Learning and Scaling Directed Networks via Graph Embedding

Mikhail Drobyshevskiy[1] Anton Korshunov[1] Denis Turdakov[1]

Institute for system programming of Russian Academy of Sciences, Moscow, Russia
{drobyshevsky,korshunov,turdakov}@ispras.ru

Abstract. Reliable evaluation of network mining tools implies significance and scalability testing. This is usually achieved by picking several graphs of various size from different domains. However, graph properties and thus evaluation results could be dramatically different from one domain to another. Hence the necessity of aggregating results over a multitude of graphs within each domain.
The paper introduces an approach to automatically learn features of a directed graph from any domain and generate similar graphs while scaling input graph size with a real-valued factor. Generating multiple graphs with similar size allows significance testing, while scaling graph size makes scalability evaluation possible. The proposed method relies on embedding an input graph into low-dimensional space, thus encoding graph features in a set of node vectors. Edge weights and node communities could be imitated as well in optional steps.
We demonstrate that embedding-based approach ensures variability of synthetic graphs while keeping degree and subgraphs distributions close to the original graphs. Therefore, the method could make significance and scalability testing of network algorithms more reliable without the need to collect additional data. We also show that embedding-based approach preserves various features in generated graphs which can't be achieved by other generators imitating a given graph.

Keywords: Random graph generating ·Graph embedding ·Representation learning

1 Introduction

Modeling and generating random graphs is an actively evolving research area. Theoretical aspects include studying features and processes defining connectivity structure in real graphs with mathematical methods. There are also important practical use-cases where random graphs are traditionally employed. A set of random graphs with similar properties could be used for testing significance of results of network mining tools, e.g. community detection. If size of generated graphs is adjustable, scalability of graph algorithms could be tested as well.
According to our experience, the pipeline of modeling and generating random graphs resembling properties of real data includes the following steps:

1. learn statistical features of real graphs: distributions, dependencies, parameter ranges, etc;
2. select features to be modeled: degree distribution, clustering, diameter, subgraphs distribution, etc;
3. define a probability space over all possible graphs with selected features (usually achieved by defining parametrized generative process for graphs);
4. sample random graphs from the defined space.

The fundamental issue here is that each graph domain (social, mobile [20], biological [23], etc) has its own specific features and many of them may be unknown. Consequently, random graph models created for one domain could be invalid for others. And generally one can't be sure that all essential features of real graph are modeled.

Uncertainty about properties of an arbitrary graph leads to the idea of automatic extraction of features from real data. Suchwise, steps 1-3 of the aforementioned pipeline are replaced with automatic model learning for the given graph. The model could be complicated to be studied analytically and could hardly provide new insights about the data. However, practical use-cases of random graphs could be improved and extended: e.g., data anonymization for publishing a synthetic version of real network preserving its information privacy.

The most popular technique of automatic feature extraction from graphs is representation learning. In recent years this area has attracted much attention in view of recent success in word embedding [18] and adaptation of these ideas to graph domain [24], [26].

Our approach incorporates graph embedding into random graph generation. Embedding result is a set of node vectors which altogether encode some statistical features of the input graph. Experiments suggest that all (or almost all) edges of the input graph could be recovered from the node vectors given proper embedding scheme.

Another advantage of using embedding in context of graph generation is possibility to approximate node vectors distribution. This allows to utilize a unified sampling scheme for random graphs, regardless of input graph domain and features. Finally, arbitrary number of node vectors could be sampled for each synthetic graph, allowing to precisely control resulting graph size.

Our main contributions are as follows:

- We present a pipeline of generating controllable size random graphs similar to a given one based on graph embedding (Embedding-based Random Graph Generator, ERGG).
- We develop an embedding method such that an arbitrary directed graph can be recovered from its embedding with small distortion.
- Within ERGG pipeline we develop ERGG-dwc – a concrete algorithm based on this embedding method which handles $\underline{d}$irected $\underline{w}$eighted graphs with $\underline{c}$ommunity structure[1].
- We show that ERGG-dwc preserves subgraph distribution, clustering, shortest path length distribution, and other properties in generated graphs which can't be achieved by other RGGs imitating real graphs.

