loops. It reads the input line by line ensuring better overall performance and for each processed token, the appropriate state is entered, while the read value is stored correctly in the corresponding data structure. The termination of the importer is guaranteed either by parsing through the input or by reaching a state, which yields a sequence of tokens not matching the selected input format. The exporter on the other side simply iterates through the selected syntax and writes the values of the states it has entered to an output file. It does not have the ability to recognise when a loop has stopped, therefore the special character & has to be appended right after the end of a loop.

4.5 Metrics

Without metrics we would not be able to analyse the underlying structure of graphs. Computing just a few metrics like degree and clustering coefficient distributions should provide us with information about the communities formed in the different regions of the graph and how dense they are. While some of the metrics like e.g. density and number of connected components can be computed fast, there are some more sophisticated characteristics of networks that could not be computed for larger graphs because of their high computational cost. At the time of writing a total of 24 different metrics were integrated inside the graph processing system GAME. All metrics falling under the centrality category are executed by a single thread. The reason for this is that they are based on the Dijkstra algorithm, where during each iteration an edge gets removed. So, in order to execute them using multiple threads, each of them should become its own copy of the graph. For graphs requiring several gigabytes of memory space it is basically impossible to fit all copies, the program code and all remaining relevant data in the main memory of a single computation unit. The only possibility would be to execute the centrality metrics on different machines at the same time but unfortunately GAME currently does not support execution inside a multi machine cluster. Summary of all available metrics can be found in Table 4.1 while detailed description about each metric can be found in Section 3.1. A cross in the second column of Table 4.1 means that the given metric entry was manually implemented from scratch. Otherwise, the implementation from JGraphT library [MKNS20] was adjusted to fit inside the graph processing system. Some of the metrics from JGraphT have been parallelized in order to reduce their execution time and thus make them applicable for large graphs.

4.6 Generation Models

Graph generators give us the opportunity to create various real-world and non-real-world graphs using different set of input metrics. While some of the generation models like the Erdös-Renyi [ER+60] and Gilbert [Gil59] models generate random graphs, others like Small-World [WS98] and

Metric	Library	Parallelized	Given or Calculated Automatically
Directed	x	No	Metric is given before Import/Generation
Weighted	x	No	Metric is given before Import/Generation
Order	x	No	Calculated During Import/Generation
Size	x	No	Calculated During Import/Generation
Density	x	No	Calculated During Import/Generation
Degree	x	Yes	-
Average Degree	x	No	-
Largest Degree	x	Yes	-
Outdegree	x	Yes	-
Average Outdegree	x	No	-
Largest Outdegree	x	Yes	-
Indegree	x	Yes	-
Average Indegree	x	No	-
Largest Indegree	x	Yes	-
Closeness Centrality	JGraphT	No	-
Harmonic Centrality	JGraphT	No	-
Betweenness Centrality	JGraphT	No	-
(Hop) Radius	JGraphT	Yes	-
(Hop) Diameter	JGraphT	Yes	-
Connected Components	JGraphT	No	-
Transitivity	JGraphT	Yes	-
Clustering Coefficient	x	Yes	-
PageRank	JGraphT	Yes	-
k-Core	x	No	-

Table 4.1: List of all currently supported metrics by the GAME framework. The 'x' sign inside the 'Library' column denotes that the metric was implemented from base. Remaining metric implementations were derived from the JGraphT library. The algorithms were further manually adjusted and extended to achieve overall better execution times.

Darwini [ELW+16] produce graphs matching not only the size but also other important properties of real-world graphs such as social networks.The graph processing system at the time of writing include two random graph generators Erdös-Renyi and Gilbert. Power-law based model R-MAT [CZF04], and the Small-World generator capable of building small real-world graphs can also be found in GAME. During this thesis we implemented Darwini along with two growth prediction models in GAME aiming to investigate how well can the model preserve diverse graph properties. Both growth extension models were designed and added to the graph processing system focusing at developing an approach capable of predicting the evaluation process of real-world networks. Detailed discussion of the latter and Darwini can be found in the coming chapter. Each above mentioned graph generation model supports parallelisation to speed-up execution time.

5 Darwini and Growth Model Extensions

Darwini is a synthetic graph generator capable of reproducing several core characteristics of real graphs. Like existing state-of-the-art algorithms it can reproduce accurately the degree distribution. In addition, Darwini also captures clustering coefficient at fine granularity [ELW+16]. Generation of large real-world graphs can be achieved by parallelizing the model and distributing the work across several machines inside a cluster, therefore making this approach suitable for industry use. In the following sections, we will firstly take a detailed look into the algorithm and its divergence and improvement over the Block Two-Level Erdös-Rényi (BTER) model [KPPS14], on which Darwini is based. In the next section we turn our attention to the growth model approaches that we developed in order to understand the inherent growth process of real-world networks implemented in the first stage of Darwini. Finally, we discuss the parallel implementation of Darwini inside the GAME framework and missing features cause by several limitations and possible future improvements.

5.1 Algorithm

Unlike growth models such as preferential attachment, whose aim is to predict the natural growth process of networks, Darwini generates synthetic graphs by making use of the computed degree and clustering coefficient distributions of the original graph. It is challenging to connect two vertices while simultaneously striving to match a given degree distribution due to scale of graphs with billions of vertices and the number of possible connections between those. However, Darwini is capable of matching the degree and clustering coefficient distribution of the source graph by using a mixture of edge generation processes [ELW+16] that iteratively add new edges. At the same time this process is done by grouping vertices inside (small) communities and connecting them with one another. Then vertices of different communities are being interconnected and thus form the synthetic graph.

The algorithm is composed of three sequentially executed stages/phases. Inside the first stage, Darwini assigns target degree and target clustering coefficient score to each vertex that should be matched at the end. Before that, in case we include our extension, if an arbitrary growth model was selected by the user, a graph consisting of twice the amount of vertices will be generated. The number of edges for the larger graph will be proportional to the increased quantity of nodes. In the next phase, Darwini distributes vertices into small communities also known as $buckets$ [ELW+16] or *affinity blocks* [KPPS14] and creates links between nodes inside the community's domain while approximating target degree and clustering coefficient scores. Lastly, in the third stage, Darwini forms the graph's final version by picking nodes randomly at uniform from different communities. The way, in which communities are formed during the second stage is essential for matching the target degree and local clustering coefficient properties.

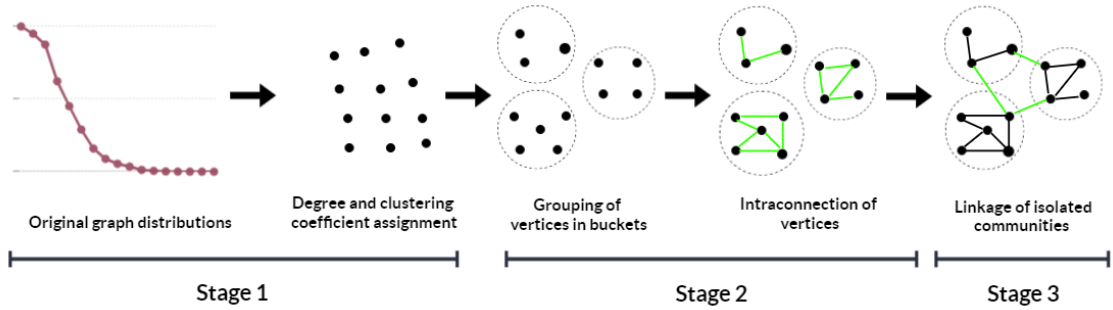

Figure 5.1: Darwini model consisting of three distinct stages. Newly inserted edges are coloured in green.

5.1.1 Assignment of Target Degree and Clustering Coefficient

The first stage, depicted in the above placed Figure 5.1, of the algorithm can also be described as the processing step. During this stage every vertex gets assigned a target degree and a target clustering coefficient provided from the original graph. While iterating through the input degree and clustering coefficient distribution, if not provided, we compute the order and size of the graph. Suppose that the generated synthetic graph $G = (V, E)$ contains $n = |V|$ nodes and $m = |E|$ edges, with $v_i \in V, 0 \leq v_i \leq n - 1$ representing the set of vertices. Then, each vertex v_i will be assigned a target degree d_i and target clustering coefficient c_i. The original algorithm takes as input the degree distribution, denoted by F_{deg}, and the local clustering distribution $F_{cc}(d)$ per degree d. However apart of F_{deg}, our implementation inside the GAME framework takes as input just the target clustering coefficient F_{cc}. The reason for this divergence is provoked by the inefficient memory usage. Despite of changing the way how the clustering coefficient distribution is stored, we are still able to retrieved the correct score for each vertex. Therefore, the algorithm should still be capable of matching the local clustering property at fine granularity. More details about this issue can be found at the end of this section. After processing the input data, the algorithm continues with execution of stage two, where vertices are grouped in *buckets* artificially trying to match the structure of the original graph.

5.1.2 Intraconnection of Vertices

The number of possible combinations in which vertices can be connected, especially for large scale graphs, makes it extremely difficult to match the degree and local clustering coefficient of vertices. For example, the BTER [KPPS14] model solves this issue by presuming that vertices having the same degree should also have the same clustering coefficient. This way given the target degree and target clustering coefficient for vertices inside a group, random edges can be placed between two nodes. By the addition of random edges the algorithm should simultaneously match both the degree and clustering coefficient properties. The conjecture in [KPPS14], however resulted in inaccurate clustering coefficient results, as stated in [ELW+16]. Therefore, another solution to this problem is required.

This is the place where Darwini [ELW+16] builds up over the BTER [KPPS14] model and thus delivers more accurate results. Darwini's first job is to match the expected number of triangles, to which a given vertex should belong to, rather than directly striving to meet the desired degree and clustering coefficient scores. At this point one would probably ask himself: How can we acquire the expected number of triangles when only the target degree and target clustering coefficient being at disposal? To answer this question, we would need to take a deeper look at the definition of local clustering coefficient:

$$c_i = \frac{2N_{\Delta,i}}{d_i(d_i - 1)}$$

with $N_{\Delta,i}$ being the number of triangles at which vertex v_i should participate. So, by transforming the formula we can actually retrieve the number of triangles $N_{\Delta,i}$, for each vertex v_i by using the following equation:

$$N_{\Delta,i} = \frac{c_i(d_i(d_i - 1))}{2}$$

Since we already know the value of the target degree d_i and the target clustering coefficient c_i for each vertex $v_i \in V$, we are able to estimate the exact number of triangles for each vertex using only the values d_i and c_i.

Now we face the following challenge: How should we group vertices in order to meet the N_Δ score for each node? To do this we must ensure that the there are enough vertices within the group with that a given node can link and thus form closed triplets. If we accept that vertex v_i can be found in $N_{\Delta,i}$ triangles there are two possible scenarios. Vertex v_i has already reached its target degree and clustering coefficient, respectively. If it has fewer edges than expected, then v_i has to be connected with other vertices in such a way that does not harm the $N_{\Delta,i}$ score.

Accomplishing these targets is realized by the usage of groups, also known as *buckets* [ELW+16] or *affinity blocks* [KPPS14]. Vertices sharing the same bucket have matching $N_{\Delta,i}$ scores. As being illustrated in the second stage of Figure 5.1 there are a total of three affinity blocks. These were distributed is such a way using the above formula expressing the expected number of triangles per vertex. After all vertices are grouped, edges are continuously added within the affinity block's domain with fixed probability P_e, which will be discussed in length later. Hence, inserting edges inside groups can be achieved using the Erdös-Renyi [ER+60] model. Both P_e and N_Δ are related to previously mentioned targets. We know that the probability of a triple of nodes creating a closed wedge equals $P_\Delta = P_e^3$. So, every single vertex can be part of up to $N_\Delta = \frac{(n-1)(n-2)}{2}$ closed triplets. Consequently the anticipated sum of triangles per vertex can be expressed using the following equation:

$$\hat{N}_\Delta = P_\Delta * N_\Delta = P_e^3 \frac{(n-1)(n-2)}{2}$$

Using the above standing definition for the desired amount of closed triplets for a vertex we can build a bucket with the desired number of closed triangles$\hat{N}_\Delta$ by finding suitable values for the probability P_e and the size of the affinity block $\|B\|$ denoted by the number of nodes in it.

Although there are several combinations of P_e and n that can be applied to match the expected number of triangles, their values are actually dependent on the constraints of the bucket's size. We must first ensure that each bucket contains enough vertices in order to form the expected number of triangles for every vertex. The lower bound can be expressed with the following equation:

$$n \geq \sqrt{c_i d_i(d_i - 1)} = n_{B,min}, \forall i \in B$$

since $P_e < 1$ and the previous two formulas hold. In the next step, we must make sure that none of the vertices exceeds it target degree. By doing this we can later correct the clustering coefficient because after the end of stage 2, every vertex has a higher clustering coefficient compared to the target one. Exception are all vertices who already match their target degree after the second phase, i.e. by already participating in the desired number of connections the vertex also has matched the target clustering coefficient. So, to avert the opportunity of vertices surpassing their target degree, the cardinality of each group should be limited up to:

$$n \leq min_{i \in B}(d_i) + 1 = n_{B,max}$$

with $min_{i \in B}(d_i)$ being the vertex with the lowest target degree inside the bucket B and $n_{B,max}$ denoting the upper bound of nodes in affinity block B. Now that a vertex $v_i \in B$ can not establish more than d_i connections in phase 2 since multiple edges and self-loops are not permitted, we can conclude for the set of vertices $V_i = \{v_i \in B_k \quad | \quad deg(v_i) = min_{i \in B_k}(d_i)\}$, that they already match their target degree and should be participating in the desired number of closed triplets. In the third stage none of the these vertices should be selected for interconnection, otherwise they will have higher actual degree and lower clustering coefficient.

