ABSTRACT
Enterprises and researchers often have datasets that can be represented as graphs (e.g., social networks). The owner of a large graph may want to scale it down to a smaller version, e.g., for application development. On the other hand, the owner of a small graph may want to scale it up to a larger version, e.g., to test system scalability. This paper investigates the Graph Scaling Problem (GSP):

Given a directed graph \( G \) and positive integers \( \tilde{n} \) and \( \tilde{m} \), generate a similar directed graph \( \tilde{G} \) with \( \tilde{n} \) nodes and \( \tilde{m} \) edges.

This paper presents a graph scaling algorithm GSCALER for GSP. Analogous to DNA shotgun sequencing, GSCALER decomposes \( G \) into small pieces, scales them, then uses the scaled pieces to construct \( \tilde{G} \). This construction is based on the indegree/outdegree correlation of nodes and edges.

Extensive tests with real graphs show that GSCALER is scalable and, for many graph properties, it generates a \( \tilde{G} \) that has greater similarity to \( G \) than other state-of-the-art solutions, like Stochastic Kronecker Graph and UpSizeR.

1. INTRODUCTION
The emergence of online social networks, like Facebook and Twitter, has attracted considerable research. However, their enormous sizes make any experiment on the entire graph impractical. It is therefore often necessary to obtain a smaller version of the graph for experiments. We call this the scaling down problem.

At the other end of the scale, a new social network service provider may have a small graph, but wants to test the scalability of their system. They may therefore want to have a larger (and necessarily) synthetic version of their current empirical graph. We call this the scaling up problem.

These two problems arise in other contexts as well, e.g., where the graph represents router topology or web links. They illustrate the Graph Scaling Problem (GSP):

Given a directed graph \( G \) and positive integers \( \tilde{n} \) and \( \tilde{m} \), generate a similar directed graph \( \tilde{G} \) with \( \tilde{n} \) nodes and \( \tilde{m} \) edges.

There are many possible ways to define “similarity”, depending on the context, but we believe the definitions must all be in terms of graph properties, like indegree distribution, clustering coefficient, effective diameter, etc.

However, it is impossible for \( G \) and \( \tilde{G} \) to have exactly the same properties; e.g., if \( G \) and \( \tilde{G} \) have the same degree distributions, then the larger graph must necessarily have smaller density. One must therefore select the graph properties that are to be preserved when scaling. GSP facilitates this selection by allowing \( \tilde{n} \) and \( \tilde{m} \) to be specified separately.

Related work in the literature have objectives that are different from GSP. There are many papers on graph sampling, such as gSH, BFS, forest fire and frontier sampling [1, 3, 18, 21, 23, 30, 35]. They can be viewed as examples of scaling down, since they produce a \( \tilde{G} \) that is a subgraph of \( G \); this can have data protection issues that do not arise if \( \tilde{G} \) is synthetic. Moreover, graph sampling cannot generate a \( \tilde{G} \) that is larger than \( G \).

Other related work use generative models that can produce a \( \tilde{G} \) that is smaller or larger than \( G \). For example, an Erdős-Rényi model generates a graph of any size \( n \) with a specified edge probability \( p \) [9]; Chung and Lu’s model generates graphs with a specified degree distribution [4]; and Stochastic Kronecker Graphs [19, 20] are generated from an initiator by applying Kronecker product. However, these do not allow a choice of both \( \tilde{n} \) and \( \tilde{m} \).

In contrast, we propose GSCALER, a solution to GSP that deviates from previous work by using a technique that is analogous to DNA shotgun sequencing [31]. The latter breaks a long DNA strand into smaller ones that are easier to sequence, then use these smaller sequences to reconstruct the sequence in the original strand.

Similarly, GSCALER (i) breaks the given \( G \) into two sets \( S_{in} \) and \( S_{out} \) of small pieces, (ii) scales them by size to \( \tilde{S}_{in} \) and \( \tilde{S}_{out} \); (iii) merges these pieces to give a set \( \tilde{S}_{in} \) of larger pieces, then (iv) assembles \( \tilde{G} \) from the pieces in \( \tilde{S}_{in} \).

This paper makes the following contributions:

1. We present GSCALER, an algorithm for solving GSP.

2. We prove that GSCALER (i) does not generate multiple edges between two nodes, and (ii) has small degree distribution error even when the average degree of \( \tilde{G} \) differs from that of \( G \).

©2016, Copyright is with the authors. Published in Proc. 19th International Conference on Extending Database Technology (EDBT), March 15-18, 2016 - Bordeaux, France: ISBN 978-3-833-8070-7, on OpenProceedings.org. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0
3. We present experiments that compare Gscaler to 4 other techniques, using 2 real graphs and 7 properties.

We begin by surveying related work in Sec. 2. We describe Gscaler in Sec. 3, and prove that it does not generate multiple edges between any two nodes. Sec. 4 reviews the graph properties, state-of-the-art algorithms and datasets that are used for comparison. Sec. 5 then proves that Gscaler has small error, and presents the experimental comparison to other algorithms. We discuss the choice of \( n \) and \( m \) in Sec. 6, before Sec. 7 concludes with a summary.

2. RELATED WORK

The closest work in graph scaling from the literature are graph sampling algorithms and generative models.

For graph sampling, the main approaches are node-based sampling, edge-based sampling and traversal-based sampling which produce a subgraph of the original graph \( G \).

Node-based sampling selects a set of nodes \( V_{sub} \), from \( G \), then \( \tilde{G} \) is just the induced graph of this set of nodes \( V_{sub} \).

Authors in [32] pointed out that node-based sampling may not preserve a power law degree distribution because of bias induced by high degree nodes.

Similarly, traditional edge-based sampling selects edges randomly. However, this might result in a sparsely connected graph \( \tilde{G} [21] \). Some other edge-based sampling variants [1,16,21] sample the graph using edge selection/deletion and combination with node selection/deletion.

Most graph sampling techniques focus on traversal-based sampling [14]. Breadth first sampling (BFS) [3,8,35] and random walk sampling (RW) [12,30] are the most basic and well-known algorithms. Similar to BFS, snow ball sampling [13] (SBS) is widely used in sociological studies. Metropolis-Hastings Random Walk (MHRW) [12,27] is a Markov-Chain Monte Carlo algorithm which samples unbiased subgraph in undirected social graphs. However, MHRW suffers from sample-rejection problem. Later, rejection-controlled Metropolis-Hastings (RCMH) [26] is proposed to reduce the sample-rejection ratio.

Frontier sampling is a multi-dimensional random walk which results in better estimators for some graph properties [30]. A probabilistic version of SBS, forest fire (FF) [21,23] captures some important observations in real social networks, e.g. small diameter.

Most traversal-based sampling requires random access to a node’s neighbors, which might not be feasible for large graphs (that cannot fit into memory). Hence, streaming graph sampling algorithms are proposed, e.g. induced edge sampling (ES-i) [2]. As mentioned previously, graph sampling algorithms are limited to scaling down problem.

For generative models, the Erdős-Rényi model generates a graph of any size \( n \) with a specified edge probability \( p \) [9]. There are variants of random models that generate graphs with specific graph properties [4,28], e.g. the Chung-Lu model generates graphs with a specified degree distribution.

One group of generative models [5,6,11,17] employ the strategy of preferential attachment. They obey a simple rule: a new node \( u \) attaches to the graph at each time step, and adds an edge \( e_{uv} \) to an existing node \( v \) with a probability \( p \) proportional to the degree of the node \( v \).

Another type of generative models is recursive matrix model [7,19,20], which recursively multiplies the adjacency matrix. For example, Stochastic Kronecker Graph (SKG) [20] recursively multiplies the graph initiator \( K_1 \) through Kronecker product, which results in a self-similar graph. SKG captures most social network properties, such as small diameter and power law degree distribution.

Scaling problem appears in other fields as well. For example, UpSizeR is a pioneer tool which synthetically scales a relational dataset [33]. UpSizeR’s focus is on preserving correlation among tuples from multiple tables. In relational terms, GSP requires preservation of correlation among tuples in a single table (for the edges).

