On Merging and Dividing of Barabási-Albert-Graphs

Pascal Held Alexander Dockhorn Rudolf Kruse
Department of Knowledge Processing and Language Engineering
Faculty of Computer Science
Otto von Guericke University Magdeburg, Germany
pascal.held@ovgu.de
alexander.dockhorn@st.ovgu.de
rudolf.kruse@ovgu.de

***Abstract*—The Barabási-Albert-model is commonly used to generate scale-free graphs, like social networks, or a lot of other natural networks. To generate dynamics in these network, methods for altering such graphs are needed. Growing and shrinking is done trivial by doing some further generation iterations or undo them. In our paper we present four methods to merge two graphs based on the Barabási-Albert-model, and five strategies to divide them. First we compared these algorithms by edge preservation, which describes the ratio of the inner structure kept after altering. To check if hubs in the initial graphs are hubs in the resulting graphs as well, we used the node-degree rank correlation. Finally we tested how well the node-degree distribution follows the power-law function from the Barabási-Albert-model.**

I. Introduction

The field of graph modeling is full of methods for the generation of graphs like the Erdős-Rényi-model [1] and the Watts-Strogatz-model [2]. However both do not exhibit a power law degree distribution (scale-free network) observed in many natural networks. In contrast the Barabási-Albert-model [3], [4] is able to produce such networks. This can be used to model networks like the world wide web [5], [6] or movie co-occurrences between Hollywood actors [7], which are proofed to exhibit a power law degree distribution.

Nevertheless the constant growing of the model does not explain all features that can occur in real-world networks. For instance networks are able to split into two separate networks for example a group of school-friends which is falling apart after graduation or the well-known karate club data set presented by Zachary in [8]. Also plausible is a merge of two previously distinct networks into one cooperate network. For example such an behavior could be observed at the fusion of two enterprises after an acquisition.

Taking these examples one step further leads us to think about characteristics of the resulting graphs. We can guess that some structures in the group of school-friends are most likely to still exist in the divided groups, missing some connections between each other. The karate club dataset shows that persons having a lot of connections e.g. the club leader, will be high connected after the divide as well. For the merge of two cooperate networks it could be guessed that the structure of the absorbing company A will not change significant. Whereas the acquired company B could be restructured such that employees will be integrated to company A's structure. Also possible is that both company structures are nearly untouched and just a few connections between both parties arise due to an increased cooperation.

This paper focuses on modeling merge and divide algorithms of graphs following the Barabási-Albert-model, while taking account of characteristics of previous examples. The Barabási-Albert-model belongs to the class of generative models. It is making use of a preferential attachment strategy and results in a scale-free network. Divided (merged) graphs should still share typical properties of the Barabási-Albert-albert model. For this purpose we propose a set of methods focusing on the preservation of different graph properties.

We believe proposed algorithms will have implications for fields of computational intelligence and complex network theory. For example merges and divides form a baseline for new hierarchical clustering methods specific to scale-free networks, which are known to exhibit hierarchical organization characteristics [9]. As well a recent study showed application of scale-free networks for particle swarm optimization [10]. Those networks can be further optimized using evolutionary algorithms, where merges and divides can be used to define a crossover operation without loosing scale-free characteristics of the network.

The rest of the paper is structured as follows. Section II introduces basic ideas of merging and splitting graphs in the field of computer science and gives a quick overview about properties of Barabási-Albert-Graphs. The third section describes methods for merging and dividing Barabási-Albert-Graphs followed by experiments explained in Section IV. In Section V found results will be discussed and finally concluded in the last section.

II. Related Work

A. Graphs

First we want to introduce a basic graph notation. Be a graph $G = (V, E)$, a set of nodes V and a set of edges E such that $E = \{(u, v) \mid u \neq v;\ u, v \in V\}$. Therefore edges

represent an undirected link between the two nodes u and v. For sake of simplicity the edges $e = (u, v)$ and $e' = (v, u)$ be the same. We will use the notation $V(g_i)$ and $E(g_i)$ to differentiate between the nodes and edges of graph g_i.

The number of links of a node $n \in V$ indicate its node-degree $k_n = |\{e = (u, n) \mid e \in E;\ u \in V\}|$ and $P(k)$ be the degree distribution of the network.

Additionally a graph can consist of a set of connected components. A component is a subgraph in which any two nodes are connected to each other by at least one path, and which is connected to no additional nodes.

Merging of two subgraphs $g1 = (V(g_1), E(g_1))$ and $g2 = (V(g_2), E(g_2))$ is defined by creating a new graph g such that $V(g) = V(g_1) \cup V(g_2)$, where $E(g)$ contains at least one edge $e = (u, v),\ u \in V(g_1),\ v \in V(g_2)$. Dividing a graph into two subgraphs works vice versa. The nodes of the divided graph will be distributed to the subgraphs g_1 and g_2 while holding the condition $V(g_1) \cap V(g_2) = \emptyset$.

B. Barabási-Albert-model

The Barabási-Albert-model [3] provides an algorithm for the generation of random scale-free networks. The scale-free properties can be observed in different natural systems such as the world wide web or social networks.