Lastly, we should compute the probability P_e for a given affinity block B by setting n within the pre-defined bounds. As stated in [ELW+16], Darwini tends to pick the lower bound $n_{B,min}$ for each group resulting in:

$$P_e = \sqrt[3]{\frac{2\hat{N}_{\Delta,B}}{(n_{B,min} - 1)(n_{B,min} - 2)}}$$

Taking advantage of the above defined values, Darwini is capable of grouping and connecting vertices inside buckets in three successive parts. First, we begin with the distribution of vertices in buckets based on their N_Δ score. Next, because there would not be enough vertices for some subset of buckets, which we refer to as incomplete buckets, two or more incomplete buckets would be merged into a single complete one. This has to be done so that each node can participate in approximately the desired number of triangles. At the end, once Darwini is provided with a set of complete buckets, the edge insertion process between vertices within a common domain starts. A detailed view about each step can be found in the upcoming subsections.

Vertex Distribution in Buckets

Darwini starts the execution of phase 2 by taking the target degree and clustering coefficient distributions. After initializing an empty set of buckets S, Darwini computes the expected number of closed triplets $N_{\Delta,i}$ for each vertex v_i and selects a bucket based on the computed $N_{\Delta,i}$ value. Figure 5.1 delineates the process in the first step of stage two, where each group has a specified $N_{\Delta,i}$ value. To select a bucket, the following two conditions have to be fulfilled: First, a bucket with $N_{\Delta,B}$ value should be available in S and the amount of members in it should not be more than $min_{j \in B}(d_j)$. If an incomplete group with $N_{\Delta,B}$ score is found, Darwini adds vertex v_i into the last and label it as full, i.e. complete block, where no more nodes can be filled, if the amount of vertices within it has reached $n_{B,max}$. It is worth mentioning that each time a vertex has been added

to a given group, a check is made ensuring that the $n_{B,max}$ value gets updated if a vertex with lower degree than the previously known minimum degree has joined. This ensures not more than target degree d_i edges within this community. Otherwise, when either all groups in S having $N_{\Delta,B}$ value are already marked as full or no such group existed to this moment, a new group is created. However, before going ahead with the insertion of edges, Darwini has to check and consequently merge incomplete groups.

Merge of Incomplete Buckets

As mentioned earlier once the process of grouping vertices into buckets finishes, it is possible that there is a set of buckets with insufficient amount of vertices to produce the required number of triangles. The solution to this problem is provided by merging the incomplete buckets into larger ones. To do this, Darwini first discovers all groups, which do not have the minimum necessary number of nodes $n_{B,min}$ in them and stores those in a set of incomplete blocks S_u. Before merging buckets, Darwini sorts the set of incomplete buckets S_u in ascending N_Δ order. Then a bucket $B \in S_u$ is being subsequently merged with other buckets from S_u until it becomes full, i.e. $B.size > min_{j \in B}(d_j)$. After being labeled as full, the bucket joins once again the set of complete groups. As you have probably already perceived, that merging leads to placement of vertices with distinct $N_{\Delta,i}$ to be part of the same group and there is no single value for P_e that could estimate precisely $N_{\Delta,i}$ for every vertex. Hence, this would lead to inaccurate end values of the clustering coefficient. Although this approach would not result in the necessary clustering coefficient for every participant in the group, it would provide better outcome than by leaving the affinity blocks incomplete, as stated in [ELW+16]. Because the incomplete buckets get merged with others blocks having close N_Δ score, the necessary amount of triangles per vertex is estimated with less divergence from the actual value than by randomly combining buckets. Having finished the distribution of vertices and merging of buckets, Darwini has prepared everything essential to start the edge creation process.

Edge Creation Within Buckets

Having a complete set of buckets, Darwini begins with the additions of edges using the Erdös-Renyi [ER+60] model in expectation of producing the desired number of triangles per vertex. Darwini does this by proceeding each bucket and adding edges with a pre-defined fixed probability P_e discussed premature. In this case we do not permit the occurrence of self-loops or multiple edges. The second step of stage 2 in Figure 5.1 sketches the procedure with each group/bucket being observed at this point of execution as a single graph.

After finishing execution of this stage, each vertex should already be participating in the target number of triangles. But as mentioned earlier it should not meet its target degree and clustering coefficient yet. For the majority of vertices, except some whose degree matches the minimum degree of the bucket $min_{i \in B}(d(i))$, their current degree should be lower than the target one and they should currently posses higher clustering coefficient than the target value. Both scores will be corrected by interconnecting vertices. More over the exact measures taken to emend the values of each vertex can be found in the forthcoming section.

5.1.3 Interconnection of Vertices

In the last stage of the algorithm, Darwini tempts to add the excess degree $d_{excess,i} = d_i - d_{cur,i}$ for each vertex. Simultaneously, it needs to preserve the number of triangles per vertex by linking a pair of nodes across buckets. Therefore, the contribution of closed triplets from random interconnection is highly unlikely. Through iterative addition of edges, Darwini eventually meets the target degree and clustering coefficient. It starts by iterating over the set of vertices. For each vertex $v_i \in V$ it first has to check, whether or not the target degree is already reached. In case the $d_{excess,i} \geq 1$, vertex v_j is being selected randomly at uniform. Since the degree of each vertex has to be monitored, v_i can be connected with v_j only when $d_{excess,j} \geq 1$ holds and $i \neq j$.

This approach allows us to connect vertices fast without having to search through the entire vertex set. From one side it reduces the execution time by removing huge overhead and thus making the algorithm applicable for generating real-world industry graphs such as online social networks [ELW+16]. On the other side however, since the selection of a candidate v_j for v_i happens uniformly leads to the following problem: Typical for the composition of social networks is that the better part of users tend to have few links, whereas minor part are adjacent to huge amount of people. Consequently, low degree vertices will reach their target degree relatively faster than high degree nodes. This will result in inaccurate results at the end of the degree distribution curve since the algorithm was designed to terminate either when all remaining edges were created or by reaching a maximum number of iterations [ELW+16]. Similarly, one can find the same problem in the BTER model [KPPS14], which has also been verified by the authors of the Darwini.

Clearly the random selection is not enough to produce the desired results. Therefore, a supplementary process for finding endpoint for high degree vertices without being necessary to search through the complete graph is needed. The authors of Darwini have designed a process addressing this issue and it works the following way: In case at least two more nodes have not found enough candidates yet, Darwini randomly shuffles them into groups of equal size, starting with size of 2 and increasing exponentially in every next iteration. The increasing size of the groups with shuffled vertices indicates expansion of the search domain and thus making it less likely that a new connection will form new triangle. The increasing size of the groups with shuffled vertices indicates expansion of the search domain and thus making it less likely that a new connection will form new triangle. Each pair of vertices inside those groups is a possible candidate for the establishment of connection.

Unlike the probability of connecting vertices within buckets in phase 2, which remained constant during the entire internal edge creation process, here the probability is dynamically adjusted for each pair of nodes. Darwini was designed to maintain degree correlation, i.e. a correlation between the degree of vertex and all its neighbours can be detected in social networks [UKBM11]. So, in this phase Darwini not only preserves the local clustering coefficient but also the degree correlation by assigning a higher edge probability for pair of nodes with closer degree than a pair having large divergence in their degrees. As it can be observed from the results in [ELW+16], this approach also produces good join-degree distribution. So, after Darwini has created edges across communities, the generation process comes to an end returning synthetic graph matching the original one, as illustarted in Figure 5.1, where green edges have been inserted in the third and final stage of Darwini.

5.2 Growth Prediction Model Extensions to Darwini

While evaluating how well Darwini preserves graph properties of disparate graph types, we asked ourselves: Is it possible to create a growth model that can predict the future structure of a graph and still maintain several attributes, like e.g. the degree distribution? Is this model applicable to all kinds off graphs? So, to answer these questions we came up with the *Random Increase of Degree (RID)* growth prediction model. We found empirically however, that *RID* is not appropriate for real-world social networks, where the largest degree of the graph grows over time and the diameter shrinks [LKF05]. This motivated us to design a *Linear Preferential Attachment* growth model suitable for graphs with high amount of low degree nodes and few vertices having high degree corresponding to what we observe in online social networks.

Each of the models is integrated inside the first stage of our Darwini implementation inside the GAME framework. We start with the set of vertices V and the set of edges E of the original graph. For simplicity we want to predict the structure of a source graph with twice as much vertices $(2n)$, although overall increase of cn, $0 < c < \infty$ is theoretically possible. Consequently the following question emerged: How can we determine the number of edges that the multiple graph should possess? After experimenting we came up with the idea that the number of nodes n is proportional to the number of edges m. If we assume that a graph G has n nodes and m edges at a snapshot $G(t)$, with $n \leq e$, then we can say that $e(t) \propto n(t)^{\alpha}$, i.e. a relation between the number of nodes and the amount edges exists at any given time. So, the multiple graph G at snapshot $G(t')$ will have in our case $n(t') = 2n$ nodes and $e(t') \propto n(t')^{\alpha} = (2n)^{\alpha}$ edges.

Nevertheless we needed an evidence that would confirm our hypothesis about the association between the quantity of vertices and edges. If we take a look back at the Facebook graph example from the *Chapter 1* it is obvious that the value of α increases over time, contradicting our expectation. After researching several papers over the evolution and structure of online graphs [UKBM11], [LKF05], [KNT10], we found in [LKF05], that a relation between the order and size of a graph exist. This paper provides proves for different types of real-world networks, including autonomous systems, citation, patent citation graphs and affiliation graphs. More importantly their study was made not only over large datasets but also for each network by computing the constant α at equally distanced time intervals they came to the conclusion that it remains constant over the observed time periods. The relation between the number of nodes and edges for each graph was tested over a large time span, i.e. acquiring more convincing results than one would receive from single snapshots, i.e. static networks.

In the forthcoming subsections will be present our growth prediction models in detail and will point out their domain of usage. We will begin with the *Random Increase of Degree* and then we will turn our attention to the more complex *Linear Preferential Attachment* model.

Random Increase of Degree Growth Model

Random Increase of Degree (RID) takes as input the number of nodes and edges of the target graph as well as the degree and clustering coefficient distribution. Then for each vertex of the original graph we copy its target degree and local clustering coefficient score and create a new vertex with the same target scores. Next, we need to add all remaining edges since the graph should have $(2n)^{\alpha}$

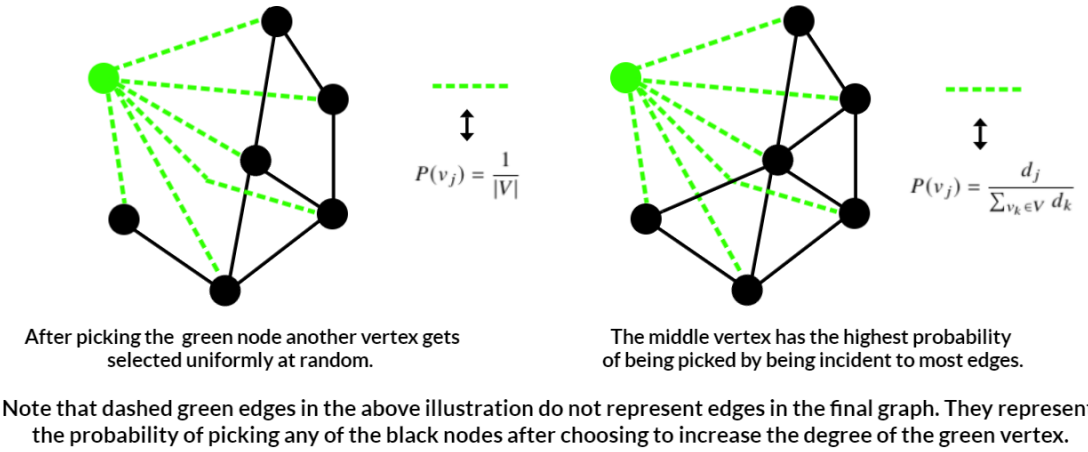

Figure 5.2: The above depicted graphs summary how a pair of nodes is chosen to increase their target degrees using Random Increase of Degree (left) and Linear Preferential Attachment (right).

edges and after having two times more vertices, with the second half having the same cumulative total degree as the first one, resulting in $2e = 2(n^\alpha)$. Thus the remaining edges that need to be inserted are equal to: $remainingEdges = (2n)^\alpha - 2(n^\alpha)$.

In the next step, we need to add the remaining edges before generating the graph. This is done by simply picking two nodes v_i and v_j randomly at uniform. If $v_i \neq v_j$ holds, as we do not want any self-edges, then we increase the degree of both vertices by one, which corresponds to the addition of an edge having two endpoints. The process is delineated on the left in Figure 5.2. After picking the green node, RID selects uniformly at random any vertex from the set and following increases the degree of both nodes. However, we can not guarantee that an edge between this pair of nodes will be present in the output graph. Because it is highly unexpected that the same node will be picked twice $\frac{1}{(2n)^2}$ in the same iteration, we deduce the running time of the prediction model to be in $O(remainingEdges)$.