For Resource Description Framework (RDF), the AO benchmark [8] is the first tool that scales down an RDF dataset. Later, RBench [29] is proposed to both scale down and up. However, these two benchmarks are evaluated with different metrics (dataset coherence, relationship specialty, literal diversity), so it would be unfair to use them for comparison.

3. GRAPH SCALER (GSCALER)

Given a graph \( G(V,E) \), \( |V| \) and \( |E| \) may need to scale by different factors to maintain similarity for certain properties (e.g. density). Hence, GSCALER allows the user to specify the target \( n \) and \( m \). As shown in Fig.1, the scaling has the following 4 steps:

![Figure 1: The 4 steps in Gscaler.](image)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G(V,E) )</td>
<td>original graph</td>
</tr>
<tr>
<td>( G(V,E) )</td>
<td>scaled graph</td>
</tr>
<tr>
<td>( n/n' )</td>
<td>number of nodes in ( G/G' )</td>
</tr>
<tr>
<td>( m/m' )</td>
<td>number of edges in ( G/G' )</td>
</tr>
<tr>
<td>( f_{in}/f_{in} )</td>
<td>( G/G' )'s indegree distribution</td>
</tr>
<tr>
<td>( f_{out}/f_{out} )</td>
<td>( G/G' )'s outdegree distribution</td>
</tr>
<tr>
<td>( f_{in}/f_{in} )</td>
<td>( G/G' )'s bidegree distribution</td>
</tr>
<tr>
<td>( f_{in}/f_{out} )</td>
<td>( G/G' )'s edge correlation distribution</td>
</tr>
<tr>
<td>( S_{in}/S_{in} )</td>
<td>set of pieces with incoming edges in ( G/G' )</td>
</tr>
<tr>
<td>( S_{out}/S_{out} )</td>
<td>set of pieces with outgoing edges in ( G/G' )</td>
</tr>
<tr>
<td>( S_{in} )</td>
<td>set of pieces with incoming and outgoing edges in ( G/G' )</td>
</tr>
<tr>
<td>( n/\Delta )</td>
<td>count function of the pieces in set ( \Delta ), ( \Delta ) can be ( S_{in}, S_{in} ) and so on.</td>
</tr>
<tr>
<td>( I(\alpha')/O(\alpha') )</td>
<td>total number of available incoming/outgoing edges for nodes with bidegree ( \alpha' )</td>
</tr>
<tr>
<td>( D(f,f) )</td>
<td>KS-D statistics of between ( f ) and ( f )</td>
</tr>
</tbody>
</table>

Table 1: Notation
Algorithm 1: Gscaler(G, \(\tilde{n}, \tilde{m}\))
1 \(S_{in}, S_{out}, f_{bi}, f_{corr} = \text{DECOMPOSE}(G)\)
2 \(\tilde{S}_{in} = \text{SCALING}(S_{in}, \tilde{n}, \tilde{m})\)
3 \(\tilde{S}_{out} = \text{SCALING}(S_{out}, \tilde{n}, \tilde{m})\)
4 \(S_{\text{hy}} = \text{NODE_SYNTHESIS}(S_{in}, S_{out}, f_{bi})\)
5 \(G = \text{EDGE_SYNTHESIS}(S_{bi}, f_{corr})\)

Algorithm 2: SCALING(S_{in}, \(\tilde{n}, \tilde{m}\))
1 node_scaling(S_{in}, \(\tilde{n}\))
2 node_adjustment(S_{in}, \(\tilde{n}, S_{in}\))
3 while \(|\sum x \cdot ct(x) - \tilde{m}| > t\) do
4 \(l = h_0; h = h_0;\)
5 if \(l > h\) then
6 \(l = l_0; h = h_0;\)
7 if \(\tilde{m} > \sum x \cdot ct(x)\) then
8 \(ct(l) > 0\) then
9 \(ct(l) = -1; ct(h) = +1;\)
10 \(l = 4; h = -4;\)
11 else \(l = +1;\)
12 end
13 if \(ct(h) > 0\) then
14 \(ct(l) = +1; ct(h) = -1;\)
15 \(l = 4; h = -4;\)
16 else \(h = -4;\)
17 end
18 \(\text{adjust } \sum x \cdot ct(x) - \tilde{m}\)

pieces, where a piece is a node with its outgoing edges (minus the target nodes). Each node in \(G\) generates 2 pieces, one in \(S_{in}\) and one in \(S_{out}\).

After that, Gscaler scales \(S_{in}\) and \(S_{out}\) to get the scaled sets of pieces \(S_{\text{hy}}\) and \(S_{\text{out}}\), using \(\tilde{n}\) nodes and \(\tilde{m}\) edges.

In node synthesis, \(S_{in}\) and \(S_{out}\) are used to form a set \(S_{\text{hy}}\) of larger pieces. Each new piece has a node with incoming edges (with no source nodes) and outgoing edges (with no target nodes); it will generate one node in \(G\).

The last step (edge synthesis) is to link pieces in \(S_{\text{hy}}\), which finally results in \(\tilde{G}\). Edge synthesis is similar to a jigsaw puzzle, where you want to fit the small pieces together. Algo. 1 summarizes the workflow for Gscaler.

In the following, we will explain each step in detail, and use a running example to show how \(G\) is scaled to \(\tilde{G}\) in Fig. 1. As presented in Fig.1, \(n = 3, \tilde{n} = 6, m = 3, \tilde{m} = 7\).

### 3.1 DECOMPOSE
This step is straightforward, and Fig. 1 shows the pieces in \(S_{in}\) and \(S_{out}\) from decomposing \(G\).

### 3.2 SCALING
To simplify the explanation, we just use \(S_{in}\) for demonstration. Let \(N = \{0, 1, 2, 3, \ldots\}\). We use the count function \(ct\) to denote \(k\) pieces in \(\Delta\) has property \(x\):

\[
ct_{\Delta}(x) = k, \ k \in N
\]

For example, \(ct_{S_{in}}(x) = k\) means \(k\) pieces in \(S_{in}\) have indegree \(x\). For both \(S_{in}\) and \(S_{out}\), the scaling process takes the following three steps:

- **node scaling**. For indegree \(x\), let \(x = ct_{S_{in}}(x) \times \frac{\tilde{n}}{n}\). As \(A_x\) might not be an integer, we round-off \(S_{in}\) as follows:

\[
ct_{\tilde{S}_{in}}(x) = \left\lfloor \frac{A_x}{x} \right\rfloor \text{ with probability } A_x - \left\lfloor A_x \right\rfloor
\]

For example, if \(A_x = 4.8\), then with 0.8 probability, \(ct_{\tilde{S}_{in}}(x) = 5\), and with 0.2 probability, \(ct_{\tilde{S}_{in}}(x) = 4\). Hence, Eq.2 can be rewritten as

\[
E[ct_{\tilde{S}_{in}}(x)] = ct_{S_{in}}(x) \times \frac{\tilde{n}}{n}
\]

Fig. 2 shows the \(\tilde{S}_{in}\) and \(\tilde{S}_{out}\) that we get after this scaling.

- **node adjustment**. With randomness in node scaling (Eq.2), we may have \(|S_{in}| \neq \tilde{n}\). If so, \(|\tilde{n} - |S_{in}||\) nodes

Figure 2: \(\tilde{S}_{in}, \tilde{S}_{out}\) after node scaling for \(G\) in Fig. 1.

with random degree are added to or removed from \(S_{in}\). For \(G\) in Fig. 1, such adjustment is not needed.

- **edge adjustment**. Next, the number of scaled edges must equal to \(\tilde{m}\), i.e. \(\sum x \cdot ct_{\tilde{S}_{in}}(x) = \tilde{m}\). If \(\sum x \cdot ct_{\tilde{S}_{in}}(x) < \tilde{m}\), we increase the number of high degree nodes, and decrease the number of low degree nodes. If \(\sum x \cdot ct_{\tilde{S}_{in}}(x) > \tilde{m}\), we decrease the number of high degree nodes, and increase the number of low degree nodes. The details are shown in Algo. 2.

In our running example, \(\sum x \cdot ct_{\tilde{S}_{in}}(x) = x = 6 < 7 = \tilde{m}\). Hence, we increase the number of high degree nodes (indegree=2), and decrease the number of low degree (indegree=1) nodes. Thus, we have 1 node with indegree=2 and 5 nodes with indegree=1 for \(S_{in}\). After the edge adjustment, the correct \(\tilde{S}_{in}, \tilde{S}_{out}\) are shown in Fig. 1.