The creation starts with an initial set of m_0 nodes. Every new node will be connected to nodes in the graph using m edges, where $m \leq m_0$. Adding a node is handled using the preferential attachment mechanism. Therefor new nodes are more likely to connect to nodes with a high node-degree. The probability for a new node connecting to a node *n* is

$$p_n = \frac{k_n}{\sum_j k_j} \tag{1}$$

where k_n is the node-degree of node n, which is divided by the sum of all node-degrees. This results in the development of heavily linked nodes called hubs, which are linked to a great part of the graph. More generally the degree distribution of the full graph follows a power law of the form

$$P(k) \sim k^{-\gamma}; \quad \gamma = 2.9 \pm 0.1 \tag{2}$$

By the definition of the preferential attachment strategy older nodes have higher chances to become hubs. In the case of $m = m_0$ we recommend to use a fully connected start graph for the m_0 nodes. Otherwise the model will be biased to favor the $(m_0 + 1)$-node, because it has the maximal node-degree. We will make use of this throughout our proposed methods for merging and splitting Barabási-Albert-Graphs.

Figure 1 shows two example graphs. The left one was created using the Erdős-Rényi-model, whereas the right graph follows the Barabási-Albert-model. Typical for the latter one is that it is always one connected component and some nodes have a much higher node-degree.

The Barabási-Albert-model was extended using a fitness model described in [11]. This can help to explain how new nodes can become hubs very fast for example the rise of Google as one of the most linked webpages of the world wide web. However such extensions will not be regarded in this paper.

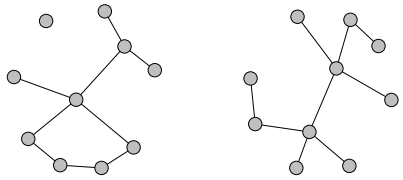

Fig. 1. Comparison of graph models, left: Erdős-Rényi-model with edge probability of 0.25, right: Barabási-Albert-model with $m_0 = m = 1$

III. Altering Barabási-Albert-Graphs

The Barabási-Albert-model itself is based on the approach of adding one node at a time and connecting this to other nodes by preferential attachment. Based on this, growing is trivial by doing some further iterations.

To shrink a given graph, we can do this process backward. One of the last added nodes should be removed, including all its edges. From the growing process, we know that every new nodes starts with exactly m edges. So at least one node should have exactly m edges, which is a good candidate to be removed. More generally spoken, we simply remove the node with the smallest node-degree. This also fixes the problem of dealing with graphs that include noise and did not fully follow the Barabási-Albert-model.

A. Estimating Barabási-Albert-Model Attributes

To alter a graph which is based on a Barabási-Albert-model, we have to know the attributes of the generation process. These are the number of nodes m each new node is connected with and the number of nodes m_0 which are initialized before the growing progress.

The number of nodes n_t is the total number of nodes in the graph and $n_a = n_t - m_0$ the number of nodes added in the growing phase. Variable e_t is the total number of edges in the graph and it is the sum of edges from the initialization phase e_i and the edges from the growing phase $e_a = n_a \cdot m$.

Depending on the assumption of the initialization e_i is between 0 (starting with no edges at all) and $0.5 \cdot m_0 \cdot (m_0 - 1)$ (full-connected graph).

Based on this, we get:

$$n_a \cdot m \quad \leq e_t \leq \quad n_a \cdot m + \frac{m_0 \cdot (m_0 - 1)}{2} \tag{3}$$

$$\frac{e_t}{n_a} \quad \geq m \geq \quad \frac{e_t}{n_a} - \frac{m_0 \cdot (m_0 - 1)}{2n_a} \tag{4}$$

$$\frac{e_t}{n_t - m_0} \quad \geq m \geq \quad \frac{e_t}{n_t - m_0} - \frac{m_0 \cdot (m_0 - 1)}{2(n_t - m_0)} \tag{5}$$

For the case that $m = m_0$ and we are always starting with a full-connected graph of m nodes the problem can be reduced

to:

$$e_t = (n_t - m) \cdot m + \frac{m \cdot (m-1)}{2} \quad (6)$$

$$0 = m^2 - 2 \cdot (n_t - \frac{1}{2}) \cdot m + 2 \cdot e_t \quad (7)$$

$$m_{1,2} = n_t - \frac{1}{2} \pm \sqrt{(n_t - \frac{1}{2})^2 - 2 \cdot e_t} \quad (8)$$

Experiments showed that subtracting the value of the square root in Equation 8 results in a correct estimation of m.

Furthermore we will need an estimation of the parameter m for a merge-graph g of two Barabási-Albert-Graphs g_1 and g_2. This will be trivial if both graphs have an equal parameter m such that $m_1 = m_2$. In this case we can separately estimate m_1 and m_2 for both subgraphs and return $m = m_1 = m_2$. Otherwise we will have to find a value for m big enough to reach at least the same number of edges in the merge graph as the sum of edges in both subgraphs ($|E(g)| \geq |E(g_1)| + |E(g_2)|$). The estimation of the parameter m of a merge graph is described in the following algorithm.

function ESTIMATEM(Graph: g_1, g_2)
  $m_1 \leftarrow calculateM(g_1)$
  $m_2 \leftarrow calculateM(g_2)$
  $n_t \leftarrow |V(g_1)| + |V(g_2)|$
  for $m \leftarrow range(m1, m2)$ **do**
    $e \leftarrow (n_t - m) \cdot m + \frac{m \cdot (m-1)}{2}$
    if $e \geq |E(g_1)| + |E(g_2)|$ **then**
      return m
    end if
  end for
end function

B. Merging of two Graphs

In the following we describe four strategies to merge two graphs.