The main advantage of this model is that by randomly increasing the degrees of vertices we preserve the degree distribution curve, i.e. we are able to produce graphs preserving the structure of the original graph but with enlarged number of small communities. A real life example of the usage of this model can be found in online video games, where players join the same lobby (community) and play against each other. In most cases a number of pre-defined users has to join the lobby, e.g. in Counter-Strike 10 people divided in two teams have to join a lobby in order to play a competitive match, before the game starts. Both the servers and the users can be described as vertices where a user joins the graph when he/she has found a free lobby. While it is hard to say the exact number of players and lobbies a single server can hold, we can say that it is not more than some empirically defined number k providing the best game experience for every player. So, going back to our model if the game becomes rapidly popular the demand of additional servers will be inevitable, i.e. a larger network would need to be designed. By using *RID* we could extend the network matching the amount of servers required to be able to host most or in best case, if the company has the budget, all players that have downloaded the game. The same holds for other real-world domains such

as online meetings between employees of the same or individual companies, online lectures for students at school or at the university and many other domains due to the fast spreading variants of COVID-19.

Although this model is not applicable for each type of graph, it still could have its role in the above mentioned areas. How well *RID* can preserve properties of the original graph can be found in Chapter 6.

Linear Preferential Attachment (Rich Get Richer) Growth Model

Our next approach, *rich get richer* growth model, is based on the linear preferential attachment [AB02]. Therefore we will refer to it as Linear Preferential Attachment (LPA). The inputs again include the target order and size and both degree and clustering coefficient distributions. Following we copy the value of each vertex of the source graph and create a new vertex with the same clustering coefficient and target degree of 0. We set the target degree to be 0 since the *rich get richer* approach is based on the preferential attachment model, where a vertex joins the network at any time and connects to l vertices, $l \leq v = \{v_i \in V\}$. Again, we need to insert $(2n)^{\alpha} - m$ remaining edges.

Because the *rich get richer* method begins with m_0 vertices corresponding to the original graph, we increase the degree of each to be added vertex and a randomly selected one from the graph and thus ensuring that all new vertices will be part of the output graph. This is done because the accumulative degree of the source graph is already high compared to the single degree of vertices. We found through experimenting that if we do not set the current degree to one for all joining the graph vertices, then a huge amount, usually between 30-60%, of nodes remain disconnected.

In the subsequent step, the second half of nodes is processed by connecting them to other vertices already participating in the graph with variable probability. The probability that a vertex v_i will be connected to v_j is

$$P(v_j) = \frac{d_j}{\sum_{v_k \in V} d_k}$$

with d_j being the degree of vertex v_j and $\sum_{v_k \in V} d_k$ being the sum made over the degree of all pre-existing nodes. So, it is possible that any of the newly added nodes is connected to two or more other nodes in the graph at time G(t). However, vertices are more likely to establish connections with high degree vertices than with low degree nodes, a feature known either as *rich get richer* or *preferential attachment* [AB02]. By using LPA in Figure 5.2 on the right-hand side we can see that the middle vertex holds the highest probability of being picked. The probability of each node being selected is computed using the above located formula. Since in Figure 5.2 the node located in the middle of the graph has five connections, it possesses the highest probability of being picked. Each remaining node gets assigned a lower probability due to the fewer amount of links they have compared to previous. Even though, the green vertex could potentially select any vertex or in the most unlikely scenario each node in the graph, i.e. the green vertex can increase its target degree multiple times by picking two or more nodes in the same iteration. Exactly the same way as in the previous model, each time an edge is inserted the number of $remainingEdges$ is adjusted, i.e. decremented by one. The above mentioned iterative process continues until the target size of the output graph has been reached.

By using the *rich get richer* model we can generate a larger graph with similar to a power-law degree distribution. And unlike in the previous approach, the structure of the graph changes due to the fact that the high degree nodes of the output graph connect more communities than the high degree nodes of the original graph. Such type of graphs can be found in online social networks like Facebook, Instagram and entertainment platforms like YouTube and TikTok. When a user joins a social network it is expected that he/she 'follows' a high degree user represented by film actors, music stars, sport players, political figures and others. This corresponds to the preferential attachment mechanism, where the more popular a given person is, the more popular to become is anticipated. Apart from the links with several popular people that a users would normally have, they will also establish connections with their friends part of the network. Another way of forming communities in online social networks is by joining a group of specific topic, where all members have a common interest. The main issue is that online networks and platforms make certain restrictions. Facebook for instance, limits the number of participants in group chats up to 250 and audio calls up to 50, respectively. Instagram restricts the maximum number of users inside a message group only to 32. Therefore, to address this issues one would need to extend the networks by attaching additional servers in regions where such demand can be observed. So, when generating larger synthetic graphs based on *rich get richer* growth model, we can identify possible future expansion of smaller communities that its significant growth is based on an arbitrary event or trend.

From the above summary of our growth prediction model, it would be logical that the diameter of the graph shrinks as online networks get larger and more dense. The number of connected components should also decrease by having several larger communities and also more nodes (people) connected to high degree vertices (popular personalities). The evidence or repudiation of those assumptions are provided in detail in Chapter 6.

5.3 Implementation

In this section we will go through our Darwini implementation inside the GAME framework. Since there is a huge difference between the Apache Giraph based on the vertex-centric programming model [TBC+13], where e.g. each vertex inherits from a predefined class and is composed of vertex values type, edge value type, ID type and a message type, thus allowing execution of graph algorithms across different machines inside a cluster. Vertices inside GAME are represented as integers and do not hold any additional information. The upcoming subsections summary how we adjusted the original algorithm to fit inside the GAME processing system, including the growth prediction models described earlier.

5.3.1 Input Processing and Growth Prediction Models

Darwini starts by invoking the first stage, where input data is being processed. It takes the degree and clustering coefficient distributions, size and order if available and an extra value indicating whether or not a larger graph should be generated using either growth model. Independent of the chosen approach, Darwini has to assign for each vertex a unique ID. For the standard approach it assigns ID in the range (0, n-1) while for RID and LPA unique identifier in [0;2n], along with target degree and target clustering coefficient. The values of the last three are stored in separate

lists providing us with instant access, since we know that vertex with ID i can be found at index i. We use $List$ data structure to store the relevant data because they provide instant insertion and access. Although data structures like $HashSet$ and $LinkedHashSet$ come up supplemental with an instant removal of entries, omitting index shift to the left of subsequent elements, those required much more space. After running simple benchmark tests, where we measured the average memory consumption per entry for diverse data structures, we came up with the conclusion that for our needs simply array based data structure would be sufficient. The reason for that being is that other data structures require 2,33 or even up to 2,67 more memory to store the same amount of data, i.e. the desired degrees, clustering coefficient and node IDs.

Having created n vertices and assigned to each of them a target degree and clustering coefficient, the algorithm consequently distributes each of the them inside affinity blocks matching their expected number of triangles. In the next step incomplete buckets are being iteratively merged and stored inside the set of complete groups preparing all the data required to begin the execution of intra connection. Implementation details closely connected to this part can be found below.

5.3.2 Vertex Distribution in Groups and Merge of Incomplete Buckets

After assigning a unique ID, target degree and local clustering coefficient for all nodes using the standard approach or any growth prediction model, Darwini distributes vertices inside buckets and if necessary merges incomplete ones. Currently the assignment of vertices to groups and merging of incomplete buckets is executed by a single thread. The lack of parallelization is due to the fact that this thesis targets to investigate how accurately Darwini is capable of reproducing properties of networks, especially real-world graphs. Based on the target degree and clustering coefficient each vertex gets assigned to a bucket. Each bucket is stored as a pair of the bucket's ID and the subset of vertices belonging to it. For each bucket we also store the minimum degree and the expected number of triangles inside two additional lists. Therefore, the ID of a bucket does not correspond to the ID of the first vertex assigned to it. This enables us to easily retrieve the necessary number of closed triplets when computing the edge probability for a given bucket. Or, during search of a non-full bucket to instantly access the minimum degree for a certain group, which defines the upper bound of elements that can be placed inside the bucket. Further, we distinguish between full and incomplete buckets by using the following approach: During initialization we place the bucket inside a list with incomplete buckets. Its minimum degree and the required number of triangles are also stored in such sets. Once a bucket can not add further vertices, therefore being 'labeled' as full, the bucket and all of reference values are removed from the incomplete set and attached to the complete one. The whole process is done to simultaneously decrease the amount of time required for finding an apposite block for each vertex and for merging incomplete buckets into a larger one. Because, in worst case scenario, when adding vertices to groups, we need to search through the incomplete set of buckets whereas during merging we iteratively retrieve incomplete affinity blocks and combine them into a new one until the newly formed group consisting of multiple incomplete buckets reaches certain size. Using the above depicted method, we have significantly reduced the runtime of phase 1 from hours for graphs with few million of vertices to just a couple of seconds. Now that we have acquired a set of full buckets, the algorithm can proceed with the edge insertion within communities.

5.3.3 Connecting Vertices within Buckets

The second half of stage two covers the random edge creation between nodes within a bucket's domain. To speed-up the intraconnection process, we retrieve the number of available processors. Next, we assign to each processor approximately equal amount of unique buckets. Those should be processed by creating the pre-computed amount of edges inside each block. During execution, we must guarantee that the degrees of vertices are accurately updated by prohibiting a concurrent update of a node's degree from two or more threads or vertices sharing the same sublist , limited to 50 nodes per row pointer ($rowPtr$). Also, we must ensure that by adding an edge from v_i to v_j the degrees of both endpoints are updated properly. This is achieved by calling an *addEdge* method which monitors which sublist of the custom *Compressed Sparse Row* data structure each process wants to access. If we assume that two threads try to obtain a node from the same sublist, then either of the two would gain the lock hence sending the other process in waiting/sleep state. After the thread, occupying the sublist index, has either inserted a new edge or confirmed the existence of an edge in the graph, it releases the sublist index. This way it gives the opportunity to other waiting processes to procure entry to the same sublist. Although this reduces the CPU utilization, in the absence of this synchronized block, the algorithm will return inaccurate results. However, we have found empirically that due to the assignment of a vertex to a single group it is unlikely that two or more threads will want to gain access to the same sublist. Therefore, during tests the CPU utilization in the second stage did fall under 81-83% for negligible periods of time. Exactly this high usage of cores leads automatically to fast build of (small) communities.

So, after establishing distinct communities, Darwini has to introduce connections outside each neighbourhood to form the final connected graph. Information about how exactly pair of vertices from independent groups are chosen to be connected with one another is depicted in the approaching subsection.

5.3.4 Connection of Vertices outside Buckets

Unlike in stage 2, here each thread searches for every vertex v_i, having fewer than expected edges, a candidate v_j by choosing at random from the range $(0, n-1)$, i.e. Darwini now allows to choose nodes from the entire graph. If a candidate not matching its target degree has been found and $v_i \neq v_j, (v_i, v_j) \notin E$ holds, then we add the edge using the *addEdge* method described before. If we take a look at the for-loop on line 7 - Algorithm 4 in [ELW+16] we could ask ourselves: What will happen if we assign a subset of nodes to each thread? Should this result in faster execution time? While assigning a smaller subset would definitely speed-up the algorithm since each thread has to execute this part multiple times on the other side this would lead to inaccurate results. After adjusting Darwini this way we figured out that in general the algorithm produces poor results regarding the overall structure and characteristics of the original network. The consequences from this change derive from the fact that now each high degree node is being processed by a single core and eventually increments its degree. And because Darwini was designed to terminate either by reaching the target size or by reaching a maximum number of iterations [ELW+16], the high degree vertices could not be generated. Even though high target degree vertices are shuffled inside groups, it is still not sufficient for meeting their target degree. Having clarified that the work done in the first for-loop in the third stage can not be 'split' we now turn our attention to the second part of the interconnection process.

The second half of Algorithm 4 in [ELW+16] is also responsible for finding connections for high degree vertices. The implementation consists of a method filtering all non-matching target degree vertices and randomly assigning them to buckets of fixed size. Inside those groups every pair of vertices is a possible candidate. Recall that the size of each group grows exponentially thus increasing the search space and the amount of possible new edges. Our decision was to let each thread execute the shuffle method and form its own buckets. Of course this results in trade-off between memory efficiency and execution time. If we assign the same set of buckets to each thread, then the algorithm would consume significantly less space. Otherwise, supplementary $(\#cores - 1) * (8B * \sum_{b \in B} b.bucketSize)$ would be required. On the other side, distinct buckets for each thread would outcome in more connection possibilities for high degree nodes. Further, the CPU utilization would be higher because no barrier before and after the shuffle method would be necessary. We have found empirically that threads do not reach the shuffle method at the same time due to the $addEdge$ method sending threads in waiting state when trying to gain access to occupied sublist index. So, we decided to allow each processor to randomly distribute non-matching degree vertices into groups since the amount of buckets decreases with every next iteration, i.e. less likely that memory fragments would advent. Also, one can see a decrease in the number of vertices that need to be randomly spread since low degree nodes quickly match their target degree. Specifically for social networks those represent huge part of the participating members.