### 3.3 NODE SYNTHESIS
Now we have \(\tilde{S}_{in}\) and \(\tilde{S}_{out}\), and we match 1 piece in \(\tilde{S}_{in}\) to 1 piece in \(\tilde{S}_{out}\) and merge them to give a larger piece. This synthesis follows a bidegree distribution \(f_{bi} : N^2 \rightarrow [0, 1]\), where \(f_{bi}(d_1, d_2) = z\) means a fraction \(z\) of nodes have bidegree \((d_1, d_2)\). We say a node \(u\) has bidegree \((d_1, d_2)\) if it has indegree=\(d_1\) and outdegree=\(d_2\). For \(G\) in Fig. 1, the corresponding \(f_{bi}\) is listed in Table 2.

Gscaler loops through \(f_{bi}(d_1, d_2)\) to synthesize nodes. However, for a desired bidegree \((d_1, d_2)\), \(\tilde{S}_{in}\) and \(\tilde{S}_{out}\) may not have the necessary pieces. Hence, a neighboring \((d_1', d_2')\) will be used. Gscaler uses a greedy heuristic that matches pieces by minimizing the Manhattan distance

\[
||((d_1, d_2) - (d_1', d_2'))_1| + |d_2 - d_2'|\]
Algorithm 3: NODE SYNTHESIS($\tilde{S}_{in}, \tilde{S}_{out}, f_{bi}$)

1 while $\tilde{S}_{in}$ and $\tilde{S}_{out}$ not empty do
2   for $f_{bi}(d_1, d_2)$ do
3     while $f_{bi}(d_1, d_2) \times \tilde{m} > 0$ do
4       $(d'_1, d'_2) \leftarrow \text{Manhattan}(d_1, d_2)$
5       $\tilde{S}_{bi} \leftarrow (d'_1, d'_2)$
6     update $\tilde{S}_{in}$, $\tilde{S}_{out}, f_{bi}(d_1, d_2)$

For Table 2, when GSCALER sees $f_{bi}(1, 0) = \frac{1}{2}$, it will first generate 1 node with bidegree $(1, 0)$, and generate the other node with bidegree $(1, 1)$. Next, GSCALER sees $f_{bi}(1, 1) = \frac{1}{2}$, two nodes both having bidegree $(1, 1)$ are generated. Lastly, GSCALER sees $f_{bi}(1, 2) = \frac{1}{4}$, and two nodes with bidegree $(1, 2)$, and $(2, 2)$ are generated.

Algorithm 3 summarizes the node synthesis. The synthesized pieces for $\tilde{S}_{bi}$ in $G$ are shown in Fig. 1. Note that each piece in $\tilde{S}_{in}$ maps to a node in $G$.

### 3.4 Edge Synthesis

Now we are almost done with the graph scaling, we only need to link the edges. This is similar to a jigsaw puzzle, we only need to make sure that each piece links to another correctly. When linking the pieces from $\tilde{S}_{in}$, we link 1 outgoing edge from a source node $v_i \in \tilde{S}_{in}$ to 1 incoming edge from a target node $v_j \in \tilde{S}_{in}$. There are numerous ways of joining the nodes. GSCALER synthesizes edges based on the edge correlation function

$$f_{corr}: N^2 \times N^2 \to [0, 1],$$

where $f_{corr}(a, \alpha)$ = $z$ means a fraction $z$ of the edges have a source node with bidegree $\alpha$ and a target node with bidegree $a$. The $f_{corr}$ for $G$ is listed in Table 3.

Instead of synthesizing edges one by one based on $f_{corr}$ directly, GSCALER undertakes Correlation Function Scaling to scale $f_{corr}$ to $\tilde{f}_{corr}$ for $\tilde{G}$. After a suitable $\tilde{f}_{corr}$ is found, GSCALER links edges based on $\tilde{f}_{corr}$.

#### 3.4.1 Correlation Function Scaling

GSCALER loops through $f_{corr}$ to produce $\tilde{f}_{corr}$. For each $f_{corr}(a, \alpha)\alpha$, it does the following Iterative Correlating:

**Manhattan Minimization**

GSCALER chooses the closest $(\alpha', a')$ for $\tilde{f}_{corr}$ by minimizing $||a - a'|| + ||\alpha - \alpha'||$. For $f_{corr}(1, 2), (1, 0) = 1/3$ in Table 3, GSCALER chooses $(1, 2)$ for $\alpha'$ and $(1, 0)$ for $a'$.

**Increment Probability Maximization**

GSCALER increments $\tilde{f}_{corr}(\alpha', \alpha')\alpha$ by $p$, where $p$ needs to be the largest number that satisfies the following constraints:

**C1.** $p \leq f_{corr}(a, \alpha)\alpha$.  

**C2.** $p \leq \min\{\frac{O(a', \alpha')}{m}, f_{corr}(a', \alpha')\alpha\}$, where $O(a')\alpha$ is the total number of available incoming edges for nodes with bidegree $a'$, and $O(a')\alpha$ is the total number of available outgoing edges for nodes with bidegree $\alpha'$. C2 guarantees incremented number of edges does not exceed the total available number of edges of source nodes and target nodes.

**C3.** $p \leq \frac{ct_{\tilde{S}_{bi}}(a', \alpha') \times \tilde{S}_{bi}(\alpha', \alpha')}{m} - \tilde{f}_{corr}(\alpha', (\alpha', \alpha')\alpha)\alpha$. C3 ensures the total number of edges from source nodes to target nodes is not more than the maximal number of edges allowed from source nodes to target nodes (no multiple edges).

For $f_{corr}(1, 2), (1, 0) = 1/3$ in Table 3:

By **C1**, $p \leq 1/3$.

By **C2**, $p \leq \min\{f_{corr}(1, 2), (1, 0)\alpha\}$.

By **C3**, $p \leq \frac{ct_{\tilde{S}_{bi}}(1, 2) \times \tilde{S}_{bi}(1, 0)}{m} - f_{corr}(1, 2), (1, 0)\alpha\alpha = \frac{1}{3} - 0$.

Hence, the incremental value is $p = \min\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\} = \frac{1}{2}$.

**Value Update**

Next, GSCALER updates the distributions:

- $f_{corr}(a', \alpha')\alpha \leftarrow f_{corr}(a', \alpha')\alpha + p$.
- $O(a')\alpha\leftarrow O(a')\alpha - p \times \tilde{m}$.
- $I(\alpha')\leftarrow I(\alpha')\alpha - p \times \tilde{m}$.

For $f_{corr}(1, 2), (1, 0) = 1/3$ in Table 3, GSCALER gets $f_{corr}(1, 2), (1, 0) = 1/7, O(1, 2) = 1, I(1, 0) = 0$. Table 4 shows the resulting scaled $f_{corr}$.

After iterative correlating, and due to the no multiple edges constraint, it is possible that $\sum f_{corr}(a', \alpha')\alpha < 1$, which we fix by random swapping. This swap first randomly permutes the leftover bidegree from $I$ and $O$ without violating C3, then takes one element with bidegree $\gamma_i^{'}$ from $O$ and one element with bidegree $\gamma_i^{'}$ from $I$ to swap with generated $f_{corr}(a', \alpha')\alpha$.

The idea is to break 1 edge from some source node $v_i$ with bidegree $a'_i$ to some target node $v_j$ with bidegree $\alpha'_i$, and form 2 new edges: 1 edge pointing from some node with bidegree $\gamma_i^{'}$ to the other node with bidegree $\alpha'_i$, and 1 edge pointing from some node with bidegree $a'_i$ to some node with bidegree $\gamma_i^{'}$. If C3 allows, then update $f_{corr}$ as follows:

- $f_{corr}(a', \alpha')\alpha \leftarrow f_{corr}(a', \alpha')\alpha + \frac{1}{m}$.
- $f_{corr}(\gamma_i^{'} \alpha) \leftarrow f_{corr}(\gamma_i^{'} \alpha)\alpha + \frac{1}{m}$.
- $f_{corr}(a', \alpha')\alpha \leftarrow f_{corr}(a', \alpha')\alpha - \frac{1}{m}$.