1) Random Merge: The simplest way to merge two graphs is a random merge (RM). The basic idea is to pick all nodes from both graphs and add them in a random order to a new Barabási-Albert-Graph.

function RANDOMMERGE(Graph: g_1, g_2)
  $m \leftarrow estimateM(g_1, g_2)$
  $graph \leftarrow EmptyBarabasiGraph(m)$
  $nodes \leftarrow V(g_1) \cup V(g_2)$
  $nodes.shuffle()$
  for all $node \leftarrow nodes$ **do**
    $graph.addNode(node)$
  end for
  return $graph$
end function

The complexity of this algorithm is $\mathcal{O}(n)$, where n is the number of nodes in both graphs.

This simplest strategy should be used as a baseline to evaluate the other strategies. The only thing, which could be expected is, that the Barabási-Albert-model properties hold in the resulting graph.

2) Node-Degree-Order Merge: The second method node-degree-order merge (NDOM) focuses on keeping the node-degree of the nodes. As in the random-merge strategy we create a new graph with the Barabási-Albert-model. We take all nodes of both graphs and put them into a combined list. The nodes were sorted by node-degree of the origin graphs in descending order. All nodes in this list would be added in this sequence into the new graph.

function NODEDEGREEMERGE(Graph: g_1, g_2)
  $m \leftarrow estimateM(g_1, g_2)$
  $graph \leftarrow EmptyBarabasiGraph(m)$
  $nodes \leftarrow V(g_1) \cup V(g_2)$
  $nodes.sortByNodeDegree('desc')$
  for all $node \leftarrow nodes$ **do**
    $graph.addNode(node)$
  end for
  return $graph$
end function

The complexity of this algorithm is $\mathcal{O}(n \cdot \log n)$, where n is the number of nodes in both graphs. The main complexity is a result of the sorting operation.

With this strategy it is expected, that the node-degree distribution relating to the specific nodes is the same. This means, that nodes which have an high node-degree in the origin graphs, also have an high node-degree in the resulting graph.

One example use case for this scenario could be the change from school to university from pupils. Different groups from different schools would be merged. Extroverted people, that one with more connections, will be still more active afterwards.

3) Preserving-Nodes Merge: The next strategy is the preserving-nodes merge (PNM). The main idea is to keep the full structure of one graph and merge the other. To do so, we start with the first graph and insert the nodes of the second graph in descending node-degree order.

function PRESERVINGNODESMERGE(Graph: g_1, g_2)
  $m \leftarrow estimateM(g_1, g_2)$
  $nodes \leftarrow V(g_2)$
  $nodes.sortByNodeDegree('desc')$
  for all $node \leftarrow nodes$ **do**
    $g_1.addNode(node)$
  end for
  return $graph$
end function

The complexity of this algorithm is $\mathcal{O}(n \cdot \log n)$, where n is the number of nodes in the second graph

From construction this strategy keeps all the information from the first graph. The inner structure of the second graph is lost, but the node-degree distribution relating to the specific nodes of the second graph is the same. Based on the fact, the first graph is used as the base for the merge, these nodes have an higher probability to get connected with the new nodes then nodes from the second graph. This yield into an domination of the nodes from the first graph. Because of this, we suggest

to use the larger graph as base graph.

Relating to the company acquisition example of the introduction the output of this algorithm will minimally change the structure of company A. Managers of company B will be added first and therefor end up with more connections.

4) Minimal-Merge: Our last strategy to merge two Barabási-Albert-graphs is the minimal merge (MM). It focuses on keeping most of the structure of both graphs.

The main idea is to use both graphs and connect them with additional edges. To do so, we increase the estimated m. Now we have free edges, we can use to connect both graphs. Similar to the basic Barabási-Albert-approach we select nodes proportional with respect to there node-degree.

function MINIMALMERGE(Graph: g_1, g_2)
  $m \leftarrow estimateM(g_1, g_2) + 1$
  $g \leftarrow UnionBarabasiGraph(g_1, g_2, m)$
  $e_{add} = g.getMaxEdges() - |E(g)|$
  while $e_{add} > 0$ **do**
    $n_1 \leftarrow V(g_1).preferedSelect()$
    $n_2 \leftarrow V(g_2).preferedSelect()$
    $g.addEdge(n_1, n_2)$
    $e_{add} \leftarrow e_{add} - 1$
  end while
  return *graph*
end function

The complexity of this algorithm is $\mathcal{O}(n)$. This strategy keeps most of the structure, with the drawback of increasing the number of edges per node. This is related to an increased cooperation between two companies.

C. Dividing into two Graphs

This subsection will explain five strategies to divide a graph in two subgraphs. Each algorithm will use the parameters graph (g) and the number of nodes expected in the first subgraph ($noNodes_1$).