We have seen that Darwini can be parallelized on a single machine or executed in a cluster from multiple machines [ELW+16].The algorithm also produces dense communities found in real-world networks by approximating the expected number of closed triplets for each vertex. Then it connects two or more communities by forming a larger one while simultaneously fixing the target degree and local clustering coefficient of individual vertices. Vital property exhibit from Darwini is the degree-correlation common for online social networks and the capability of reproducing high degree nodes, which e.g. are absent in the BTER model [KPPS14]. Although Darwini combines several positive feature, there are couple of drawbacks that will be discussed in Chapter 7, which contradict the assumptions made in this chapter.

6 Evaluation

In this section, we evaluate the characteristics of Darwini. We do this by testing the original algorithm and our proposed growth prediction models against several metrics, and conclude to what extent each of them is able to preserve the properties of real-world datasets. Direct comparison of the original method against our growth prediction approaches is also provided. Datasets from individual domains were used in this study to measure the generation accuracy of Darwini for social networks and further important domains like autonomous systems, road networks, email networks, entertainment platforms with ground-truth communities and citation networks. Before we focus on the results, we would like to first provide information about every utilized dataset.

6.1 Public Available Data

In this section, we briefly introduce each network used to test the capabilities of the synthetic graph generator Darwini and our approaches in maintaining real-world graph properties. Each publicly available dataset we used in our evaluation belongs to the set of *small data* in case both the order and size of the original network do not exceed 1 million. In contrast, if either set V or E exceeds the 1 million upper bound, we simply categorise the given graph as *large data*. For every graph G we should also note that $m >> n$, i.e. the cardinality of E is relatively higher than the cardinality of V, so edges tend to grow super-linearly compared to the order of the graph. We also provide statistics from SNAP [LK14] for each data set while covering possible tasks (if available), that each data can be used for including link prediction, community detection and binary classification.

6.1.1 Small Data

GEMSEC [RDSS19] is the first dataset we chose to test Darwini on since it contains networks for different verified Facebook pages. Grouped into categories such as athletes, artists, TV-shows, politician, pages are represented by nodes and connection between two pages corresponds to mutual like among them. The smallest network is composed of only 3,982 vertices and 17,262 links while the largest one has 50,515 nodes and well over 819,306 edges.

Next, we picked two citation datasets from the e-print arXiv. *High Energy Physics-Theory* and *Astro Physics* [LKF07] are scientific networks covering collaborations between authors of papers submitted to the High Energy Physics - Theory and Astro Physcis categories. Both collections of papers are derived from a 10 year period from 1993 to 2003. The set of vertices constitutes to authors and links are present for co-authoring at least one paper. Distinctive marks are the high average clustering coefficient 47% and 63% and low effective diameters 7,4 and 5, respectively.

We move forward to an email communication network *Enron* [LLDM09] formed by user exchange of electronic messages within a dataset of nearly half million emails. User connections denote that at least one message was sent from address i to address j or vice versa. The communication of non-Enron email users is also taken into account. High average clustering coefficient 49,7%, over 90% of the nodes and edges are found within the largest strong connected component and thus evincing low effective diameter - 4,8.

Further, we observe *Twitch* from MUSAE [RAS19], an American video service that focuses on video game live streaming. More precisely, we observe *MUSAE* dataset: user-user networks of gamers who broadcast in a specific language. Total of six European countries, among those being Germany, England and Russia are found within the dataset where links represent mutual friendship. MUSAE is suitable for community detection, prediction of language in which certain user streams know as binary node prediction. Details about each network are depicted in Table 6.1.

AS-733 [LK14] is a set of subgraphs constituting of routers compromising the Internet organized in groups forming *Autonomous Systems (AS)*. AS exchange traffic flow with neighbours in a communication network of who-talks-to-whom constructed using the *Border Gateway Protocol (BGP)* logs. AS-733 dataset holds over 700 daily graph instances. We have randomly picked two out of three graphs (largest AS being the last one) to test Darwini and our extension models on.

Despite the fact that the above outlined datasets cover variety of domains, they either only represent a small portion of the whole network or their size is significantly smaller compared to larger datasets such as the social network of Facebook. Besides that domains like road infrastructure networks have not been considered yet. The size of many real-world networks is magnitudes larger than couple of thousand vertices and edges. Therefore, in the coming subsection we outline several large datasets used in our evaluation.

6.1.2 Large Data

Created in the early 90's *DBLP* [YL15] is a computer science bibliography containing broad range of computer science papers. Representing a co-authorship network where two authors are connected only if they have worked on the same paper. Ground-truth communities are formed by researchers publishing their work to a given conference or a journal. Key feature of this network is the correspondence of the strongly connected component to the entire graph. Consequently, DBPL exhibits high average clustering coefficient 63% and relatively low effective diameter - 8 can be noticed.

YouTube is arguably one of the most popular video-sharing web sites on the internet, where users can watch numerous videos from arbitrary domains. In addition to that, YouTube also has a social network with users forming friendships with others or participating in different groups. It is a strongly connected network, having average clustering coefficient of around 81% and with the largest strongly connected component equal to G. Hence a low effective diameter of 6,5 hops on average can be perceived.

The road infrastructure network of California [LLDM09] represented by either intersections, of two or more roads, or road endpoints and roads connecting the previous. Almost every road and intersection can be found within the largest strongly connected component. California's road

network is the first dataset exhibiting extremely long diameter 849 and one of the lowest average clustering coefficients 4,64%. This is somehow expected since roads more often intersect themself inside cities and rarely form triangles.

Next, we present the *Autonomous systems by Skitter* [LKF07], an internet topology graph collected from traceroutes, where routers are connected to sinks resulting in numerous links. Apart from a negligible amount of nodes, a path from one router to any other can be found. Average clustering coefficient of 25% and four time smaller effective diameter compared to the longest geodesic path hints that the graph contains dense and sparse communities.

LiveJournal is an online-blogging community allowing users to form friendships and join groups which are considered to be ground-truth communities. The network consists of millions of users and multiple times more connections. Every user in LiveJournal is part of the largest strongly connected component. Interesting fact about the social blogging network can be illustrated in the low clustering coefficient - only 28%.

And lastly we study the structure of the web graph *Wikipedia* of top categories. The final version was constructed by taking the largest connected component of Wikipedia and restricting to pages in the top set of categories and taking the largest strongly connected component of the restricted graph. This resulted in an average clustering coefficient of 27,5% with measured diameter 9 and effective diameter of just 3,8 hops on average.

After delineating previously measured characteristics of each dataset we will use in our evaluation, we move further to the next section where we test Darwini and our growth prediction models against several key characteristics including higher metrics such as PageRank and k-Core.

6.2 Results

In this section we delineate the outcome for a variety of metrics measured on real-world networks. Instead of summarising the set of data we collected for each metric, we will begin the discussion for each dataset with the degree distribution and further try to stepwise give a possible explanation of possible aspects of Darwini that could have overall affected the results for each remaining metric in certain direction. A subset of the measured metrics can also be found in Table 6.1 for each observed network. Besides the expound of results we would like to point out that only one visual representation per dataset in the form of a diagram is given. This way we attempt to facilitate the reader while going through the large set of results while also aiming at presenting them as compact as possible. We will use the abbreviations *RID* and *LPA* as a reference to generating multiple graph with Darwini and either *Random Increase of Degree* or *Linear Preferential Attachment*. In addition to that we have omitted many entries for each curve in order to make the results as clear as possible. We will further provide information for distributions where we have cut off entries above certain value for both x-axis and y-axis. This is normally the case when either very few values can be found within huge span and when only entries for the original graph are present.

We start with degree distribution outcome and mainly focus ourselves on it and clustering coefficient distribution because they are often used to distinguish between arbitrary random graphs and real-world networks. First, we will résumé the data acquired from testing all Facebook pages. In Figure 6.1 one could visualize part of the results for the Facebook pages consisting of athletes. As observing the degree distribution curve, we would like to point out that the original graph has

Dataset	Graph Name	Order	Size	Density	Largest Degree	Average Degree	Connected Components
GEMSEC	Athletes	13.8K	86.8K	0,09%	468	12,52	2340
GEMSEC	Artists	50.5K	819K	0,064%	1469	32,43	5752
GEMSEC	Company	14.1K	52.3K	0,052%	215	7,43	3177
GEMSEC	Government	7K	89.4K	0,36%	697	25,35	793
GEMSEC	New Sites	27.9K	206K	0,053%	678	14,77	4248
GEMSEC	Politician	5.9K	41.7K	0,239%	323	14,12	1016
GEMSEC	Public Figures	11.5K	67.1K	0,1%	326	11,6	2357
GEMSEC	TV Shows	3.8K	17.2K	0,23%	126	8,87	831
arXiv	CA-HepTh	9.8K	26K	0,05%	65	5,26	816
arXiv	CA-AstroPh	18.7K	198K	0,011%	504	21,1	562
SNAP	DBLP	317K	1M	0,002%	343	6,62	12706
SNAP	Email-Enron	36.6K	183K	0,027%	1383	10,22	1092
Musae	DE	9.5K	153K	0,34%	4259	32,24	1188
Musae	ENGB	7.1K	35.3K	0,01%	720	9,91	1495
Musae	FR	6.5K	112K	0,05%	2040	34,4	666
Musae	ES	4.6K	59.3K	0,055%	1022	25,55	611
Musae	PT	1.9K	31,3K	0,17%	767	32,74	176
Musae	RS	4.3K	37.3K	0,04%	1229	17,01	730
AS-733	as19971108	3K	5.5K	0,121%	592	3,67	331
AS-733	as19971215	3.1K	6K	0,121%	634	3,83	343
AS-733	as20000102	6.4K	13.8K	0,066%	1460	4,92	575
SNAP	roadNet-California	1.9M	2.7M	0,0001%	12	2,81	70366
SNAP	com-youtube	1.1M	2.9M	0,0004%	28754	5,16	23866
SNAP	com-livejournal	4M	34.6M	0,0004%	14815	17,34	1560
SNAP	as-skitter	1.7M	11.1M	0,771%	35455	13,08	756
SNAP	wiki-topcats	1.8M	28.5M	0,0018%	238607	31,83	1

Table 6.1: Graph datasets used in the evaluation of Darwini and the growth prediction models. Several important graph characteristics including the density and the unique number of connected components are depicted in the table above.

vertices with degree over 300. But because neither Darwini nor our growth extension models were able to generate those high degree vertices, we restrict the upper bound of the x-axis to 300 for better legibility. Although all generation models are able to match the degree distribution curve quite accurately (excluding omitted region), the lack of high degree nodes influences the overall structure and the remaining metric measurements. While 468 being the largest degree of the original graph, Darwini, RID and LPA we incapable of generating nodes with degree higher than 269, which is a substantial difference. Similarly, in each of the remaining networks within the dataset tested, we could also see absence of high degree nodes. One exception being the artists network, where LPA and RID managed to match the high end of the original curve. In all cases LPA comes closest followed by RID and Darwini to reproducing high degree vertices of the original graph. However,

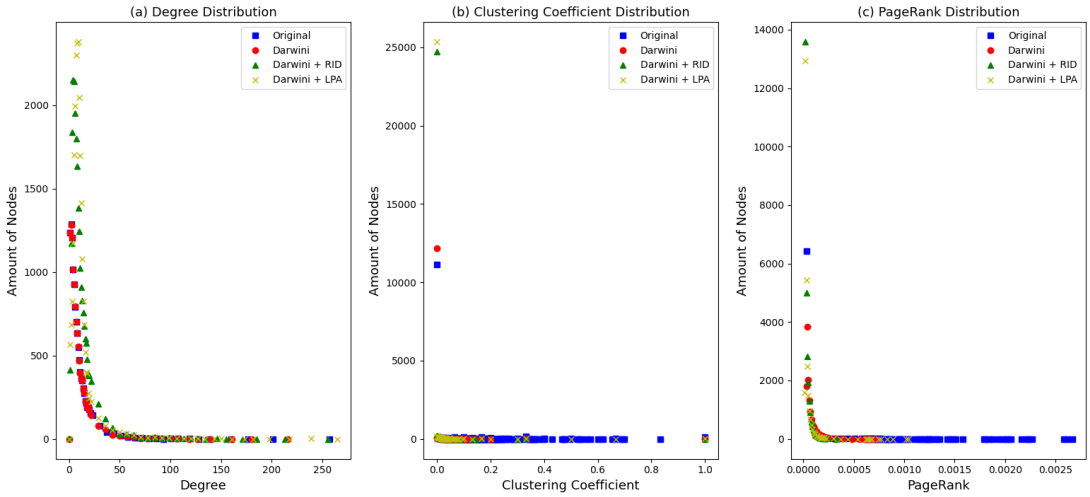

Figure 6.1: Comparing Darwini and both growth prediction models under different graph metrics on the Facebook athletes pages network. Every model struggles to accurately reproduce distributions.

this is somehow expected since LPA and RID have twice as more vertices and at least two times more edges. And further considering the fact that LPA connects new joining vertices to already well linked nodes with higher probability. Clearly noticeable in Figure 6.1 (b) is the inaptitude of every model to reproduce the clustering coefficients in the [0,2;0,99] region. This is congruous across the whole dataset. A possible explanation can be seen in missing high degree nodes for Darwini, RID and LPA. The way nodes have been distributed across buckets, i.e. the formation of communities, can also influence the local clustering coefficient. Despite the fact that consistently each Facebook network has greater amount of connected components than its corresponding synthetic graphs, Darwini in general can not cope with closing triangles even in dense communities. Darwini also exhibits relatively shorter diameter and radius then the original network, which also hints about wrong approximation of the underlying structure. RID and LPA unsurprisingly evince decrease in the longest and geodesic paths proven to be a common process in real-world networks [LKF05]. The dip in the PageRank distribution observed in [ELW+16], can also be perceived in Figure 6.1 (c) with vertex to single group mapping considered as the factor causing this inaccuracy. The structure of Darwini along with the missing high degrees lead to false estimation of the curve. By its natural way of linking, LPA tries to correct the results of Darwini but reproduces overall negligible amount of PageRank scores. In terms of k-Core, Darwini produces in most cases more distinct cores with degree k. By RID and LPA major part of the k-cores are in the [5;20] region for k and only remarkably fewer outside the specified region.