One successful swap thus increases $f_{corr}$ by $\frac{1}{m}$.

In the worst case (this did not happen in our experiments), after random swaps, $\sum f_{corr}(a', \alpha')\alpha < 1$ might still hold.

We just leave $f_{corr}$ as it is, and we will introduce some dummy nodes to link these $(1 - \sum f_{corr}(a', \alpha')\alpha )\times \tilde{m}$ edges

$$f_{corr}(1, 2), (1, 0) = \frac{1}{7} \quad f_{corr}(1, 2), (1, 1) = \frac{1}{7}$$

$$f_{corr}(1, 1), (1, 2) = \frac{1}{7} \quad f_{corr}(2, 2), (1, 1) = \frac{1}{7}$$

Table 4: Edge Correlation for $\tilde{G}$ in Fig. 1.
Figure 3: Local linking of source and destination.

3.4.2 Edge Linking

After $\tilde{f_{corr}}$ is generated, GSACALER links the $p \times \tilde{m}$ edges locally for each $\tilde{f_{corr}}(\alpha_s', \alpha_t') = p$. Note that some linking steps are related, since $\alpha_s'$ might appear in a series of local linking steps, such as

$$\tilde{f_{corr}}(\alpha_s', \alpha_t_1) = p_1, \ldots, \tilde{f_{corr}}(\alpha_s', \alpha_t_k) = p_k$$

Hence, one must make sure that after $\tilde{f_{corr}}(\alpha_s', \alpha_t_1') = p_1$ is done, $\tilde{f_{corr}}(\alpha_s', \alpha_t_2') = p_2, \ldots, \tilde{f_{corr}}(\alpha_s', \alpha_t_k') = p_k$ are still possible to be linked with no multiple edges. To emphasize the importance of this step, we use the following example to demonstrate both good and bad approaches.

In Fig.3, $\tilde{S}_b$ and $\tilde{f_{corr}}$ are presented at the top. $\tilde{S}_b$ has 2 pieces of bidegree (2, 2), 2 pieces of bidegree (0, 1), and 2 pieces of bidegree (1, 0). The dotted edges represent newly linked edges at each step. If an edge has no source or target node attached, then it has not linked any two nodes yet. Example A demonstrates a failed strategy, while Example B demonstrates a successful strategy.

For $\tilde{f_{corr}}((0, 1), (2, 2)) = \frac{1}{3}$, it corresponds to local linking step $\tilde{J}$, which links two edges from (0, 1) to (2, 2). Both examples link edges successfully. Example A links all 2 edges to 1 target node with bidegree (2, 2), whereas example B links 2 edges to 2 different target nodes with bidegree (2, 2).

For $\tilde{f_{corr}}((2, 2), (1, 0)) = \frac{1}{4}$, it corresponds to local linking step $\tilde{G}$, which links 2 edges from (2, 2) to (1, 0). Both examples link edges successfully. Example A links all 2 edges from 1 source node with bidegree (2, 2). Example B links 2 edges from 2 different source nodes with bidegree (2, 2).

For $\tilde{f_{corr}}((2, 2), (2, 2)) = \frac{1}{4}$, it corresponds to local linking step $\tilde{J}$, which links edges from (2, 2) to (2, 2). Example A produces multiple edges, which is not allowed, whereas example B successfully produces the graph.

Apart from the multiple edge constraint, the algorithm must be efficient as well. The naive idea of back tracking previous edge linking processes is obviously not practical for large graphs. GSACALER not only generates $G$ with no multiple edges, it also links edges in linear time (see below). GSACALER first selects the proper source nodes $L_s$ and target nodes $L_t$, and then link the edges from $L_s$ to $L_t$.

Let $S_{a_s}/T_{a_t}$ be a queue of the source/target nodes with bidegree $\alpha_s'/\alpha_t'$ from $\tilde{S}_b$. For each $\tilde{f_{corr}}(\alpha_s', \alpha_t') = p$, GSACALER dequeue $\tilde{m}$ elements from $S_{a_s}$ to $L_s$, and dequeue $\tilde{m}$ elements from $T_{a_t}$ to $L_t$. More specifically, whenever one element is dequeued from $S_{a_s}/T_{a_t}$, it will be put into $L_s/L_t$, and then been enqueued to $S_{a_s}/T_{a_t}$ again. For example, if we dequeue 7 elements from $S_{a_s}$, we will have $L_s = [1, 2, 3, 4, 5]$ to $L_t$, then $L_t = [1, 2, 3, 4, 5, 1, 2]$, and $\tilde{S}_{a_t} = [3, 4, 5, 1, 2]$. In general,

$$L_s = \{v_{1}^{\tilde{m}}, \ldots, v_{\theta_0}^{\tilde{m}}, u_{\theta_1+1}^{\tilde{m}}, \ldots, u_{\theta_0}^{\tilde{m}}(\alpha_{a_t}')\}, \forall u_i \in S_{a_s},$$

$$L_t = \{v_{\theta_1+1}^{\tilde{m}}, \ldots, v_{\theta_0}^{\tilde{m}(a_t')}, v_{\theta_1+1}^{\tilde{m}}, \ldots, v_{\theta_0}^{\tilde{m}(a_t')}\}, \forall v_j \in \tilde{T}_{a_t}.$$  

where $r_s = p \times \tilde{m}$ mod $|S_{a_s}|$, $r_t = p \times \tilde{m}$ mod $|T_{a_t}|$, (4)

where $r_j$ means $v_j$ appears $k$ times in $L_t$, and $k$ is called the multiplicity of $v_j$.

After $L_s$ and $L_t$ are generated, GSACALER links edges from $L_s$ to $L_t$. Before that, we prove a theorem for Compound Multiplicity Reduction: we later use this to show GSACALER does not generate multiple edges between two nodes.

**Theorem 1. Compound Multiplicity Reduction.** Given multisets $\{u_1^{m_1}, \ldots, u_{\theta_0}^{m_{\theta_0}}\}, \{v_1^{m_1}, \ldots, v_{\theta_1}^{m_{\theta_1}}\}$, such that $U = \bigcup_{u_i \in U} \{u_i^{m_i}\}$, $V = \bigcup_{v_j \in V} \{v_j^{m_j}\}$, (5)

$$\sum_{u_i \in U} m_{u_i} = |U| = |V| = \sum_{v_j \in V} m_{v_j},$$

$$\max_{u_i \in U} m_{u_i} - \min_{u_i \in U} m_{u_i} \leq 1,$$

$$\max_{v_j \in V} m_{v_j} - \min_{v_j \in V} m_{v_j} \leq 1,$$

there exists a multisets $W = \{w_1|w_1(0) \in U, w_1(1) \in V\}$ such that $U = \bigcup_{k=1}^{\theta_0} \{w_k^{m_{w_k}}\}$, $V = \bigcup_{k=1}^{\theta_1} \{w_k^{m_{w_k}}\}$, (8)

$$\max_{w_k \in W} m_{w_k} - \min_{w_k \in W} m_{w_k} \leq 1,$$

$$\forall w_k \in W, \ m_{w_k} \leq \left\lceil \frac{|W|}{\theta_0 \times \theta_1} \right\rceil.$$  

**Proof.** Reorder $U, V$ to

$$U = (u_1, \ldots, u_1, u_2, \ldots, u_2, \ldots, u_{\theta_0}, \ldots, u_{\theta_0}),$$

$$V = (v_1, v_2, \ldots, v_{\theta_1}, v_1, v_2, \ldots, v_{\theta_1}, \ldots).$$

Construct $W$ as follows: $\forall w_k \in W, \ w_k = (U(k), V(k))$, where $U(k)$ means the $k$th element in $U$. Therefore, Eq.8 is satisfied. Further, for any $u_i$, let $W_{u_i} = ((u_i, V(c_i)), (u_i, V(c_i + 1)), \ldots, (u_i, V(d_i)))$ be the sequence of all elements in $W$ containing $u_i$ as first coordinate. Consider $V_{u_i} = (V(c_i), V(c_i + 1), \ldots, V(d_i))$, a sequence of elements in $V$ pairing with $u_i$. Note $V_{u_i}$ is a periodic sequence with period=$\theta$. Thus, the maximum multiplicity in $V_{u_i}$ is $\left\lceil \frac{|W|}{\theta} \right\rceil$, and similarly for $W_{u_i}$. Hence, the maximum multiplicity of $W$ is

$$\max_{w_k \in W} m_{w_k} = \max_{1 \leq k \leq \theta_0} \left\lceil \frac{|U(k)|}{\theta_1} \right\rceil,$$  