1) Random-Divide: The random-divide strategy (RD) is the simples idea to divide a given graph. It is correlated to the RM-Strategy. The basic idea is to create two sets of nodes, for each new graph one. Then create a new Barabási-Albert-graph from both of these sets.

function RANDOMDIVIDE(Graph: g, $noNodes_1$)
  $v_1 \leftarrow V(g).randomSelect(noNodes_1)$
  $v_2 \leftarrow V(g) - v_1$
  $m \leftarrow estimateM(g)$
  $g_1 \leftarrow EmptyBarabasiGraph(m)$
  $g_2 \leftarrow EmptyBarabasiGraph(m)$
  for all $node \leftarrow v_1$ **do**
    $g_1.addNode(node)$
  end for
  for all $node \leftarrow v_2$ **do**
    $g_2.addNode(node)$
  end for
  return g_1, g_2
end function

The complexity of the random-divide strategy is $\mathcal{O}(n)$, where n is the number of nodes. This simplest method does not care about the underlying structure of the graph, but it ensures the Barabási-Albert-properties in the resulting graphs.

2) Random-Subgraph-Divide: The random-subgraph divide strategy (RSD) is based on the Random-Divide. We randomly split the graph into two subgraphs, but we do not create new Barabási-Albert-graphs. Instead, we use the existing edges. This yield into violations of the Barabási-Albert-properties, so we have to do some repairing steps, described in Subsection III-D.

function RANDOMSGDIVIDE(Graph: g, $noNodes_1$)
  $v_1 \leftarrow V(g).randomSelect(noNodes_1)$
  $v_2 \leftarrow V(g) - v_1$
  $e_1 = \{(x_1, x_2) : \forall (x_1, x_2) \in E(g) \wedge x_1, x_2 \in v_1\}$
  $e_2 = \{(x_1, x_2) : \forall (x_1, x_2) \in E(g) \wedge x_1, x_2 \in v_2\}$
  $g_1 \leftarrow Graph(v_1, e_1)$
  $g_2 \leftarrow Graph(v_2, e_2)$
  $m_1 \leftarrow estimateM(g_1)$
  $m_2 \leftarrow estimateM(g_2)$
  $repairGraph(g_1, m_1)$
  $repairGraph(g_2, m_2)$
  return g_1, g_2
end function

The complexity of our second divide algorithm is $\mathcal{O}(n^2)$, where n is the number of nodes.

3) Node-Degree-Divide A: Related to the NDOM we can divide the graph by node-degree (NDDa). First we order all nodes descending by their node-degree. We split this list and use the first part for the first subgraph and the second respectively. The nodes are added into a new Barabási-Albert-graph with respect to their order.

function NODEDEGREEDIVIDEA(Graph: g, $noNodes_1$)
  $m \leftarrow estimateM(g)$
  $g_1 \leftarrow EmptyBarabasiGraph(m)$
  $g_2 \leftarrow EmptyBarabasiGraph(m)$
  $nodes \leftarrow V(g)$
  $nodes.sortByNodeDegree('desc')$
  for all $node \leftarrow nodes[: noNodes]$ **do**
    $g_1.addNode(node)$
  end for
  for all $node \leftarrow nodes[noNodes :]$ **do**
    $g_2.addNode(node)$
  end for
  return g_1, g_2
end function

The complexity of this algorithm is $\mathcal{O}(n \cdot \log n)$, where n is the number of nodes in the graph. The most complexity is based on the ordering.

This strategy keeps the Barabási-Albert-properties and also the node-degree distribution related to the nodes.

4) Node-Degree-Divide B (NDDb): This is a modification of the NDDa. We do not split the list into two parts, but instead we pick alternating elements for every subgraph, with respect to the selected relation. The function $createIndexList()$ returns all node indexes of nodes used for the first subgraph such that when possible they are equally spaced to each other.

This ensures, that every subgraph has some of the top connected nodes and overall the node-degrees are well distributed between both graphs. However edges between picked nodes will not be preserved in this strategy as well as with algorithm NDDa.

function NODEDEGREEDIVIDEB(Graph: g, $noNodes_1$)
  $m \leftarrow estimateM(g)$
  $g_1 \leftarrow EmptyBarabasiGraph(m)$
  $g_2 \leftarrow EmptyBarabasiGraph(m)$
  $nodes.sortByNodeDegree('desc')$
  $indexlist \leftarrow createIndexList(|V(g)|,\ noNodes_1)$
  for all $node \leftarrow nodes$ **do**
    if $index(node)\ in\ indexlist$ **then**
      $g_1.addNode(node)$
    else
      $g_2.addNode(node)$
    end if
  end for
  return g_1, g_2
end function

The complexity of strategy NDDb is $\mathcal{O}(n \cdot \log(n))$ where n is the number of nodes. This is equal to the the complexity of NDDa.