Next, we will be looking at the MUSAE dataset [RAS19], covering networks of Twitch streamers from different European countries. Digital representation in the form of a diagram is provided for the network of German streamers, appearing to also be the biggest one in the dataset. Details about the single networks can be found in Table 6.1. In Figure 6.2 comparisons of Darwini, RID and LPA under degree, clustering coefficient and PageRank distributions are depicted. The x-axis of the degree and PageRank diagrams were restricted to values up to 800 and 0,01 to provide better clearness. As in the previous dataset, in Figure 6.2 (a) we can see that none generation approach is able to generate high degree vertices, which for the original graph reach degree 4259,

i.e. approximately from 6 to 8 times larger than the largest degrees produced by Darwini, RID and LPA. Surprisingly, the synthetic graphs are far way better connected compared to the original despite the lack of high degree nodes. Consequently resulting in times fewer amount of connected components for all networks in MUSAE, excluding the United Kingdom graph. Therefore, we would assume that Darwini is not suitable for low dense real-world networks leading to build up of an inaccurate structure. However, our assumption is in contradiction with the measured diameter and radius of the individual graphs and the clustering coefficient distribution results, which are partially not consistent. For example, the diameter and radius measured from Darwini for the German network match those of the original graph while it is not the case for the rest. On the other side as expected, RID and LPA exhibit lower radius and diameter compared to both Darwini and original graph across the dataset. Turning our attention to Figure 6.2 (b) one can spot better results compared to those in Figure 6.1 (b) although more high degree vertices are missing in the German graph of streamers than in the Facebook pages of athletes network. Almost indistinguishable are the remaining local clustering coefficient results, apart from those of the United Kingdom where one can find few missing values in [0,3;0,4] region. Excluding the cut off region [0,0125;0,025] where we have a few PageRank entries for the original graph, each generation model follows more accurately the original compared to the Facebook dataset. Several unmatched values in the middle of the curve can also be found in the rest of the dataset. Consistent with the measurements for the previous dataset is the dip at the beginning of the curve, which seems to be a general issue for all types of graphs generated using Darwini. If it was capable of reproducing the degree distribution more precisely, then we would have probably retrieved even better results for MUSAE. Even then we presume that Darwini would not match ideally the PageRank distribution due to the fact that the algorithm tends to be designed in generating graphs with dense but non-hierarchical communities. And at the end we would also like to briefly describe the results for k-Core. Because Darwini attempts to create highly dense communities, the k-Core distribution of the original graph contrasts from those of Darwini, RID and LPA. Covering the whole dataset we can conclude that Darwini, RID and LPA posses cores with a value of k over two times than the largest core that can be found in the k-Core distribution of the original graph. One more thing worth pointing out is the fact that the sum of all cores in the region from k to $2 * k$ for Darwini, RID and LPA is almost identical to the amount of cores with value k for any Twitch network.

Moving on we represent in Figure 6.3 and Figure 6.4 outcomes for CA-AstroPh and CA-HepTh collaboration networks. Turning our attention to the distribution curves we can notice that in case of CA-AstroPh, Darwini and RID have missed on replicating several high degree entries unlike LPA where one can not see a couple of entries spread across the [500;597] region, taken out for better clarity. For Ca-HepTh, every model matches the original curve precisely and in addition to that LPA again had generated some higher degree nodes. Of course, RID and LPA do not necessary overlap the degree distribution results of the original graph which makes sense since they both have twice as much vertices. Here we say that they match the original curve in the way that the amount of nodes drastically decreases in the [0;10] region. At this point we would also like to provide a possible explanation to why the degree distribution for CA-HepTh is so well matched. If we take a deeper look at Figure 6.4 (a) we see that there are almost no holes in the degree distribution sequence. A hole in a sequence of positive numbers $seq(k, n) = k, k + 1, ..., n - 1, n$ with $k < n$ occurs when in $seq(k, n)$ after any number l with $k \leq l < n$ the following element does not equal $l + 1$. Up to this point one would probably ask himself: What do these holes in the original degree distribution sequence have to do with the results provided from Darwini? Because Darwini tries to maintain the degree correlation of graphs, it uses a probability formula discussed in subsection

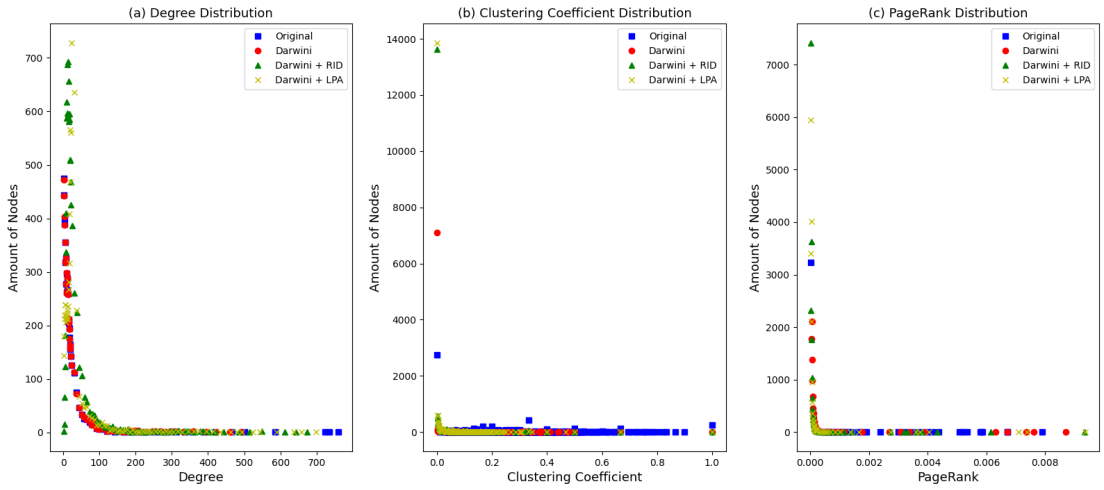

Figure 6.2: Comparing Darwini and both growth prediction models under different graph metrics on the Twitch network of German streamers.

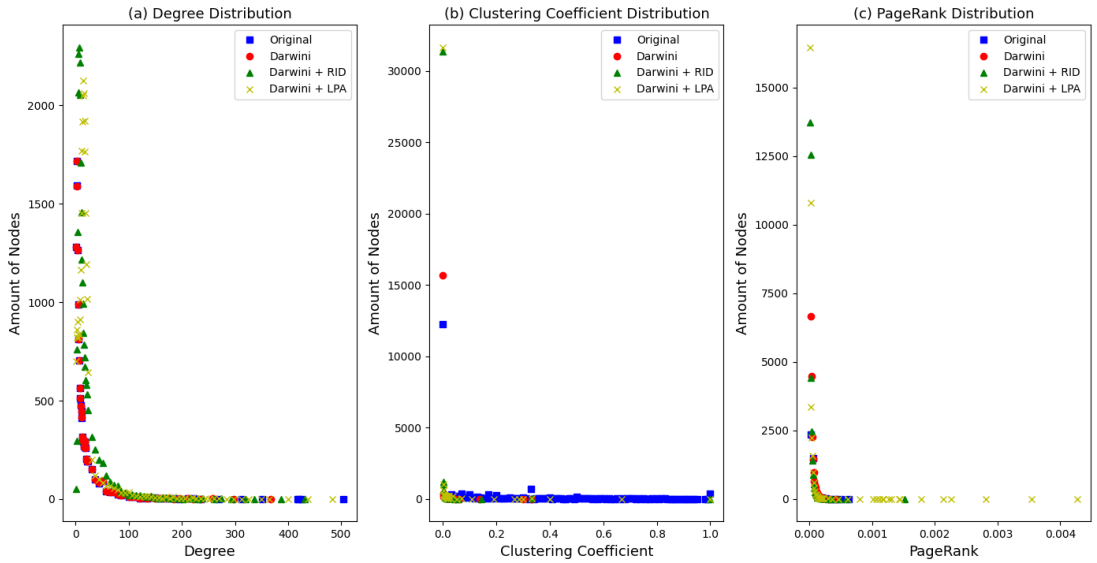

Figure 6.3: Comparing Darwin and both prediction models under different graph metrics on CA-AstroPh network. Only LPA manages to create more high degree vertices than the original graph.

5.1.3, making it highly unlikely that a pair of vertices with significant difference in their degrees will ever be connected. So, if the original degree distribution contains high degree vertices with large holes present in between, then the algorithm would be unable to find enough candidates to link them to. Recall that majority of vertices in real-world networks posses low degree and their corresponding target degree is either already matched in the second phase or in worst case in the early iterations of phase 3. In addition to that, multiple links and self-loops are permitted. However, this may not be the only reason leading to faulty results. After removing the probability formula in the second half of stage three, we witnessed smaller improvements although most high degree vertices were still missing. So along the previously mentioned holes in the degree distribution, possible aspects for this behaviour could be the way vertices are grouped together in the second stage, density of the communities and the hierarchical structure of the original graph. As we will later see, although the holes in the degree distribution may not be the only aspect for wrongly estimated degree distribution, their absence leads to significantly finer results. The local clustering coefficient results retrieved for both networks serve as a perfect example that Darwini's inability to preserve graph characteristics could possible be mixture of several of the above suggested factors. Although all models were able to generate accurate degree distributions for CA-HepTh, none of the models is capable of reproducing the local clustering properties of the original graph, as it can be clearly seen in Figure 6.3 (b) and unsurprisingly in Figure 6.4 (b) for CA-AstroPh. Even though, Darwini maintains the degree distribution, the way that it groups vertices perhaps could have heavily influenced the local clustering properties of the output graphs. Ignoring the beginning of the PageRank curve, where we could see the usual dip, results are much better compared to those of GEMSEC and in some extent similar to those in Figure 6.2 for both collaboration networks. Due to the double amount of vertices possessed by the synthetic graphs generated with RID and LPA, couple of entries extending their corresponding curves can be observed in Figure 6.3. This leaves us yet again without a feasible explanation about the factor(s) influencing the overall structure and as a consequence the measured properties of the final graph. Finally, we would like to note that both collaboration networks gain distinction as the k-Core distributions measured for each model are unique. However, the original graph contains cores with value for k up to a magnitude larger. The fact that Darwini, RID and LPA have generated networks with more connected components, thus a less connected graph, and the bad estimation of the clustering coefficient could be considered as solid arguments leading to such inaccurate k-Core results.

In Figure 6.5 we depict the measured distributions for the Enron email network. Like in the other cases, Darwini lacks the ability of reproducing the high degree vertices, where its node with largest degree 526 is more than two times lower than the one of the original graph - 1383. RID and LPA produce vertices with higher degree than Darwini but none in the omitted on the x-axis in Figure 6.5 (a) [850;1400] region. As a consequence, none generation model produces graph with the desired connectivity and in combination with the possible wrong distribution of nodes resulting in poor estimation of the clustering coefficient curve. As it can be discerned in Figure 6.5 (b), Darwini is the only model making attempts to match somehow the original results. Possible explanation can be the random increase of degree for an arbitrary pair of vertices for RID and that new joining vertices are mainly linked to existing high degree nodes for LPA following a totally different distribution across buckets due to the increased target degree of some subset of nodes. In addition, we should point out that the second half of vertices in RID and LPA have been assigned a target clustering coefficient of node from the first half. Each vertex's target score remains unaffected even in case a higher target degree gets assigned to it. Observing Figure 6.5 (c), provides the biggest surprise in terms of how well Darwini captures the PageRank distribution. However, we would

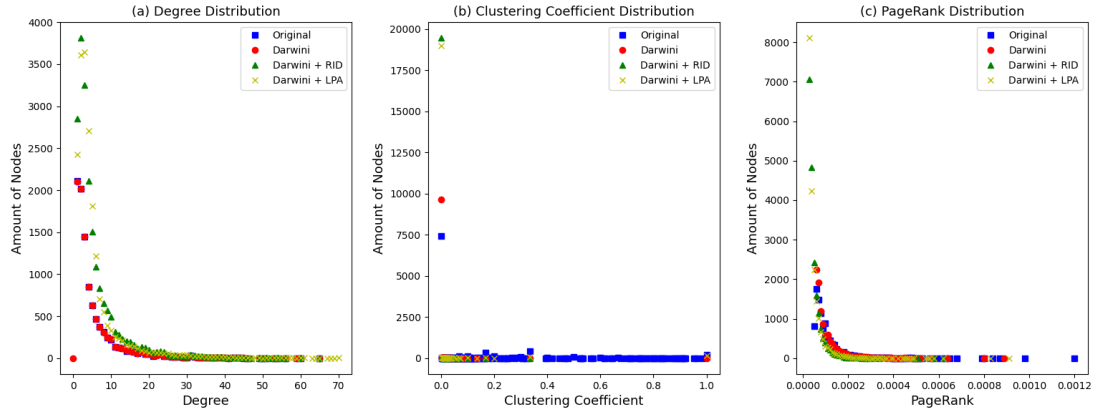

Figure 6.4: Comparing Darwini and both growth prediction models under different graph metrics on CA-HepTh network. All models approximate the original graph PageRank distribution curve.