(11)

It is trivial that

$$|V_{u_i}| = m_{u_i}, 1 \leq i \leq \theta_0$$  

(12)

Moreover, by Eq.6, we will have

$$\forall u_i \in U, m_{u_i} \leq \left\lceil \frac{|U|}{\theta_0} \right\rceil,$$  

(13)

Hence, by Eq.11, Eq.12, Eq.13, we will have

$$\max_{w_k \in W} m_{w_k} = \max_{1 \leq k \leq \theta_0} \left\lceil \frac{|U(k)|}{\theta_1} \right\rceil \leq \left\lceil \frac{|U|}{\theta_0 \times \theta_1} \right\rceil = \left\lceil \frac{|U|}{\theta_0 \times \theta_1} \right\rceil.$$  

(14)
Since |U| = |W|, therefore
\[
\max_{w_k \in W} m_{w_k} \leq \left\lceil \frac{|W|}{\theta_0 \times \theta_1} \right\rceil,
\]
so Eq.10 holds. Similarly, the minimum multiplicity of W is
\[
\min_{w_k \in W} m_{w_k} = \min_{1 \leq i \leq \infty} \left\lceil \frac{V_{w_k}}{\theta_1} \right\rceil \geq \left\lceil \frac{|W|}{\theta_0 \times \theta_1} \right\rceil,
\]
Eq.9 follows from Eq.15 and Eq.16. We call such a \( W \) a compound multiset.

For edge linking between \( L_s \) and \( L_t \), GSCALER links \( u_i \in L_s \) to \( v_j \in L_t \) to form one edge \((u_i, v_j)\) in \( L_e \) (edge set). By Eq.4, \( L_s/L_t \) satisfies \( U/V \) in Theorem 1 respectively, and it is easy to see that \( L_e \) is the compound \( W \) in Theorem 1.

**Theorem 2.** Given non-empty \( L_s(U) \) dequeued & enqueued from \( S_{s^c} \), and non-empty \( L_t(V) \) dequeued & enqueued from \( T_{a^c} \), the maximum multiplicity for edge set \( L_e(W) \) as described in Theorem 1 is 1.

**Proof.** If \( |L_s| \leq |S_{s^c}| \), then every element in \( L_s \) is unique. Hence, the maximum multiplicity for elements in \( L_s \) is 1, so \( L_s \) has maximum multiplicity of 1. The case is similar for \( |L_t| \leq |T_{a^c}| \).

If \( |L_s| > |S_{s^c}| \) and \( |L_t| > |T_{a^c}| \), then \( L_e \) has \( |S_{s^c}| \) distinct elements, and \( L_e \) has \( |T_{a^c}| \) distinct elements. By Theorem 1, the maximum multiplicity of elements in \( L_e \) is
\[
\left\lceil \frac{|L_s|}{|S_{s^c}| \times |T_{a^c}|} \right\rceil \quad (17)
\]
Since \( |L_s| = |L_e| \), and \( L_e \) has \( \bar{f}_{corr}(\alpha_s', \alpha_t') \times \bar{m} \) elements, the maximum multiplicity of elements in \( L_e \) is
\[
\left\lceil \frac{\bar{f}_{corr}(\alpha_s', \alpha_t') \times \bar{m}}{|S_{s^c}| \times |T_{a^c}|} \right\rceil
\]
Moreover, by C3 in Sec.3.4.1, we know that
\[
\bar{f}_{corr}(\alpha_s', \alpha_t') \times \bar{m} \leq |S_{s^c}| \times |T_{a^c}|,
\]
so
\[
\left\lceil \frac{|L_e|}{|S_{s^c}| \times |T_{a^c}|} \right\rceil = \left\lceil \frac{\bar{f}_{corr}(\alpha_s', \alpha_t') \times \bar{m}}{|S_{s^c}| \times |T_{a^c}|} \right\rceil \leq 1 \quad (19)
\]
Hence, by Theorem 2, there are no multiple edges from \( L_s \) to \( L_t \). Therefore, GSCALER successfully produces a graph without multiple edges.

Moreover, the generation runs in linear time: For each \( \bar{f}_{corr}(\alpha_s', \alpha_t') \), the generation for \( L_s \) and \( L_t \) is in linear time: After \( L_s, L_t \) are generated, the \( L_t \) is formed by sequentially matching the elements in \( L_s \) and \( L_t \) as described in Theorem 1. Hence, the total edge linking time is linear. Given the edge correlation \( \bar{f}_{corr} \) in Table 4, GSCALER generates \( \bar{G} \) as presented in Fig. 1.

Finally, as stated at the end of Sec.3.4.1, there might be some small possibility that \( \sum \bar{f}_{corr}(\alpha_s', \alpha_t') < 1 \) holds. This is the theoretical worst case (which has not happened in our experiments). To resolve this, GSCALER introduces \( \epsilon \) dummy nodes into \( \bar{G} \) for the purpose of maintaining degree distribution similarity. After the dummy nodes are introduced, all the \( 1 - \sum \bar{f}_{corr}(\alpha_s', \alpha_t') \) edges are linked to these dummy nodes. To avoid multiple edges, we can set \( \epsilon = \max_x \{I(a), O(a)\} \); this \( \epsilon \) is obviously not the smallest possible. However, such an \( \epsilon \) is already small (negligible) compared to \( |\bar{G}| \), and such a scenario is rare. Algo.4 summarizes the edge synthesis.

### 4. EVALUATION

We first review the graph properties that we use for similarity measurement.

#### 4.1 Graph Properties

There are numerous graph properties that can be chosen as the similarity measurement criteria, e.g. degree distribution, diameter, k-core distribution, etc. We choose the 7 most common graph properties used in the literature. Let \( N = \{0,1,2,3,...\} \) and consider the following local and global graph properties:

1. **Indegree distribution** \( f_{in} : N \rightarrow [0,1] \)
   \( f_{in}(d) = z \) means a portion \( z \) of nodes have indegree \( d \).

2. **Outdegree distribution** \( f_{out} : N \rightarrow [0,1] \)
   \( f_{out}(d) = z \) means a portion \( z \) of nodes have outdegree \( d \).

3. **Bidegree distribution** \( f_{bi} : N^2 \rightarrow [0,1] \)
   Defined in Sec 3.3.

4. **Ratio of largest strongly connected component (SCC)**
   An SCC of \( G \) is a maximal set of nodes such that for every node pair \( u \) and \( v \), there is a directed path from \( u \) to \( v \) and another from \( v \) to \( u \). The ratio is the number of nodes in the SCC divided by \( |V| \).

5. **Average clustering coefficient (CC)**
   For node \( v_i \), let \( N_i \) be the set of its neighbors. The local clustering coefficient [34] \( C_i \) for nodes \( v_i \) is defined by
   \[
   C_i = \frac{[(v_j, v_k) : v_j, v_k \in N_i, (v_j, v_k) \in E]}{|N_i(|N_i| - 1)}
   \]
   The average clustering coefficient [15] \( \bar{C} = \frac{\sum C_i}{|V|} \).

6. **Average shortest path length (ASPL)**
   For \( u \) and \( v \) in \( V \), the pairwise distance \( d(u, v) \) is the number of edges in the shortest path from \( u \) to \( v \); \( d(u, v) = \infty \) iff there is no path from \( u \) to \( v \) (where \( \infty \) is some number greater than \( |E| \)). The ASPL is
   \[
   \sum_{d(u,v) < \infty} d(u,v) / |V| \times (|V| - 1).
   \]
The effective diameter is the smallest $k \in \mathbb{N}$ that is greater than 90% of all pairwise distances that satisfy $d(u,v) < \infty$.