5) Subgraph-Expansion-Divide (SED): The subpgrah-expansion-divide algorithm is closely related to the random subset. Main difference is that chosen nodes are definitely one connected component. We achieve this by iteratively adding nodes to the current subgraph till the size of first graph is reached. Which is equal to a breadth-first search (BFS). Remaining nodes will be used as basis for the second subgraph. However it cannot be guaranteed that both graphs approximate properties of the Barabási-Albert-model e.g. the second subgraph can be split into several components. For this reason both graphs will be repaired using the repair-operator described in Subsection III-D.

function SUBGRAPHEXPDIVIDE(Graph: g, $noNodes_1$)
  $startNode \leftarrow g.NodeWithLowestDegree()$
  $v_1 \leftarrow BFS(startNode, noNodes_1)$
  $v_2 \leftarrow V(g) - v_1$
  $e_1 = \{(x_1, x_2) : \forall (x_1, x_2) \in E(g) \wedge x_1, x_2 \in v_1\}$
  $e_2 = \{(x_1, x_2) : \forall (x_1, x_2) \in E(g) \wedge x_1, x_2 \in v_2\}$
  $g_1 \leftarrow Graph(v_1, e_1)$
  $g_2 \leftarrow Graph(v_2, e_2)$
  $m \leftarrow estimateM(g)$
  $repairGraph(g_1, m)$
  $repairGraph(g_2, m)$
  return g_1, g_2
end function

The complexity of our last divide is $\mathcal{O}(e) \sim \mathcal{O}(n)$, where e is the number of edges of graph g and n n is the number of nodes in the graph. The repairing step increases the complexity to $\mathcal{O}(n^2)$.

D. Repairing-Steps

Some of our algorithms produce graphs that do not hold typical properties of the Barabási-Albert-model. We propose a three step repairing process to achieve the minimal properties:

1) Barabási-Albert-graph is always one component
2) Every node has at least m edges
3) The maximal number of edges is $n_a \cdot m + \frac{m \cdot (m_0 - 1)}{2}$

This does not lead to a graph perfectly following the Barabási-Albert-model, but achieves the most properties with minimal manipulation of the graph. Edges added to the graph will first prioritize to connect nodes with degree lower than m. If all nodes already have a degree of m and higher adding an edge will use preferential attachment to decide which nodes to connect. For the case that the number of edges is not high enough an recursive run with $repairGraph(g, m+1)$ will be started.

function REPAIRGRAPH(Graph: g, m)
  if $|V(g)| \leq m$ **then**
    return $completeGraph(V(g))$
  end if
  $e_{add} = getMaxEdges(g) - |E(g)|$
  while $|g.components| > 1$ **do**
    if $e_{add} < 0$ **then**
      return $repairGraph(g, m + 1)$
    end if
    $Connect(g.components.randomSelect(2))$
    $e_{add} \leftarrow e_{add} - 1$
  end while
  for all $node \leftarrow \{n : n \in V(g), n.degree < m\}$ **do**
    while $node.degree < m$ **do**
      if $e_{add} < 0$ **then**
        return $repairGraph(g, m + 1)$
      end if
      $g.addEdge(node, V(g).preferedSelect(1))$
      $e_{add} \leftarrow e_{add} - 1$
    end while
  end for
  while $e_{add} > 0$ **do**
    $g.addEdge(V(g).preferedSelect(2))$
    $e_{add} \leftarrow e_{add} - 1$
  end while
  return g
end function

The repair operator has a complexity of $\mathcal{O}(n^2)$ where n is the number of nodes.

It can be argued that better approximation of the model properties can be achieved by first calculating the desired node-degree distribution using Equation 2 and then use remaining edges to best fit this distribution. However this is against the philosophy of the generative model. Desired node-degrees should be a result of the preferential attachment strategy and not created by force.

IV. Experiments

The following subsections will describe our experiments for evaluating the behavior of proposed merge and divide algorithms for Barabási-Albert-Graphs. Subsection IV-C introduces our comparison measures. Additionally we recorded runtimes of all algorithms. See Section V for results.

A. Merging of two Graphs

We always merged two Barabási-Albert-Graphs and altered the parameters size (n) and connectivity $(m_0 = m)$.

We used the following test scenarios for merging two Barabási-Albert-graphs:

name	n_1	m_1	n_2	m_2
equal	5000	3	5000	3
diff_m	5000	3	5000	8
diff_size	5000	8	25000	8
diff_all	5000	8	25000	3

B. Dividing into two Graphs

Similar to the experiments for merging graphs we divided graphs in different ratios of nodes in the resulting subgraphs. Following test scenarios were created for dividing a Barabási-Albert-graph into two subgraphs:

name	n	m	$noNodes1$
$10:90$	10000	5	1000
$20:80$	10000	5	2000
$30:70$	10000	5	3000
$40:60$	10000	5	4000
$50:50$	10000	5	5000

C. Measurements

To measure the quality of our strategy, we focus on three aspects: edge preservation, node-degree rank and node-degree distribution.

1) Edge Preservation: One goal is to preserve the inner structure of the graphs. This means, that nodes which are connected before should be connected afterwards. The same with not connected nodes.

For this measure we simply calculate the percentage of preserved edges during the altering process.

2) Rank-Correlation: Nodes which are more active, or higher connected should also have a higher node-degree in the resulting graphs. To estimate this, we calculate the node-degree rank of every node in both graphs. Afterwards, we calculate the two well known rank correlation coefficients Spearman's ρ [12, Section 14.7] including tie-correction and Kendall's τ [13]. These measures have a range from -1 to $+1$, where $+1$ (-1) indicates that the order is completely preserved (reversed).