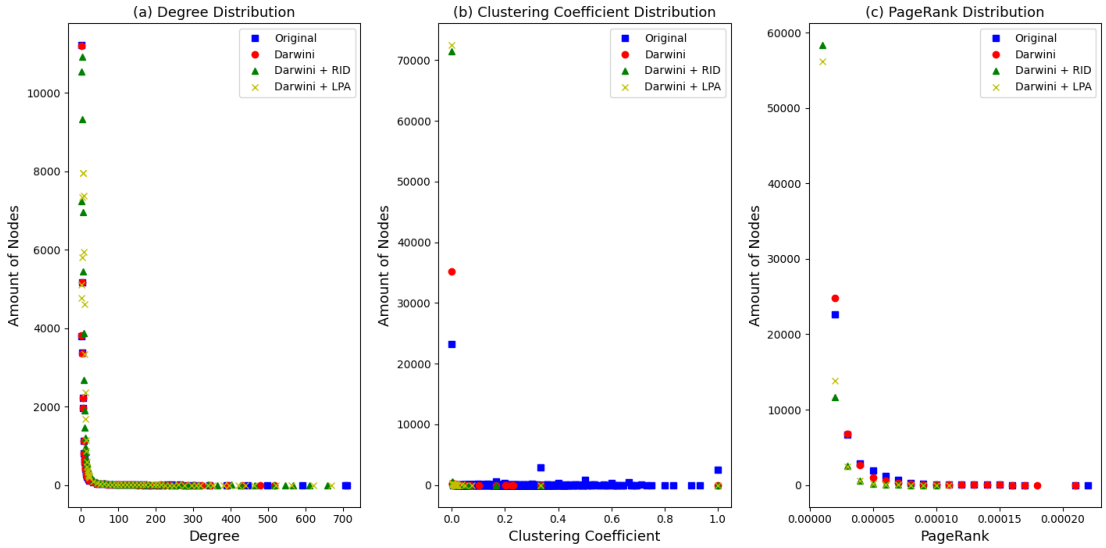

Figure 6.5: Comparing Darwini and both growth prediction models under different graph metrics on Enron email network. Darwini nearly reproduces the PageRank distribution curve of Enron.

presume that this is no coincidence derived not from the very few entries available in the series rather due to the relatively low span from the lowest to the highest PageRank value, which can also be observed in other networks. Regarding the k-Core results, we can conclude that only Darwini manages to partially match the k-Core of the original graph but produces several larger k-Cores not present in the last. RID and LPA provide very distinctive results but unlike Darwini form only few larger cores.

Following we turn our attention to the AS-733 dataset. More precisely we cover three daily instances from different years and hence expecting with higher probability noticeable changes due to the longer time span. In Figure 6.6 one can depict measurements about the largest instance

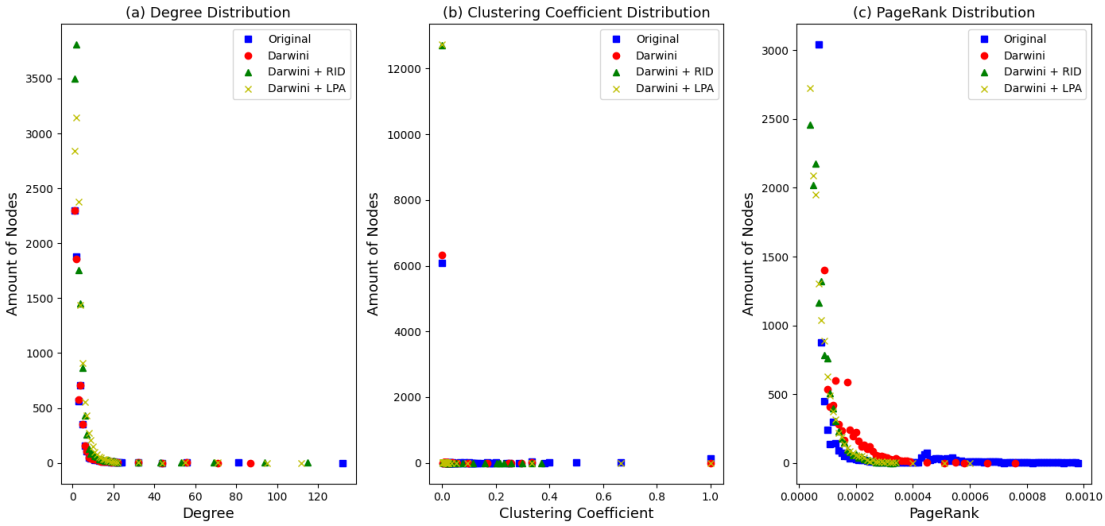

Figure 6.6: Comparing Darwini and both growth prediction models under different graph metrics on AS-733 from 2 February, 2000. RID is able to replicate most clustering coefficient values.

across AS-733. Even after restricting the x-axis of the degree distribution diagram, one can still pretty clearly spot the missing high degree vertices in each synthetic graph, which also holds for the remaining measured daily instances. From this point, we can straightforward conclude the rest of the data collected. Since there are many missing key edges that would further interconnect several communities, on the one side output graphs exhibit longer diameter and on the other side they can be described as sparser compared to original daily instances. Because the size and order are remarkably fewer than majority of real-world networks, each missing link influences heavily the structure. When taking into consideration the above discussion, one could possibly think that no model comes close in estimating the clustering coefficient curve. However, as it can be seen in Figure 6.6 (b), each model matches the original curve far better than expected, even when the so called bridge vertices are absent in each generated graph. There is a simple explanation behind this rather interesting and unexpected result. As elucidated in subsection 5.1.2, Darwini tries to first match the expected number of triangles each vertex should belong to while indirectly striving to simultaneously reach the target degree and clustering coefficient. In the final phase, the algorithm will in worst case produce an insignificant amount of triangles when picking random candidates, as stated in [ELW+16]. Therefore, the final clustering distribution found in Figure 6.6 (b) was mostly formed in second stage, which suggests that Darwini has done a good job the way it has distributed vertices across groups. The erroneous PageRank and k-Core distribution ensued by missing key connections, i.e. vertices serving as bridges, and are congruous across each covered network.

As next, we will take a look at the results for another network part of the autonomous system domain by the name Skitter. First and foremost, no results can be spot in Figure 6.7 for LPA because the algorithm did not terminate in reasonable time. Exactly the same can be said for the PageRank and k-Core distributions, radius and diameter. An upper bound of 50K for the y-axis was also set and thus omitting few entries in the [50K,400K] region matched by Darwini. Although small amount of data was collected for this network, it would still be sufficient to confirm to some degree our assumptions formed on previous datasets. We start yet again with the degree distribution

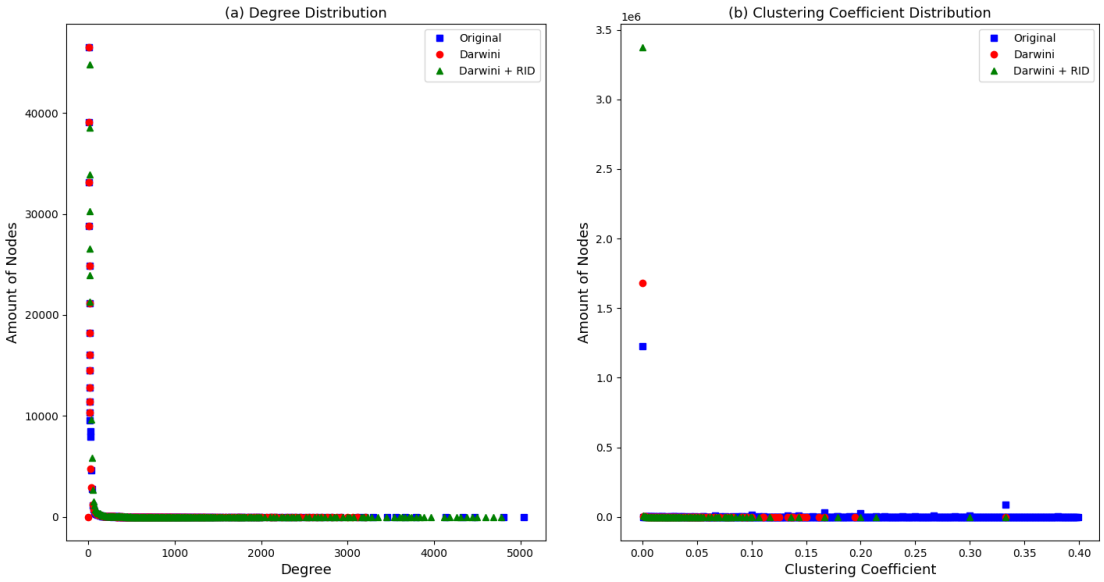

Figure 6.7: Comparing Darwini and RID models under different graph metrics on AS-Skitter. Major part of the high degree vertices can not be reproduced leading to inaccurate graph structure.

in Figure 6.7 (a) where Darwini and RID estimate the curve up to some point - 3280 and 4895, respectively. From then on however, both models miss on reproducing considerable amount of nodes with degree up to 35455, not shown in the diagram in order to improve the legibility. With that said, Darwini in general has over 800K edges less than the original graph and as a consequence the number of connected components increases over 100 times from 756 to over 80K for the previous. Same conclusion can be made for the synthetic graph of RID, where a further increase, raising well over 110K, in the amount of connected components can be observed. Overall, both output graphs do not even come close to the original structure of AS-Skitter due to the numerous holes found in the degree distribution sequence, previously raised during explication in the discernible differences in the degree distribution of collaboration networks CA-AstroPh and CA-HepTh. And as a consequence the local clustering properties, illustrated in Figure 6.7 (b), of the original graph can not be reproduced by any means, where only minor part of the AS-Skitter curve is replicated by both approaches.

Further measured network, is the DBLP graph found in the evaluation section of [ELW+16]. We were not able to collect data about the PageRank and k-Core distributions but provide data about the synthetic graph generated by LPA. Darwini follows closely the original degree distribution until the gap between two neighbouring entries in the series becomes considerable, while RID and LPA are able to reproduce the missing entries in the Darwini curve mainly due to the double amount of nodes available and relatively low highest degree in DBLP - 343. At this point we would like to point out that it seems that very few entries in Figure 6.8 (a) were unmatched. Actually this is a result of filtering the major part of the entries for each model, moreover setting an upper bound for the y-axis. Otherwise, one would not be able to perceive much if we plot every single entry for larger datasets like DBLP. These results further confirm our presumption that Darwini is actually not applicable to graphs with huge holes in their degree distribution rather than being suitable only for dense online

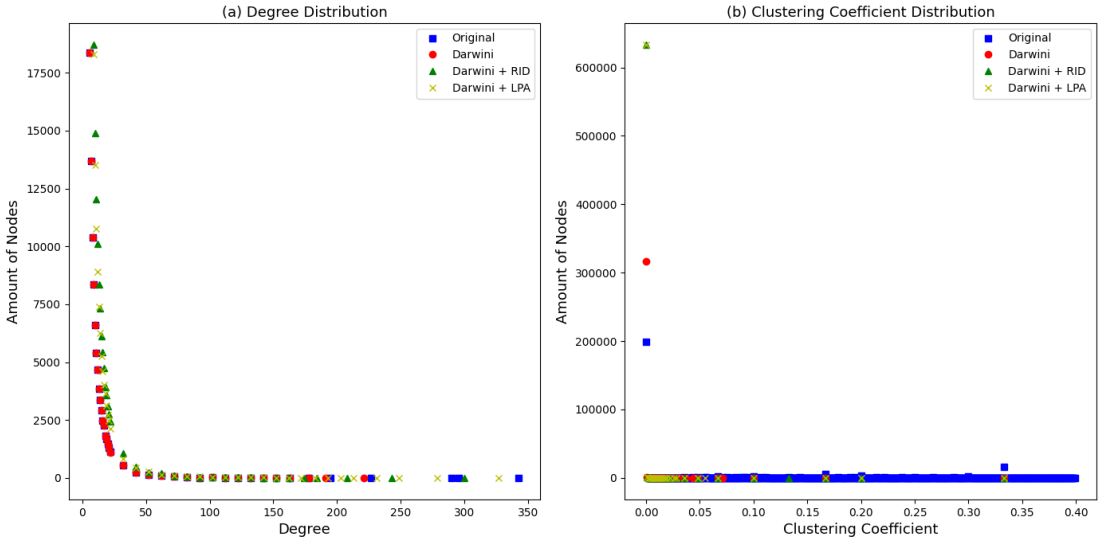

Figure 6.8: Comparing Darwini and both growth prediction models under different graph metrics on DBLP. Darwini captures low and medium region of degrees while LPA also reproduces the high once.

social networks. If we ignore the very few missing high degree entries in the degree distribution, we can conclude that the algorithm produces synthetic graphs with diverging from DBLP's structure expressed in 12706 against over 23K, 40K and 39K connected components for Darwini, RID and LPA. And in addition to that, unlike in the AS-Skitter network, there are almost no missing edges that need to be inserted. The worst estimation of the local clustering coefficient can be noticed in Figure 6.8 (b), where only a small fraction of all entries have been matched. Another great example of good estimation from Darwini and RID in the absence of holes in the series of degrees present in a graph is the road network of California. From the visualized results in Figure 6.9 we can conjecture that even a perfect estimation of the degree distribution does not correlate with a good match of the clustering properties. As already discussed and also pointed out in [ELW+16], Darwini does not observe hierarchical structure and creates communities by assigning a vertex to only one group. Interesting contrast in the amount of connected components between Darwini and RID can be derived from the results - 323K against only 65K which corresponds to the assumption of densification as the graph evolves and grows.