### 4.2 Measuring Similarity

For a scalar graph property $\alpha$, let $\alpha_G$ denote the $\alpha$ value for $G$ and $\alpha_G^{(i)}$ denote the $\alpha$ value for $\tilde{G}$ constructed with algorithm $A_G^{(i)}$. We can compare $A^{(i)}$ and $A^{(2)}$ by the absolute difference $|\alpha_G^{(i)} - \alpha_G|$ or the relative difference $|\alpha_G^{(i)} - \alpha_G|/\alpha_G$

These are equivalent since

$$|\alpha_G^{(i)} - \alpha_G| < |\alpha_G^{(2)} - \alpha_G| \iff \frac{|\alpha_G^{(i)} - \alpha_G|}{\alpha_G} < \frac{|\alpha_G^{(2)} - \alpha_G|}{\alpha_G}$$

However, an $\alpha$ like effective diameter is an integer, whereas a property like average clustering coefficient has $\alpha < 1$. Some information on $\alpha_G$ is thus lost if we plot relative differences, so we will plot absolute differences instead.

For a degree distribution $f : \mathbb{N}^d \to [0,1]$, we follow Leskovec and Faloutsos [21] and use a Kolmogorov-Smirnov (KS) D-statistic to measure the difference between distributions $f_G$ and $f_{\tilde{G}}$. For $d = 1$, the statistic is defined as

$$\sup_x |F_G(x) - F_{\tilde{G}}(x)|$$

where $F_G$ and $F_{\tilde{G}}$ are the cumulative distribution functions (cdf) for $f_G$ and $f_{\tilde{G}}$. For $d > 1$, defining the cdf is not straightforward, since there are $2^d - 1$ possibilities. In this paper, we adopt Fasano and Franceschini’s computationally efficient variant [10] of the KS D-statistic.

### 4.3 Algorithms

We compare GscaLer to state-of-art algorithms (for graph sampling, generative models, and database scaling) that have been widely used as baselines for comparison.

- In Random Walk with Escaping (RW), a starting node $v_0$ is chosen uniformly at random. Each step in the random walk samples an unvisited neighbor uniformly at random, and there is a probability 0.8 (following [21]) that the walk restarts at $v_0$. If the walk reaches a dead end, the walk restarts with a new $v_0$.

- In Forest Fire (FF) [21], a $v_0$ is similarly chosen. At each step, first choose a positive integer $y$ that is geometrically distributed with mean $p_f(1-p_f)$, and let $x = \lfloor y \times c \rfloor$, $c \in [0,1]$. (We follow Leskovec and Faloutsos and set $p_f = 0.7$ and $c = 1$.) Pick uniformly at random neighbors $w_1, \ldots, w_x$ that are not yet sampled, then recursively repeat these steps at each $w_i$.

- In Stochastic Kronecker Graph (SKG) [20], SKG first trains a $N_1$ by $N_1$ matrix $K_1$, where $N_1$ is typically set as 2. Then recursively multiply $K_1$ through Kronecker product. Thus, $d$ multiplication of Kronecker product results in a graph of $N_1^d$ nodes. In our experiment, we use the $N_1^d$ closest to $\bar{n}$ as the target number $\bar{n}$.

- $UpSizeR$ was designed to synthetically scale a relational dataset by maintaining the degree distribution [33]. However, $UpSizeR$ does not allow free choice of $\bar{m}$. For our experiments, we transform the graph to relational tables so $UpSizeR$ can scale it.

### 4.4 Datasets

We pick 2 real directed graphs from Stanford’s collection of networks [24]. They are large enough that it makes sense for scaling down, but small enough for global properties like diameter to be determined in reasonable time. These two graphs were also used by previous authors [20–22,25].

- Epinions ($|V| = 75879$, $|E| = 508837$) for the website Epinions.com, where an edge $(x,y)$ indicates user $x$ trusts user $y$.

- Slashdot is a website for technology-related news, where users tag others as friend or foe. The graph contains friend/foe links between the users of Slashdot. The dataset Slashdot0811 ($|V| = 77360$, $|E| = 828161$) was from November 2008.

Given the space constraint, we choose to compare more properties for 2 graphs, instead of comparing fewer properties for more graphs.

### 5. RESULTS AND DISCUSSION

All experiments are done on a Linux machine with 128GB memory and AMD Opteron 2.3GHz processor. For SKG, we use the C++ implementation [24]. For UpSizeR, we use the authors’ C++ implementation 1. For FF, RW and GscaLer, we implemented them in Java.

These algorithms use the number of nodes $n$ to specify sample size or scale factor, so we define $s = \bar{n}/n$. In our experiments, we set $s = \frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{10}$, $\frac{1}{30}$, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{30}$, $\frac{1}{50}$, $\frac{1}{75}$, $\frac{1}{100}$, $\frac{1}{150}$, $\frac{1}{250}$ for scaling down, and $s = 2, 3, 4, 5, 6$ for scaling up.

GscaLer allows the user to specify the number of edges $\bar{m}$. However, an $\bar{m}$ that is arbitrarily small or arbitrarily large will make it impossible for $G$ to be similar to $\tilde{G}$. To be fair, we choose an $\bar{m}$ for GscaLer that yields the best results. We will revisit this issue in Sec.6.

For each $s$ value, we run each algorithm 10 times (using different seeds) on each dataset. The average of these 10 runs is then plotted as one data point.

### 5.1 Execution Time

For a fair comparison, we exclude the I/O time for all algorithms.

Execution time for SKG has two parts: training time and running time. SKG needs to train the graph initiator matrix $K_1$, where $K_1$ is a 2 by 2 matrix in our case; $K_1$ can be pre-computed. After $K_1$ is trained, SKG uses $K_1$ to generate a graph with $2^d$ nodes.

To get a better understanding of SKG’s time complexity, we plot both training time and running time. SKG-Train represents the time needed for training $K_1$, while SKG-Run represents the running time of graph generation given $K_1$.

Fig.4 and Fig.5 show the execution time for all algorithms using log scale.

For both datasets, SKG-Train is very large for training the graph initiator $K_1$. However, after $K_1$ is trained, graph generation is fast (seconds), so SKG-Run is small. That aside, GscaLer has the smallest execution time, which is faster than SKG-Train by about 2 orders of magnitude, and faster than RW, FF, UpSizeR by about 1 order of magnitude.

1http://www.comp.nus.edu.sg/~upsizer/
5.2 Theoretical Bounds of Degree Distribution

Before looking at the similarity comparison, we first show theoretically that GSCALER performs well on varying $\tilde{m}$ and $\tilde{n}$ for indegree/outdegree distribution. To save space, we just consider indegree distribution throughout this section. The proof for the outdegree distribution is similar.

**Theorem 3.** Given original graph $G$’s indegree distribution is $f_{in}$, and GSCALER scales $G$ to $\tilde{G}$, where $\tilde{m}$ and $\tilde{n}$ scale by the same ratio $s$. Then, $E[D(f_{in}, \tilde{f}_{in})] = 0$.

**Proof.** As explained in Sec.3.2, GSCALER scales $S_{in}$ with the following criterion:

$$E[ct_{\tilde{S}_{in}}(x)] = ct_{S_{in}}(x) \times \frac{s}{m}$$

Now, consider the current scaled number of nodes $|\tilde{V}|$ and number of edges $|\tilde{E}|$. After node scaling,$\frac{E[|\tilde{V}|]}{E[|\tilde{V}|]} = \frac{\sum_{x} ct_{\tilde{S}_{in}}(x)}{\sum_{x} E[ct_{\tilde{S}_{in}}(x)]} = \frac{\sum_{x} ct_{S_{in}}(x) \times s}{\sum_{x} E[ct_{\tilde{S}_{in}}(x)]} = \frac{n \times s}{\tilde{n}}.

$$E[|\tilde{E}|] = E[\sum_{x} ct_{\tilde{S}_{in}}(x) \times x] = \sum_{x} E[ct_{\tilde{S}_{in}}(x) \times x]

= \sum_{x} x \sum_{x} ct_{S_{in}}(x) \times x \times s

= \sum_{x} ct_{S_{in}}(x) \times x \times s \times m = s \times s = \tilde{m}.