[1]Online demo: `http://ergg.at.ispras.ru/`

Many graph domains are directed naturally (e.g. mobile call graphs), while edge weights contain additional information about the object modelled by the graph (e.g. duration of call). Furthermore, many graphs have community structure which determines high-level organization of a network into groups of nodes sharing similar function, property, role, etc. That's why ERGG-dwc targets edge direction and weight along with community structure of nodes.
Important aspect of task definition is how we define similarity between graphs, especially for the case of different graph sizes. We believe that any fixed set of features could be incomplete. Still, we selected two distributional features which seem to cover all levels of network organisation. We require these distributions in the generated graphs to be close to those of a given graph.
Node degree distribution is an important global characteristic of graph. For instance, many complex networks demonstrate power-law degree distribution. The form of degree distribution is known to determine some other graph characteristics: for instance, power-law distribution determines small **diameter** [5], **clustering coefficient** as a function of node degree [6], etc.
Subgraph distribution is another informative topological characteristic of a graph which defines node clustering behavior and other features. For instance, it allows to categorize graphs over their domains with better precision than other features [2].
The rest of the paper is organized as follows. In Sect.2 we survey works in related areas, then in Sect.3 we introduce our ERGG pipeline together with ERGG-dwc algorithm. In Sect.4 we provide a detailed research of ERGG-dwc steps, and then in Sect.5 we experimentally evaluate them and perform other tests. Finally, we discuss main results and give a conclusion in Sect.6.

Notation

$G = (N, E)$ — graph with nodes N and edges $E \subseteq N \times N$, $n = |N|$ — number of nodes, $e = |E|$ — number of edges;
$H = (M, F)$ — generated graph with nodes M and edges F;
i, j, k, l — nodes, (i, j), $i \to j$ — edges;
$deg(i)$ – degree of node i, $deg_+(i)$ – out-degree of node i;
w_{ij} — weight of edge $i \to j$;
n-GP — graph profile, L^2-normalized vector of counts of all possible (up to isomorphism) connected subgraphs with n nodes in a graph (see [2] for details). For example, 3-GP is a vector of 13 numbers which reflects distribution of subgraphs with 3 nodes in a graph (see Fig.1).

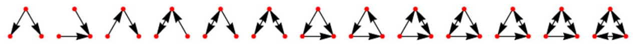
Fig. 1: 13 possible connected directed subgraphs with 3 nodes, up to isomorphism.

2 Related Work

2.1 Random graph generation

A lot of random graph models were suggested in order to reflect particular properties discovered in real networks: power law degree distribution, small diameter, high clustering coefficient and others. Such models include Erdös-Rènyi model [8], Watts-Strogatz model [27], Barabàsi-Albert model [1], R-MAT [3], Kronecker graphs [14], MFNG [22], dot-product graphs [31], hyperbolic geometric graph [11], etc.
Some models are designed to generate graphs with community structure: stochastic block model [19], LFR [12], CKB [4], ReCoN [25]. Several models aim at producing specified subgraph distribution: triplet model [28], STS [29]. A few models allow specifying of graph features to be produced: Feature constraints method [30], MFNG [22].
To the best of our knowledge, there is no method for generating controlled size directed weighted graphs with communities, capable of automatically reproducing important statistical properties of a given graph, namely degree and subgraph distribution (see table 1).

2.2 Directed graph embedding

The task of mapping nodes of a particular graph into real-valued vectors of some low-dimensional space with preserving useful features of this graph is referred to as graph *embedding* or *representation learning*. In view of recent success of graph embeddings based on machine learning techniques [24], [10], [26] we concentrate on them.

BLM Bilinear link model (BLM) [10] uses the following model for directed graphs:

$$p(j|i) = \frac{\exp(\boldsymbol{u}_i \cdot \boldsymbol{v}_j)}{\sum_k \exp(\boldsymbol{u}_i \cdot \boldsymbol{v}_k)} \tag{1}$$

Here each node $i \in N$ is associated with input and output node vectors: $\boldsymbol{u}_i, \boldsymbol{v}_i$. With joint link probability $p(i,j) = p(i)p(j|i)$, the objective is a log-likelihood of the whole graph:

$$J_\Theta = \sum_{(i,j)\in E} \log p(i,j) \to \max_\Theta \tag{2}$$

For softmax approximating authors implement Noise contrastive estimation (NCE) [9], which was developed for estimation of unnormalized probabilistic models, treating the normalizing constant Z_i as an additional parameter.