We end up the discussion of our results with the Wikipedia top categories, YouTube and LiveJournal networks. As foreseeable, each network is unable to reproduce the set of high degree vertices. For the YouTube graph, we have restricted the x-axis in Figure 6.10 (a) to 1700 instead of 29 thousand for better readability and cut off values of the y-axis above 10 thousand, where we had few entries matched by Darwini. Likewise, the diagram in Figure 6.11 (a) omits values for the x-axis and y-axis above 5500 and 5000, while for LiveJournal no entries on the x-axis other than those of the original graph can be observed in the [4000;15000] region and y-axis restricted to 5000 for overall better clarity. Up to this point after analysing various small and large datasets from distinct domains, we can conclude Darwini's lack of ability in duplicating the degree properties of original graphs in the presence of holes in the degree distribution series. Impacted from this, the output graph holds numerous small connected components, which by no means can be consistent

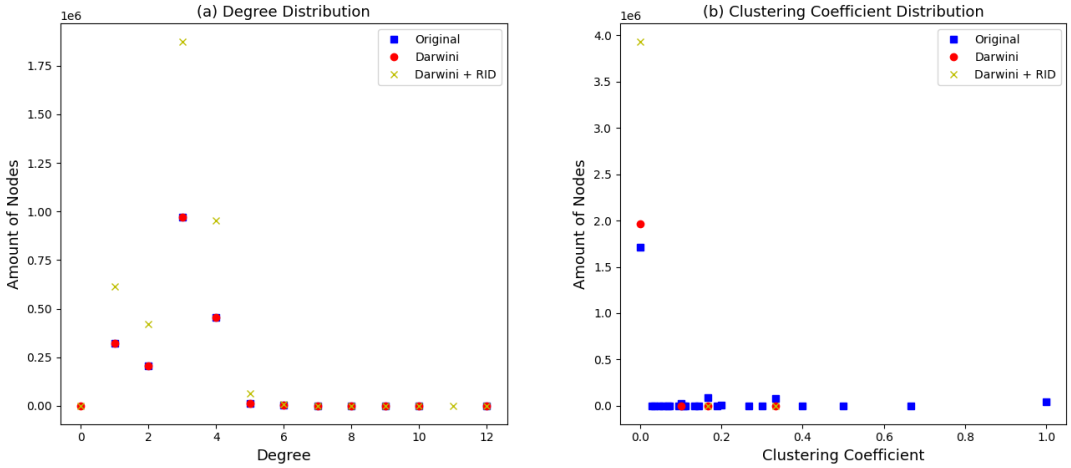

Figure 6.9: Comparing Darwini and RID models under different graph metrics on California road network. Darwini manages to match the degree distribution curve quite accurately.

with the orginal graph's shape. In case of the Wikipedia network, it consists of a single strongly connected component while the outputs of Darwin and RID contain more than 56 and 98 thousand unique components, respectively. The same trend is exhibited by the remaining networks. Inside the right diagram of each previously denoted figure, one can perceive that Darwini puts out together more accurate clustering coefficient curve than RID. The main reason for this outcome is the fact that RID copies each vertex's target degree and clustering coefficient and then increases the target degree by randomly picking a pair of nodes. However, during the course of this process we do not change the clustering coefficient of any vertex. Notwithstanding the fact vertex's clustering property, whose degree gets increased, should be dynamically adjusted in the first stage. However, it is not an easy task to predict in any way the amount of triangles the corresponding vertex will belong to because we are not aware of the future structure of the final network.

Following our argumentation in the last paragraph about absent accommodation of vertex's local clustering characteristic, one can argue that we should expect a densification process for real-world networks, as stated and proven for various domains in [LKF05]. While this is a good argument, one should also ask himself a couple of questions in this direction: How denser will each community become? How many new communities will arise? Will any old groups merge into a larger one? Can existing communities potentially disappear after a given period of time? Also, could some larger and/or dense groups become smaller and even sparse after, e.g. the end of some trend? Which edges will be removed in the future? Depending on the domain, which and how many potential movements can drastically affect the structure of the graph? All these questions come to show that designing a generator preserving multiple graph properties is a complex and challenging task. Our comprehensive results manifest the main weaknesses of Darwini. The way it picks for each vertex a single group affects contemporaneously the final graph's underlying structure by creating non-nested communities and reducing the amount of possible candidates that high degree vertices can be linked with. Nevertheless, as we will conclude in the upcoming chapter, the results of Darwini can be drastically improved by allowing mapping of vertices to multiple groups. Since

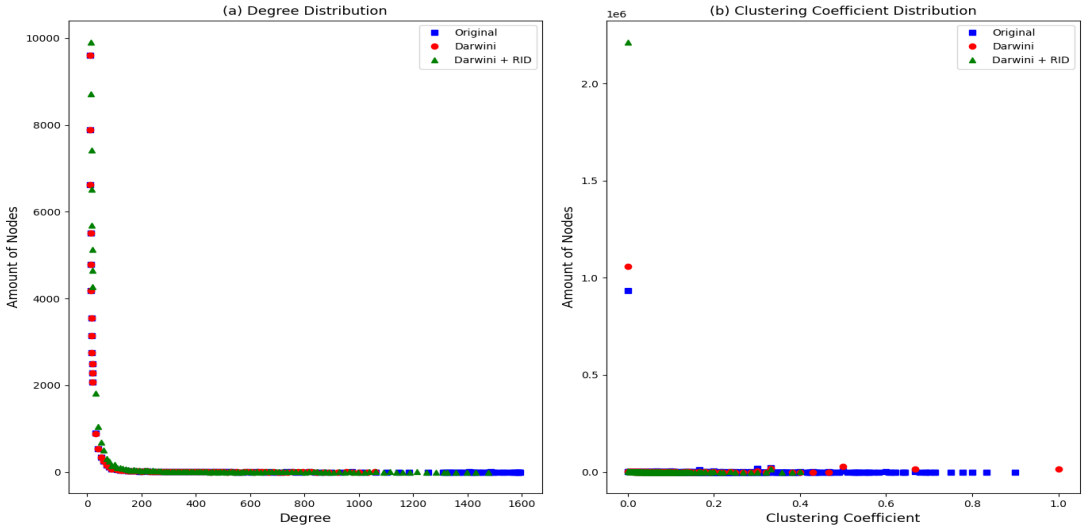

Figure 6.10: Comparing Darwini and RID models under different graph metrics on Youtube network. The non-hierarchical structure of Darwini makes it impossible to match the clustering properties.

graphs with nested communities will be generated while nodes possessing high degree would have more candidate options to link to and thus meet their target degree. Our evaluation has further revealed that calibrating only vertex's degree during generation of a multiple graph does not lead to accurate results. Leaving aside the assignment of each node to a single bucket, RID and LPA would be incapacitate of capturing graph structures due to missing configuration of local clustering properties. From this moment forth each metric we measure and observe will be implicitly altered to a certain degree. Nevertheless, work aimed at designing model predicting how local clustering would change over time, should result in overall considerable metric improvements originating from strict structure estimation.

Before outlining future work towards improvement of Darwini and our growth prediction models, we would like to point out a couple of things about the results retrieved for DBLP and Facebook subgraph in [ELW+16]. If we compare our degree distribution measurements in Figure 6.8 (a) and with those in [ELW+16] we could see that Darwini matches the original curve perfectly in the [0,30] x-axis region. However, we have restricted the x-axis to 350 in Figure 6.8 (a), where one can see that Darwini steadily begins to mismatch entries for nodes with degree 175 or higher. As a consequence the graph structure and local clustering characteristics of nodes are being heavily impacted, depicted in Figure 6.8 (b) and in the results section in[ELW+16]. Those results lead to partially isolated communities due to the absence of bridge vertices thus producing a graph with isolate and/or sparse communities. And as a consequence, each remaining metric would be effected from the set of missing key connections as we already saw in the face of clustering coefficient distribution. This also explains the good degree and local clustering properties estimation found in [ELW+16] for the Facebook subgraph. Facebook limits the amount of friends each person can establish up to 5000. Therefore, it is highly unlikely that first, multiple holes are available in the distribution series. And second, in case of such holes the degree difference between neighbouring

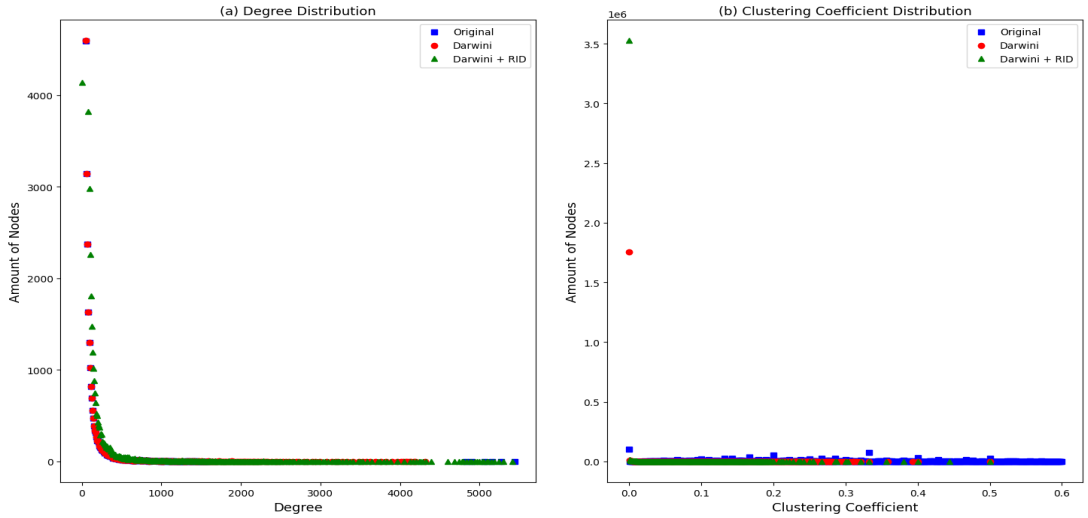

Figure 6.11: Comparing Darwini and both growth prediction models under different graph metrics on Wikipedia network.

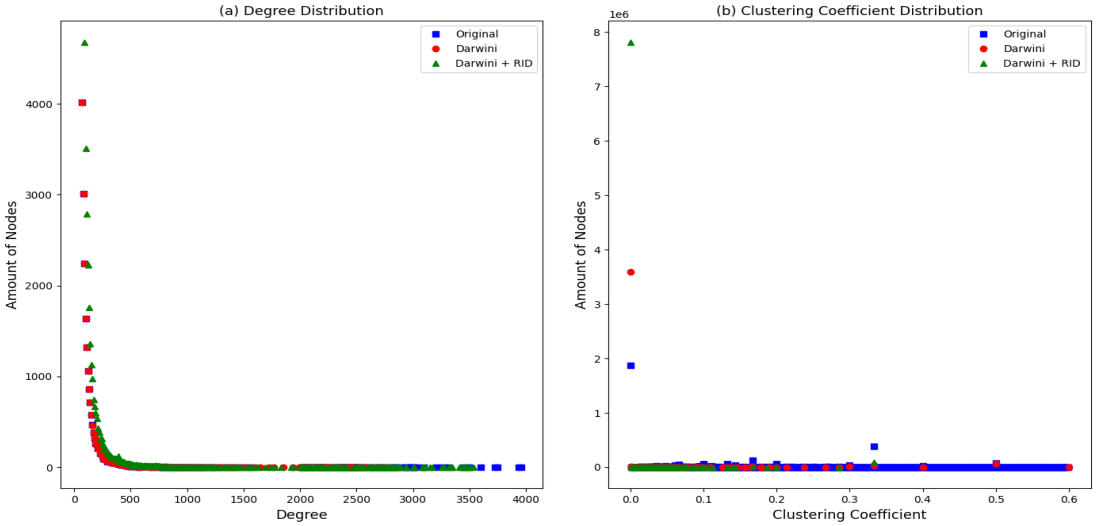

Figure 6.12: Comparing Darwini and both growth prediction models under different graph metrics on LiveJournal network.

elements in the series would reach a couple of thousand, since largest difference that could possibly occur in this scenario is 5000. From the depicted diagram in [ELW+16] this is certainly not the case unlike real-world networks including Wiki, YouTube and LiveJournal.