Hence, no edge adjustment is expected in Sec.3.2. Thus, $\forall x, E[\tilde{f}_{in}(x) - f_{in}(x)] = 0$.

Consequently, $E[D(f_{in}, \tilde{f}_{in})] = 0$. \qed

Hence, edge adjustment is needed, as stated in Sec.3.2. In total, we are expecting $m \times s \times (1 + r)$ edges. Hence, $|m \times rs|$ edges are expected to be added/removed. We refer to $ct_{\tilde{S}_{in}}(x)$ as $ct(x)$ if there is no ambiguity.

Consider $r > 0$, so $m \times rs$ edges are to be added. This is done by the edge adjustment operation as stated in Alg.2: (i) $ct(l) = -r$, $ct(h) = +r$, (ii) $l + r$, $h - r$. The net effect is to add $r$ edges per adjustment.

Let $d_m$ be the maximum degree of $G$. We assume $l$ is initiated as 0, while $h$ is initialized as $d_m$.

**Case ct(l) > 0 for all the first k adjustments:**
Then GSCALER will add $d_m, d_m - 2, d_m - 4, \ldots, d_m - [(k - 1) \mod \frac{d_m}{2}] \times 2$ (20) edges for the first $k$ adjustments. Let $T_k$ be the total number of edges added by first $k$ adjustments, then

$$T_k \geq \frac{d_m}{2} \times k$$

Since $m \times rs$ edges are expected to be added, the expected number of adjustments $k$ satisfies

$$k \leq \frac{m \times rs}{\frac{d_m}{2}} = \frac{m \times rs}{d_m}$$

Moreover, each edge adjustment changes $\tilde{f}_{in}$ by (i) decrementing $\frac{1}{n}$ for some $f_{in}(x)$, where $x < \frac{d_m}{2}$
(ii) incrementing $\frac{1}{n}$ for some $f_{in}(x)$, where $x > \frac{d_m}{2}$

Thus, the expected total decremental changes made to $\tilde{f}_{in}$ is $\frac{k}{n} \times k$. By Eq.22,

$$\frac{1}{n} \times k \leq \frac{1}{n} \times \frac{2m \times rs}{d_m} = \frac{2m \times rs}{m \times d_m} = \frac{2mr}{n \times d_m}$$

Since $D(f_{corr}, \tilde{f}_{corr})$ measures largest difference between the cumulative function of $f_{corr}, \tilde{f}_{corr}$. Hence, by Eq.23,

$$E[D(f_{corr}, \tilde{f}_{corr})] \leq \frac{2mr}{n \times d^* \times d^*} \times d^* = \frac{d_m}{2} \times i$$

**Theorem 4.** Given an original graph $G$’s indegree distribution is $f_{in}$, and GSCALER scales $G$ to $\tilde{G}$, where $\tilde{n} = n \times s$, $\tilde{m} = m \times s \times (1 + r)$ and $r \neq 0$. Then,

$$E[D(f_{in}, \tilde{f}_{in})] \leq \frac{2m|r|}{n \times d^*}$$

where $d^*$ is approximately the largest degree of $G$.

**Proof.** As shown in the proof of Theorem 3, after node scaling, $E[|\tilde{V}|] = n \times s = \tilde{n}$ and $E[|\tilde{E}|] = m \times s = \tilde{m}$. 

Figure 4: Execution time (log scale) for Slashdot. FF and RW do not work for $s > 1$.

Figure 5: Execution time (log scale) for Epinions. FF and RW do not work for $s > 1$. 

Before looking at the similarity comparison, we first show theoretically that GSCALER performs well on varying $\tilde{m}$ and $\tilde{n}$ for indegree/outdegree distribution. To save space, we just consider indegree distribution throughout this section. The proof for the outdegree distribution is similar.
Then, $T_i \geq \frac{d_m}{2} \times i - 1 = \frac{d_{m-1}}{2} \times i + (\frac{i}{2} - 1)$
Since $T_1 = d_{m-1} \geq \frac{d_{m-1}}{2}$, then $T_i \geq \frac{d_{m-1}}{2} \times i \forall i \in N$
2. $l_0 + 1 = h$: Then $l$ will be reset to 0, and $h$ will be reset to $d_m$, so the number of edges at $i$th adjustment is obviously larger than $d_m - \lfloor (i-1) \mod \lfloor \frac{d_m}{2} \rfloor \rfloor \times 2 - 1$. Hence, Eq. 25 will hold as well.

Therefore, during the first $k$ adjustments, if we encounter $w$ different $l_0, l_1, \ldots, l_{w-1}$, such that $c(l_j) = 0, 0 \leq j < w$, then by mathematical induction, one can still conclude that

$$T_i \geq \frac{d_m - w}{2} \times i.$$ 

Hence, let $d^* = d_m - w$ and follow the proof after Eq. 21; then $E[D(f_{corr}, f_{corr})] \leq \frac{n}{2 \sqrt{d^*}}$, where $d^*$ is close to $d_m$ in general.

The proof for $r < 0$ is similar \( \square \)

### 5.3 Experimental Results

All figures in this section (except Figs. 7, 9, 11, 12) plot the similarity measures (Sec. 4.2) comparing $G$ and $\tilde{G}$, where

- the horizontal axis is scale factor $s = \frac{m}{n}$;
- the vertical axis is KS D-statistic for indegree, outdegree and bidegree distributions;
- the vertical axis is absolute error for the other 5 properties (effective diameter, largest SCC ratio, etc.);
- the true value for all datasets’ graph properties are provided above each figure;
- we choose $\tilde{m}$ to give best results for GSCALER; the other algorithms do not allow a free choice of $\tilde{m}$.

We first look at scaling down for all the algorithms. As mentioned in Sec. 1, scaling down in GSP has an objective that is different from graph sampling. However, we use graph sampling algorithms for comparison because GSP is a new problem, so there is no other algorithms for comparison except UpSizeR and SKG.

5.3.1 Indegree Distribution

For the indegree distribution, Fig. 6 shows that both GSCALER and UpSizeR perform very well for Slashdot, with error $< 0.01$. For Epinions, UpSizeR has an error of 0.05 on average, whereas GSCALER again has a small error $< 0.01$. For more details, Fig. 7 plots the indegree distribution for $s = 0.2$, where the $x$-axis is the indegree, and the $y$ axis is the ratio of nodes having that indegree. We only show the plot up to indegree $= 15$, which covers more than 87% of all nodes.

The plot shows that GSCALER and UpSizeR mimics $G$’s indegree distribution very well. Although all algorithms produce power law shaped distributions, only UpSizeR and GSCALER give a close fit for the empirical distribution.

5.3.2 Outdegree Distribution

For the outdegree distribution, Fig. 8 similarly shows that both GSCALER and UpSizeR perform very well for Slashdot, with error $< 0.03$ on average. For Epinions, UpSizeR has an error of 0.05 on average, whereas GSCALER still has an error $< 0.01$. The outdegree distributions are plotted in Fig. 9. We observe that GSCALER and UpSizeR also closely match $G$’s outdegree distribution. Again, although all algorithms produce power law shaped distributions, UpSizeR and GSCALER give the best fit for the empirical outdegree distribution.

5.3.3 Bidegree Distribution

5.4 Conclusion

In summary, we have shown that scaling down in GSP has a clear objective different from graph sampling. We have introduced GSCALER, a new algorithm for scaling down in GSP. We have shown that GSCALER performs very well with small error and is close to the true value in the majority of cases. Additionally, we have compared GSCALER to other algorithms and shown that for all datasets, GSCALER provides a good fit for the empirical distribution of each property. For more details, please refer to our technical report. Moreover, we have shown that GSCALER is scalable and efficient, making it an attractive choice for graph scaling applications.

Figure 6: KS-D statistics for Indegree Distribution

Figure 7: Indegree Distribution Plot

Figure 8: KS-D statistics for Outdegree Distribution

Figure 9: Outdegree Distribution Plot

Figure 10: KS-D statistics for Bidegree Distribution
Even though UpSizeR matches the empirical indegree and outdegree distributions, it is not able to capture the bidegree distribution that describes the correlation between indegree and outdegree. Such correlation is especially important for social network graphs. For example, in Epinions, the number of people he/she trusts and the number of people who trust him/her are correlated. As shown in Fig. 10, only Gscaler captures such correlation very well.