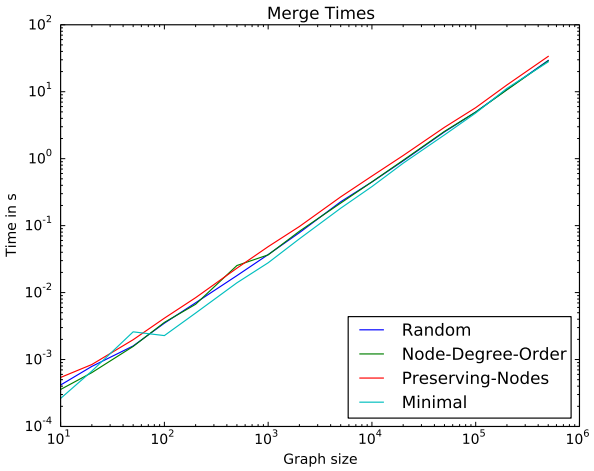

Fig. 2. Computation time for merge operators with different graph sizes

Merge	equal	diff_m	diff_size	diff_all
RM	0.1 / 0.1	0.1 / 0.1	0.1 / 0.1	0.0 / 0.0
NDM	0.4 / 0.4	0.3 / 1.1	0.3 / 0.4	0.5 / 0.1
PNM	100 / 0.0	100 / 0.0	100 / 0.0	100 / 0.0
MM	100 / 100	100 / 100	100 / 100	100 / 100

TABLE I
Average part of edges preserved after merge-operation in percent

3) Node-Degree Distribution: Based on the Barabási-Albert-model the node-degree of all nodes in the graph should follow the power low distribution, described in Equation 2.

We determine the node-degree distribution of the resulting graph and compare them with the theoretical node-degree distribution based on the presented formula and the number of edges in the graph. We calculate the root-mean-squared-error to check how good the distribution fits the node-degree distribution of our graph.

V. Results

In the following section we present the results of our experiments. First we give a brief comparison about the computation times, and afterwards we show the in Subsection IV-C described measures.

A. Merging of two Graphs

1) Computing Time: In Figure 2 we show the computing time for the presented merge algorithms. We used two graphs with the same number of nodes and a connectivity of $m = 4$. The measurement is based on 10 runs for each graph size. The runtime for all algorithms is almost linear in graph-size. This admits that the constants of the logarithmic parts from the ordering are relative small in contrast to the constant factors of the merge process itself.

Merge	ρ_{equal}	ρ_{diff_m}	ρ_{diff_size}	ρ_{diff_all}
RM	0.000	-0.001	0.000	0.001
NDM	0.726	0.842	0.874	0.779
PNM	0.475	-0.134	0.612	0.850
MM	1.000	0.979	1.000	0.985

TABLE II
MEASURED VALUES FOR SPEARMAN'S ρ

Merge	τ_{equal}	τ_{diff_m}	τ_{diff_size}	τ_{diff_all}
RM	0.000	0.000	0.000	0.001
NDM	0.625	0.709	0.745	0.661
PNM	0.406	-0.061	0.486	0.718
MM	1.000	0.927	1.000	0.964

TABLE III
MEASURED VALUES FOR KENDALL'S τ

2) Edge Preservation: The edge preservation follows the design of the algorithms. The Minimal-Merge preserves all edges, and the Preserving-Nodes Algorithms the edges from the first graph. All others algorithms lose the inner structure. A more detailed view could be taken from Table I. The first part of every column describes the percentage of preserved edges from the first resulting subgraph, and the second part the other subgraph respectively.

3) Rank Correlation: Recorded values were averaged over 100 iterations of specified experiments and are shown in Table II and Table III. The baseline algorithm Random-Merge (RM) had always values around 0 which indicates that degree rank order before and after the merge stand in no correlation to each other. Preserving-Node-Merge (PNM) performed better in the merge of two equal graphs and on the same level as RM for graphs with differing m. The Node-Degree-Merge algorithm (NDM) ranked second best in first three experiments. For an equal merge both rank-correlation coefficients had much higher values ($\rho_{equal} = 0.726$ and $\tau_{equal} = 0.625$) than PNM. It reached even higher values for merging graphs with differing m ($\rho_{diff\_m} = 0.842$ and $\tau_{diff\_m} = 0.709$) and differing size ($\rho_{diff\_m} = 0.874$ and $\tau_{diff\_m} = 0.779$). However PNM performed better than NDM for graphs differing in size and connectivity ($\rho_{diff\_all} = 0.850$ and $\tau_{diff\_all} = 0.718$). The algorithm Minimal-Merge (MM) scored best with values near to +1 for both experiments. This is due to the minimal change by adding just a few edges.

4) Node-Degree Distribution: Table IV shows the RMSE between the observed node-degree and the theoretical node-degree distribution. The upper part of the table shows the RMSE for the initial graphs, and below the results for all merged graphs for every method.

The Random-Merge (RM) algorithm generates the best results except the diff_all dataset. Minimal-Merge (MM), yield to huge error, especially with different m. This is based on the fact, that most of the inner structure is kept and no combined graph with all nodes is created from scratch. PNM and NDM generate results with RMSE between the initial graphs.