7 Conclusion and Outlook

We have seen that a great variety of generation models exist. Some of them appertain to state-of-the-art-models such as R-MAT [CZF04] and Forest-Fire [LKF05], while others like Erdös-Renyi [ER+60] and CL [ACL01] are being used as base for the development of more complex approaches like BTER [KPPS14]. Nonetheless, up to the time of writing and to the best of our knowledge, every single model is not precise enough in preserving the following graph characteristics: Matching the graph structure is arguably the prime goal that has to be set when developing generation model. The ability of capturing abrupt changes in the degree distribution curve and thus estimating the proper amount of nodes with certain degree sticks together to some extent with the end structure of the graph. Both aspects can either directly or indirectly influence local clustering properties. Aim at reproducing additional metrics like radius and diameter is also desired. The overall process becomes even more challenging when only small portion of network data can be shared in order to concern user privacy and protect sensible data.

Graph generation models can not be applied to diverse domains due to several factors. First some of them like R-MAT are specifically designed to produce networks with hierarchical community structure and a set of nodes serving as bridge by establishing connection between groups. Such formations are common for online social networks, where for example drivers of distinct transport vehicles, e.g. cars and trucks, can be part of a group where one can share about road incidents and thus inform other drivers travelling through the route about the danger. Not all networks however exhibit nested groups. Email service networks would highly unlike be composed of such types because most users send messages to different institutions, friends, colleagues and others. Further, several methods ignore the fact that graphs can also be directed with multiple edges between a pair of vertices and simply interpret unidirectional edges as bidirectional, which completely changes the structure. In the same fashion weight of edges is represented by a hop in order to reduce the complexity of given generation model. And lastly, big part of the studies focus on small amount of static datasets typically belonging to the same domain. Therefore, when testing generators on restricted data, it is generally hard to capture possible weaknesses caused by specific structures and possible restrictions like e.g. upper bound of connections for a vertex found in social networks like Facebook.

So, due to no presence of studies in analysing Darwini, we deeply investigated this fascinating approach using multiple datasets from unassociated domains. From the retrieved results we came to the conclusion that Darwini lacks the ability of generating graphs matching the original underlying structure. Reason for this being the assumption that each vertex belongs to a single groups, also observed in [KPPS14] and [ELW+16]. Missing node distribution across multiple buckets impacts the way communities are linked with one another due to absence of high degree vertices. The later occur when significant differences in the degrees of two neighbouring elements in the degree distribution series, known as holes, are present. In the appearance of those holes, Darwini struggles to produce vertices incident to huge amount of edges due to a mixture of mapping each vertex to one group and to some extent affected by its effort of preserving the degree correlation. Even when no

major holes exist in the degree distribution, the algorithm is incapacitated from providing accurate results for the remaining metrics cause by the falsely estimated graph structure. Regarding our proposed growth prediction models, we can summarise that they both do not adjust the clustering coefficients. The motive for leaving them unmodified origins from the unknown structure of the multiple graph, i.e. how will communities evolve and how many new communities will be formed. RID is in general more suitable for non-nested community based graphs. Opposite, LPA fits better for real-world data like online social networks by predicting increase of popularity for a given person or trend. And lastly, in the succeeding section we will point out several suggestions for both Darwini and our proposed growth prediction models, which could possibly explicitly and implicitly lead to notable improvements of the output graph and all measured on it metrics.

Outlook

In this section we résumé couple of aspects that could further improve the results and expand the domains, where Darwini can be utilized. First, we will start with the fact that Darwini currently generates only unweighted and undirected, i.e. simple graphs. This makes Darwini inadmissible for reproducing certain online social networks such as Instagram, where people *follow* each other. So, we have to clearly distinguish between a single unidirectional connection and two unidirectional links in both directions. It is possible to convert such types of graphs to undirected but this would not lead to accurate results. Up to the time of writing and to the best of our knowledge there is no well known approach to how Darwini can be adjusted to support weighted edges.

By providing additional metadata denoting the labels for each vertex, one could potentially retrieved synthetic graph with structure consistent to the original one. Another approach would to approximate for each vertex the number of communities it should be part of. The last approach requires however in-depth investigation of numerous networks. Of course either method would introduce additional complexity since one would have to guarantee non-concurrent update of the degree. But even an additional adjustment of the probability formula preserving degree correlation may be required in order to find candidates for high degree vertices. This way by estimating the form and hierarchy of communities, Darwini would quite possibly implicitly improve in a certain degree the results for each metric. With regards to our prediction models, method dynamically fixing the clustering properties of each node with increased degree is required. Otherwise, the output graph would be composed of a incorrect structure cause by wrongly grouping vertices based on their degree and clustering coefficient. In addition to that, Linear Preferential Attachment (LPA) is also not suitable for graphs restricting the maximum amount of connections per vertex. This could be fixed by defining an additional variable monitoring the current degree of each node and thus not allowing to exceed certain value. Scalability of LPA can be considered as a bigger challenge since it current complexity is not suitable for generation of large graphs. Increasing the edge probability after certain number of iterations perhaps should reduce the execution time. Lastly, as for our implementation of Darwini within the GAME framework, we would like to point out that currently first stage is being sequentially executed. Parallelisation can be achieved only by using data structures allowing concurrent modification, otherwise the introduction of barriers and synchronized execution of certain blocks provides little to none performance boost.

Bibliography

[AB02] R. Albert, A.-L. Barabási. “Statistical mechanics of complex networks”. In: *Reviews of modern physics* 74.1 (2002), p. 47 (cit. on pp. 28, 43).

[ACL01] W. Aiello, F. Chung, L. Lu. “A random graph model for power law graphs”. In: *Experimental Mathematics* 10.1 (2001), pp. 53–66 (cit. on pp. 18, 28, 65).

[BB05] V. Batagelj, U. Brandes. “Efficient generation of large random networks”. In: *Physical Review E* 71.3 (2005), p. 036113 (cit. on p. 27).

[BFF+09] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, C. E. Leiserson. “Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks”. In: *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*. 2009, pp. 233–244 (cit. on p. 32).

[CEK+15] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, S. Muthukrishnan. “One trillion edges: Graph processing at facebook-scale”. In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 1804–1815 (cit. on p. 13).

[CVW19] S. Chin, J. Vos, J. Weaver. “JavaFX, the Web, and Cloud Infrastructure”. In: *The Definitive Guide to Modern Java Clients with JavaFX*. Springer, 2019, pp. 367–409 (cit. on p. 31).

[CZF04] D. Chakrabarti, Y. Zhan, C. Faloutsos. “R-MAT: A recursive model for graph mining”. In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM. 2004, pp. 442–446 (cit. on pp. 18, 19, 26, 27, 34, 65).

[DKPS13] N. Durak, T. G. Kolda, A. Pinar, C. Seshadhri. “A scalable null model for directed graphs matching all degree distributions: In, out, and reciprocal”. In: *2013 IEEE 2nd Network Science Workshop (NSW)*. IEEE. 2013, pp. 23–30 (cit. on pp. 18, 19, 28).

[ELW+16] S. Edunov, D. Logothetis, C. Wang, A. Ching, M. Kabiljo. “Darwini: Generating realistic large-scale social graphs”. In: *arXiv preprint arXiv:1610.00664* (2016) (cit. on pp. 13–15, 20, 23, 26, 29, 34–40, 46, 47, 53, 58–60, 62, 63, 65).

[ER+60] P. Erdos, A. Rényi, et al. “On the evolution of random graphs”. In: *Publ. Math. Inst. Hung. Acad. Sci* 5.1 (1960), pp. 17–60 (cit. on pp. 18, 26, 27, 33, 37, 39, 65).

[Gar17] B. Garcia. *Mastering Software Testing with JUnit 5: Comprehensive guide to develop high quality Java applications*. Packt Publishing Ltd, 2017 (cit. on p. 32).

[Gil59] E. N. Gilbert. “Random graphs”. In: *The Annals of Mathematical Statistics* 30.4 (1959), pp. 1141–1144 (cit. on pp. 18, 26, 27, 33).

[Gup03] S. Gupta. *Logging in Java with the JDK 1.4 Logging API and Apache log4j*. Springer, 2003 (cit. on p. 31).

[IHN+16] A. Iosup, T. Hegeman, W. L. Ngai, S. Heldens, A. Prat-Pérez, T. Manhardto, H. Chafio, M. Capotă, N. Sundaram, M. Anderson, et al. "LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms". In: *Proceedings of the VLDB Endowment* 9.13 (2016), pp. 1317–1328 (cit. on p. 14).

[Joh14] R. A. Johnson. "Java database connectivity using SQLite: A tutorial". In: *International Journal of Information, Business and Management* 6.3 (2014), p. 207 (cit. on p. 31).

[KLEC16] M. Kabiljo, D. Logothetis, S. Edunov, A. Ching. "A comparison of state-of-the-art graph processing systems". In: *Facebook Blog Post, http://tinyurl. com/giraph-vs-graphx* (2016) (cit. on p. 14).

[KNT10] R. Kumar, J. Novak, A. Tomkins. "Structure and evolution of online social networks". In: *Link mining: models, algorithms, and applications*. Springer, 2010, pp. 337–357 (cit. on pp. 13, 17, 41).

[KPPS14] T. G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri. "A scalable generative graph model with community structure". In: *SIAM Journal on Scientific Computing* 36.5 (2014), pp. C424–C452 (cit. on pp. 13, 18, 20, 23, 28, 29, 35–37, 40, 47, 65).

[LCK+10] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, Z. Ghahramani. "Kronecker graphs: an approach to modeling networks." In: *Journal of Machine Learning Research* 11.2 (2010) (cit. on p. 27).

[LK14] J. Leskovec, A. Krevl. *SNAP: Stanford network analysis project*. 2014 (cit. on pp. 49, 50).

[LKF05] J. Leskovec, J. Kleinberg, C. Faloutsos. "Graphs over time: densification laws, shrinking diameters and possible explanations". In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 2005, pp. 177–187 (cit. on pp. 13, 14, 19, 20, 23, 28, 41, 53, 61, 65).

[LKF07] J. Leskovec, J. Kleinberg, C. Faloutsos. "Graph evolution: Densification and shrinking diameters". In: *ACM transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), 2–es (cit. on pp. 49, 51).

[LLDM09] J. Leskovec, K. J. Lang, A. Dasgupta, M. W. Mahoney. "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters". In: *Internet Mathematics* 6.1 (2009), pp. 29–123 (cit. on p. 50).

[LT16] J. Lu, A. Thomo. "An experimental evaluation of giraph and graphchi". In: *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE. 2016, pp. 993–996 (cit. on p. 14).

[MAB+10] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, G. Czajkowski. "Pregel: a system for large-scale graph processing". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010, pp. 135–146 (cit. on p. 14).

[MKNS20] D. Michail, J. Kinable, B. Naveh, J. V. Sichi. "JGraphT—A Java Library for Graph Data Structures and Algorithms". In: *ACM Transactions on Mathematical Software (TOMS)* 46.2 (2020), pp. 1–29 (cit. on pp. 31, 33).

[MVM10] F. P. Miller, A. F. Vandome, J. McBrewster. *Apache Maven*. Alpha Press, 2010 (cit. on p. 31).

[New18] M. Newman. *Networks*. Oxford university press, 2018 (cit. on pp. 24–26, 32).

[RAS19] B. Rozemberczki, C. Allen, R. Sarkar. *Multi-scale Attributed Node Embedding*. 2019. arXiv: 1909.13021 [cs.LG] (cit. on pp. 50, 53).

[RDSS19] B. Rozemberczki, R. Davies, R. Sarkar, C. Sutton. "GEMSEC: Graph Embedding with Self Clustering". In: *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019*. ACM. 2019, pp. 65–72 (cit. on p. 49).

[TBC+13] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, J. McPherson. "From"think like a vertex"to"think like a graph"". In: *Proceedings of the VLDB Endowment* 7.3 (2013), pp. 193–204 (cit. on p. 44).

[UKBM11] J. Ugander, B. Karrer, L. Backstrom, C. Marlow. "The anatomy of the facebook social graph". In: *arXiv preprint arXiv:1111.4503* (2011) (cit. on pp. 13, 40, 41).

[VCG+18a] J. Vos, S. Chin, W. Gao, J. Weaver, D. Iverson. "Getting a Jump-Start in JavaFX". In: *Pro JavaFX 9*. Springer, 2018, pp. 1–32 (cit. on p. 31).

[VCG+18b] J. Vos, S. Chin, W. Gao, J. Weaver, D. Iverson. "Using Scene Builder to Create a User Interface". In: *Pro JavaFX 9*. Springer, 2018, pp. 129–191 (cit. on p. 31).

[WPHV15] R. Wang, Y. Perez-Riverol, H. Hermjakob, J. A. Vizcaino. "Open source libraries and frameworks for biological data visualisation: A guide for developers". In: *Proteomics* 15.8 (2015), pp. 1356–1374 (cit. on p. 31).

[WS98] D. J. Watts, S. H. Strogatz. "Collective dynamics of 'small-world'networks". In: *nature* 393.6684 (1998), pp. 440–442 (cit. on pp. 26, 27, 33).

[XGFS13] R. S. Xin, J. E. Gonzalez, M. J. Franklin, I. Stoica. "Graphx: A resilient distributed graph system on spark". In: *First international workshop on graph data management experiences and systems*. 2013, pp. 1–6 (cit. on p. 14).

[YL15] J. Yang, J. Leskovec. "Defining and evaluating network communities based on ground-truth". In: *Knowledge and Information Systems* 42.1 (2015), pp. 181–213 (cit. on p. 50).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Sofia, den 19.05.2021 Valentin Marianov

place, date, signature