For a detailed look at the bidegree distributions, we give 2-dimensional plots Fig. 11 and Fig. 12. We transform the bidegree distribution to a \( k \times k \) matrix \( B \). Each cell \( i, j \) represents the portion of nodes with bidegree \( (i, j) \). In other words, \( B[i, j] = f_{bi}(i, j) \). When visualizing \( B \), we use a gray scale intensity plot for cell \( i, j \) to indicate \( B[i, j] \). The larger \( f_{bi} \) is, the darker the cell \( (i, j) \) is. In our case, we set \( k = 50 \) which covers more than 90% of total nodes.

Fig. 11 and Fig. 12 show indegree is positively correlated to outdegree. For both Slashdot and Epinions, Gscaler is the best algorithm in capturing this indegree/outdegree correlation. We also observe that UpSizeR tends to have indegree and outdegree negatively correlated. SKG has very concentrated and similar-shaped plots for both datasets. We suspect this is because of the self-similar matrix operation, Kronecker product. For FF, it captures the bidegree correlation for Slashdot’s bidegree, but not for Epinions.

5.3.4 Effective Diameter
Fig. 13 shows that, for Slashdot, Gscaler produces exactly the real effective diameter; for Epinions, it produces an effective diameter with an absolute error no larger than 1. Overall, Gscaler, FF, SKG are the best algorithms in producing similar effective diameters.

5.3.5 Average Clustering Coefficient
For both datasets, Gscaler significantly reduces the error for average clustering coefficient. Fig. 14 shows that, for Slashdot, the average error for the other algorithms are between 0.03 and 0.045. This corresponds to a relative error of 60% to 90%. However, Gscaler only has an absolute error 0.005 on average, which corresponds to a relative error < 10%. Similarly, for Epinions, Gscaler also improves the relative error from 70% to 10% on average.

5.3.6 Largest SCC Ratio
Fig. 15 shows that, for Slashdot, Gscaler produces exactly the real SCC ratio; for Epinions, it produces a SCC ratio with an absolute error no larger than 50%. Overall, Gscaler, FF, SKG are the best algorithms in producing similar SCC ratios.

Figure 16: Absolute Error for ASPL
For largest SCC ratio, Fig. 15 shows that FF, UpSizeR, and Gscaler have the best performance for Slashdot. For Epinions, SKG is the best performing algorithm, whereas Gscaler only loses to SKG by an error < 0.01.

Note that, Gscaler and FF are the best performing algorithms which produce best similar largest SCC ratio for both datasets on average.

5.3.7 Average Shortest Path Length

Fig. 16 shows that Gscaler is the best algorithm in producing similar ASPL for the scaled graph.

For Slashdot, Gscaler has relative error < 10%, whereas the second best performing algorithm UpSizeR has the average relative error 40%

For Epinions, Gscaler is the most stable and accurate algorithm, with a consistent absolute error < 0.25. RW performs well for large s, but does badly for small s.

5.3.8 Scaling Up

To save space, Fig. 17 plots performance of both Slashdot and Epinions for each graph property.

Similar to scaling down, Gscaler improves the accuracy of indegree/outdegree/bidegree distribution significantly. For effective diameter, Gscaler and SKG are the best performing algorithms. Gscaler is the best algorithm which produces the most accurate results for average clustering coefficient, largest SCC ratio, and average shortest path length.

5.3.9 Summary of Comparisons

For indegree, outdegree and bidegree distributions, Gscaler reduces the error from about 0.1 for the other algorithms to about 0.01. This is expected since Gscaler uses $f_{bi}$ to construct $\tilde{G}$, and agrees with the theoretical bound in Sec. 5.2.

Gscaler only uses $P_{csv} = \{f_{bi}, f_{err}\}$ to construct $\tilde{G}$, but measures similarity to $G$ with a larger set $P$ of both local and global properties listed in Sec. 4.1. The results show that enforcing $P_{csv}$ suffices to induce similarity for $P$.

6. LIMITATION

Unlike the other algorithms, a user can choose both $\tilde{n}$ and $\tilde{m}$ for Gscaler. Table 5 illustrates Gscaler’s accuracy for Slashdot, using different $\tilde{n}$ and $\tilde{m}$.

<table>
<thead>
<tr>
<th>$\tilde{n}$</th>
<th>$\tilde{m}$</th>
<th>Effective Diameter</th>
<th>Largest SCC</th>
<th>ASPL</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 77360$</td>
<td>$m = 828161$</td>
<td>5</td>
<td>0.909</td>
<td>3.74</td>
<td>0.052</td>
</tr>
<tr>
<td>$\tilde{n} = 4421$</td>
<td>$\tilde{m} = 32179$</td>
<td>5</td>
<td>0.916</td>
<td>3.59</td>
<td>0.058</td>
</tr>
<tr>
<td>$\tilde{n} = 4421$</td>
<td>$\tilde{m} = 33831$</td>
<td>5</td>
<td>0.916</td>
<td>3.52</td>
<td>0.053</td>
</tr>
<tr>
<td>$\tilde{n} = 4421$</td>
<td>$\tilde{m} = 34399$</td>
<td>5</td>
<td>0.916</td>
<td>3.62</td>
<td>0.058</td>
</tr>
<tr>
<td>$n = 15472$</td>
<td>$m = 141583$</td>
<td>5</td>
<td>0.916</td>
<td>3.57</td>
<td>0.053</td>
</tr>
<tr>
<td>$\tilde{n} = 15472$</td>
<td>$\tilde{m} = 144895$</td>
<td>5</td>
<td>0.916</td>
<td>3.51</td>
<td>0.060</td>
</tr>
<tr>
<td>$\tilde{n} = 15472$</td>
<td>$\tilde{m} = 147839$</td>
<td>5</td>
<td>0.916</td>
<td>3.45</td>
<td>0.059</td>
</tr>
<tr>
<td>$n = 154720$</td>
<td>$m = 1609003$</td>
<td>5</td>
<td>0.917</td>
<td>3.71</td>
<td>0.055</td>
</tr>
<tr>
<td>$\tilde{n} = 154720$</td>
<td>$\tilde{m} = 1671288$</td>
<td>5</td>
<td>0.917</td>
<td>3.71</td>
<td>0.062</td>
</tr>
<tr>
<td>$\tilde{n} = 154720$</td>
<td>$\tilde{m} = 1672057$</td>
<td>5</td>
<td>0.917</td>
<td>3.70</td>
<td>0.062</td>
</tr>
<tr>
<td>$n = 386800$</td>
<td>$m = 4182213$</td>
<td>5</td>
<td>0.917</td>
<td>3.89</td>
<td>0.058</td>
</tr>
<tr>
<td>$\tilde{n} = 386800$</td>
<td>$\tilde{m} = 4186353$</td>
<td>5</td>
<td>0.917</td>
<td>3.89</td>
<td>0.048</td>
</tr>
<tr>
<td>$\tilde{n} = 386800$</td>
<td>$\tilde{m} = 4190908$</td>
<td>5</td>
<td>0.917</td>
<td>3.89</td>
<td>0.053</td>
</tr>
</tbody>
</table>

Table 5: Gscaler accuracy for different $\tilde{n}$ and $\tilde{m}$.

7. CONCLUSION

We considered the problem of synthetically scaling a given graph. Our solution Gscaler first breaks $G$ into pieces, scales them, then merges them using the degree and correlation functions from $G$.

Different from previous approaches, Gscaler gives user a choice for $\tilde{n}$ and $\tilde{m}$. We proved that Gscaler does not produce multiple edges between two nodes, and has a small distribution error even when the average degree of $\tilde{G}$ differs from the original graph $G$.

Experiments with 2 well-known real datasets show that the $\tilde{G}$ constructed by Gscaler is more similar to $G$ for most properties than random walk, forest fire, UpSizeR and Stochastic Kronecker Graph.

Our current work aims to extend Gscaler to scale relational databases by representing the tables as graphs.

8. REFERENCES


the Hungarian Academy of Science, pages 17–61, 1960.

Figure 17: Scaling Up Experiments (s values on horizontal axes; legend format is dataset-algorithm.)