RMSE	equal	diff_m	diff_size	diff_all
g_1	23.2	22.9	4.9	4.8
g_2	22.6	4.9	12.3	75.7
RM	38.2	11.7	14.0	53.9
NDM	39.4	12.0	14.3	54.2
PNM	38.3	38.2	14.1	14.3
MM	43.3	199.1	14.8	229.2

TABLE IV
RMSE OF NODE-DEGREE DISTRIBUTION BEFORE AND AFTER MERGE

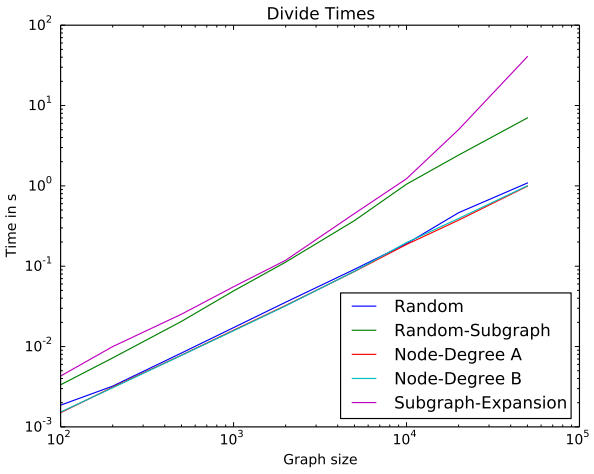

Fig. 3. Computation time for divide operators with different graph sizes

B. Dividing into two Graphs

1) Computing Time: Figure 3 shows the computing time for the presented divide algorithms. We used a graph with a connectivity of $m = 4$. The divide operation divides the graph into two subgraphs with the same size. The measurement is based on 10 runs for each graph size.

The two algorithms using the repair operator are significant slower then the other ones. The Random-Divide (RM), and the Node-Degree-Divide (NDDa, and NDDb) algorithms are almost equal and linear in computation time. The Random-Subgraph-Divide (RSD) algorithms looks also linear, which is an in an indication that the repair operation is less used in this algorithm then in the Subgraph-Expansion-Divide (SED) algorithm.

Divide	10:90	20:80	30:70	40:60	50:50
RD	1.0 / 0.1	0.5 / 0.1	0.3 / 0.1	0.2 / 0.2	0.2 / 0.2
RSD	100 / 100	100 / 100	100 / 100	100 / 100	100 / 100
NDDa	3.9 / 0.2	2.3 / 0.2	1.6 / 0.2	1.3 / 0.3	1.1 / 0.3
NDDb	4.0 / 0.7	2.5 / 0.8	1.8 / 0.9	1.4 / 1.0	1.1 / 1.2
SED	100 / 100	100 / 100	100 / 100	100 / 100	100 / 100

TABLE V
AVERAGE PART OF EDGES PRESERVED AFTER DIVIDE-OPERATION IN PERCENT

Divide	$\rho_{10:90}$	$\rho_{20:80}$	$\rho_{30:70}$	$\rho_{40:60}$	$\rho_{50:50}$
RD	0.000	0.001	0.001	0.000	0.000
RSD	0.771	0.650	0.632	0.683	0.722
NDDa	0.661	0.541	0.456	0.401	0.385
NDDb	0.813	0.814	0.813	0.814	0.814
SED	0.675	0.568	0.525	0.511	0.516

TABLE VI
MEASURED VALUES FOR SPEARMAN'S ρ

Divide	$\tau_{10:90}$	$\tau_{20:80}$	$\tau_{30:70}$	$\tau_{40:60}$	$\tau_{50:50}$
RD	0.000	0.000	-0.001	0.000	0.002
RSD	0.709	0.558	0.520	0.569	0.612
NDDa	0.561	0.456	0.379	0.330	0.314
NDDb	0.692	0.693	0.692	0.693	0.693
SED	0.558	0.449	0.407	0.394	0.402

TABLE VII
MEASURED VALUES FOR KENDALL'S τ

2) Edge Preservation: The RSD and the SED preserve all structure information from the selected subgraph. The NDD algorithms preserve some structure, $2-4\%$, while the Random Divide loses almost all inner structure. Detailed information are presented in Table V. The first part of each column represents the portion of preserved edges from the first resulting subgraph, and the second portions represents the ratio of the other subgraph.

3) Rank Correlation: Measured rank-correlations of all divides are shown in Table VI and Table VII. Recorded values were averaged over 100 iterations of specified experiments. It can be seen that our baseline algorithm Random-Divide (RD) scores always near 0 and therefor the degree rank order before and after the divide stand in no correlation to each other. Algorithms Node-Degree-Divide-A (NDDa) and Subgraph-Expansion-Divide (SED) scored nearly similar for uneven divides (see $\rho_{10:90}$ and $\tau_{10:90}$). However SED seems to be more resistant to a change of the node-ratio. The rank-correlation values of SED are slowly declining from $\rho_{10:90} = 0.675$ to $\rho_{50:50} = 0.516$ (decrease of $\approx 23\%$). Whereas rank-correlation values of NDDa were decreasing from $\rho_{10:90} = 0.661$ to $\rho_{50:50} = 0.385$ (decrease of $\approx 42\%$). Similar observations can be done comparing values for Kendall's τ. Random-Subgraph-Divide (RSD) was evaluated as second best algorithm. Both correlation coefficients have a high range of values, where distributing nodes in a ratio of 30:70 resulted in minimal values of $\rho_{30:70} = 0.632$ and $\tau_{30:70} = 0.520$. Higher values were reached for more equal divides with a ratio of 50:50 ($\rho_{50:50} = 0.722$ and $\tau_{50:50} = 0.612$) or more uneven divides with a ratio of 10:90 ($\rho_{10:90} = 0.771$ and $\tau_{10:90} = 0.709$). The Node-Degree-Divide-B (NDDb) algorithm scored best with nearly constant values of $\rho \approx 0.813$ and $\tau \approx 0.692$.

4) Node-Degree Distribution: The measured root-mean-squared-error for all dividing algorithms is presented in Table VIII. The Random-Divide Algorithm (RD) as well as the Node-Degree-Divide algorithms (NDDa, and NDDb) generate subgraphs with a lower RMSE then the initial graph. On the contrary the Random-Subgraph-Divide (RSD) algorithm and the Subgraph-Expansion-Divide (SED) method yield into subgraphs with much higher RMSE. Both algorithms use much of the underlying structure and the repair method. That is the reason why the joined node-degree distribution does not fit the theoretical node-degree distribution provides by the Barabási-Albert-model.

RMSE	10:90	20:80	30:70	40:60	50:50
g	16.3	16.4	16.2	16.3	16.5
RD(g_1)	3.8	5.7	7.2	8.9	10.2
RD(g_2)	15.3	14.1	12.8	11.6	10.1
RSD(g_1)	91.8	128.8	190.9	197.0	175.0
RSD(g_2)	309.6	310.8	283.2	234.2	170.6
NDDa(g_1)	3.9	5.7	8.8	8.8	9.8
NDDa(g_2)	15.4	14.1	11.8	11.8	10.1
NDDb(g_1)	3.9	5.5	8.7	8.7	10.4
NDDb(g_2)	15.3	14.2	11.5	11.5	10.1
SED(g_1)	32.0	60.8	93.6	127.9	167.5
SED(g_2)	442.6	464.0	444.4	412.4	374.9

TABLE VIII
RMSE OF NODE-DEGREE DISTRIBUTION BEFORE AND AFTER DIVIDE

Alg.	Run-Time	Edge Preservation	Rank-Correlation	Node-Degree-Distribution
RM	○	-	-	○
NDM	○	-	○	○
PNM	○	○	○	○
MM	○	+	+	-
RD	+	-	-	+
RSD	○	+	○	-
NDDa	+	-	○	+
NDDb	+	-	+	+
SED	-	+	○	-

TABLE IX
OBERVIEW OF MERGE AND DIVIDE ALGORITHMS ($-, \circ, +$)

VI. CONCLUSION

The proposed algorithms were derived from theoretical use cases. It is not possible to determine one best algorithm for either merging or dividing Barabási-Albert-Graphs. Every algorithm has its potential use cases and specific benefits in a subset of our evaluation measures. It is up to the user to decide, which algorithm fits best. A summary of our experimental evaluation results is shown in Table IX and can be used as a guideline.

Results will be included in our tool for event generation of dynamic social network simulations [14] in the next step of development. Further research could be done in favor of the repairing function. With same more advanced repairing steps it would be possible to generate graphs that do fit more the theoretical node-degree distribution.

We provide a Python implementation of the presented algorithms at http://bitbucket.org/paheld/dynamix.

REFERENCES

[1] P. Erdös and A. Rényi, "On random graphs, i," *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959. [Online]. Available: http://www.renyi.hu/~p_erdos/Erdos.html#1959-11

[2] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998. [Online]. Available: http://www.nature.com/nature/journal/v393/n6684/abs/393440a0.html

[3] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[4] R. Albert, H. Jeong, and A.-L. Barabasi, "The diameter of the world wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, Sep. 1999, arXiv:cond-mat/9907038. [Online]. Available: http://arxiv.org/abs/cond-mat/9907038

[5] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Extracting large-scale knowledge bases from the web," in *Proceedings of the 25th VLDB Conference*, 1999, p. 639–650.

[6] A.-L. Barabasi, R. Albert, and H. Jeong, "Mean-field theory for scale-free random networks," *Physica A: Statistical Mechanics and its Applications*, vol. 272, no. 1-2, pp. 173–187, Oct. 1999, arXiv:cond-mat/9907068. [Online]. Available: http://arxiv.org/abs/cond-mat/9907068

[7] W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, pp. 452–473, 1977.

[8] A.-L. Barabási and J. Frangos, *Linked: The New Science Of Networks Science Of Networks*. Basic Books, 2002.

[9] G. Bianconi and A.-l. Barabási, "Competition and multiscaling in evolving networks," *Europhysics Letters*, vol. 54, p. 436–442, May 2001.

[10] D. Zwillinger and S. Kokoska, *CRC standard probability and statistics tables and formulae*. CRC Press, 1999.

[11] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1-2, pp. 81–93, 1938.

[12] P. Held, A. Dockhorn, and R. Kruse, "Generating events for dynamic social network simulations," in *Proceedings of 15th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2014, to be published.