LINE A similar approach for embedding of directed weighted graph was suggested in [26]. Along with (1) which is called 2-nd order proximity, authors also maximize 1-st order proximity of node pairs $p_1(i,j) = \frac{1}{1+\exp(-\boldsymbol{u}_i \cdot \boldsymbol{u}_j)}$. To overcome a large summation in denominator negative sampling (NEG) [18] is used, which is a simplification of NCE.

Table 1: Comparison of supported properties for several RGG algorithms. Properties notation: **learn** – how features are extracted from a graph; **dir**, **wei**, **com** – support of directed, weighted graphs and communities; **DD** – degree distribution; **size** – support of controllable size of generated graphs; **complexity** – complexity of graph generation. Values notation: ' ' – not supported, '+' – supported (specified to a model); '+-' – partially supported; '=' – exactly reproduced; '≈' – approximately reproduced; '*' – supported in theory, but unchecked in practice.

algorithm	**learn**	**dir**	**wei**	**com**	**DD**	**3-GP**	**size**	**complexity**
Feature constraints	manual	+			=	*	=	$O(\|E\|)$
SKG	auto	+			≈		+-	$O(\|E\| \log \|N\|)$
MFNG	manual	+			≈	*	+	$O(\|E\| \log \|N\|)$
ReCoN	auto	+		=	=		+-	$O(\|E\|)$
Dot product	no	+			+-		+	$O(\|N\|^2)$

Finally we note that although graph embedding is widely used for learning graph features no one has yet applied it to graph generation. Of our interest are such embedding methods of a directed graph that would 1) encode most of its properties in the representation, and 2) allow to reconstruct from it graphs of various size with these properties preserved.

3 Embedding Based Graph Generating

Now we present the general pipeline of our Embedding-based Random Graph Generator (ERGG) for generation of controllable size random graphs similar to a given one. Then we suggest ERGG-dwc algorithm tailored for directed weighted graphs with community structure.

3.1 General pipeline of ERGG

The nodes of an original graph are first embedded into vectors in low dimensional space $\mathbb{R}^d$. Then new vectors corresponding to new nodes are sampled from some probability distribution which approximates the distribution of node vectors. Finally, new nodes are connected with edges giving a new graph.
More formally, ERGG input is a graph $G = (N, E)$ and scaling factor $x > 0$ ($x \in \mathbb{R}$). The output is a new random graph $H = (M, F)$ with $|M| \approx \lfloor x|N| \rfloor$ nodes. The steps are as follows:

1. Embed graph $G = (N, E)$ into low-dimensional space, such that its nodes $i \in N$ are mapped into real value vectors $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$.
2. Approximate the distribution of vectors $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$ and sample a new set of $\lfloor x|N| \rfloor$ random vectors $\{\boldsymbol{q}_i\}_{i=1}^{|M|}$ from this distribution. These vectors will correspond to nodes of a new graph $(M, \cdot)$.
3. Connect nodes of graph $(M, \cdot)$ with edges using the embedding model from step 1, obtaining a result graph $H = (M, F)$.

We assume that at step 1 one may use any embedding method providing for a pair of nodes i, j a score function $s_{ij} = s(\boldsymbol{r}_i, \boldsymbol{r}_j)$, characterizing a link $i \to j$. Since an input graph G is embedded and the distribution of its node vectors $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$ is approximated by some distribution model $\mathcal{R}$, we get a generative model which defines a probability distribution over graphs similar to G. To sample such a random graph one should sample M vectors from $\mathcal{R}$ and use s_{ij} function to connect corresponding nodes. Note that this pipeline covers both directed and undirected graphs and allows for extending it to graphs with weights and/or communities.

3.2 ERGG-dwc — algorithm for scaling a directed weighted graph with communities

Now we suggest ERGG-dwc, a concrete algorithm capable of handling graphs with weights and communities. Weights are treated as edge labels, while communities — as node labels. Suppose an input is a directed weighted graph $G = (N, E)$ with community structure given as node labelling $\{\mathcal{C}_i\}_{i=1}^{|N|}$ and a scaling factor x. Additional parameters are noise magnitude ϵ and default edge weight w_0. Steps to be performed are the following:

1. **Embedding**. Embed graph $G = (N, E)$ with a modified embedding method (see 4.1) into node vectors $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$; and find a threshold t_G which discriminates top E node pairs (i, j) with relatively higher score $s(\boldsymbol{r}_i, \boldsymbol{r}_j)$ from the rest of all node pairs $i, j \in N$. Threshold t_G is utilized in edge generation process during the connecting step.
2. **Approximating + sampling**. Randomly sample (with repetitions) $m = \lfloor x|N| \rfloor$ vectors $\{\boldsymbol{q}_i\}_{i=1}^{m}$ from $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$ adding small gaussian noise $\boldsymbol{g} \sim \mathcal{N}(0, diag(\epsilon, ..., \epsilon))$. A mapping φ from nodes of original graph to a new graph is thus defined: $N \overset{\varphi}{\mapsto} M$.
3. **Connecting**. Connect with edges those pairs of nodes k, l from M that have $s(\boldsymbol{q}_k, \boldsymbol{q}_l) > t_G$. Dangling nodes if present are removed, obtaining graph $H = (M', F)$.
4. **Attributing**.
   (a) For each edge $(k, l) \in F$, assign weight $w'_{kl} = w_{ij}$, where $i = \varphi^{-1}(k)$, $j = \varphi^{-1}(l)$. If $(i, j) \notin E$, pick a default edge weight w_0.
   (b) For each node $k \in M$, assign community labels $\mathcal{C}'_k = \mathcal{C}_i$, where $i = \varphi^{-1}(k)$.

4 Detailed Algorithm Description

Here we elaborate on ERGG-dwc steps described in Sect.3.1. Instead of solving the whole ERGG-dwc task we have splitted it in a sequence of simpler subtasks and conducted a research of their possible solutions, optimizing a separate objective for each one. Namely, these subtasks are: embedding + connecting 4.1, approximating + sampling 4.2 and attributing 4.3.

4.1 Embedding + connecting

The first task is to embed nodes of a given graph into vectors preserving maximal information. In order to achieve this we try to recover edges of the same graph back from the vectors and maximize F_1 score of restored edges versus existed edges. We will refer to it as edge-recovery F_1 or simply F_1 further. Thereby Embedding step is coupled with Connecting step.

Graph reconstructing. We assume that embedding method provides a score function $s_{ij} = s(\boldsymbol{r}_i, \boldsymbol{r}_j)$ on pairs of nodes, which is trained to score edges of the graph higher than non-edges. Then we just score all node pairs and choose a threshold t_G equal to s_{ij} with rank $E+1$ and therefore all pairs with $s_{ij} > t_G$ become edges.

Modified embedding method. We have elaborated BLM and LINE algorithms according to our goal of separating edges from non-edges, and have suggested a new COMBO algorithm. After some experiments (omitted due to limited space) we found an optimal combination in terms of F_1 for all tested graphs. Namely, negative sampling is used to optimize the objective as done in LINE:

$$J_\theta = \frac{1}{E} \sum_{(i,j)\in E} \left(\log \sigma(s_{ij}) + \sum_{j' \sim p_n(j')}^{\nu} \log \sigma(-s_{ij'}) \right), \tag{3}$$

where bilinear model from BLM is used as score function:

$$s_{ij} = \boldsymbol{u}_i \cdot \boldsymbol{v}_j - Z_i. \tag{4}$$

Embedding vectors are initialized as $\boldsymbol{u}_i, \boldsymbol{v}_i \sim \mathcal{U}[-\frac{1}{2\sqrt{d}}, \frac{1}{2\sqrt{d}}]$ (where d is dimensionality of embedding space) and $Z_i = \log |N|$; noise edges are filtered such that $(i, j') \notin E$ only are sampled as negative examples; noise distribution $p_n(j) \propto deg(j)^{3/4}$ [26]. Regularization is eliminated. Number of noise samples for each edge $\nu = 25$, learning rate $\eta = 0.025$, training is performed during 200 epochs.
Thereby, the set of parameters of the algorithm is: $\theta = \{\boldsymbol{u}_i, \boldsymbol{v}_i, Z_i\}_{i=1}^{|N|}$, representation vector learnt for node $i \in N$ is: $\boldsymbol{r}_i = \begin{bmatrix}\boldsymbol{u}_i \; \boldsymbol{v}_i \; Z_i\end{bmatrix}^T$.
For optimization we used asynchronous stochastic gradient descent like in LINE and BLM. At each step gradient is computed by one edge:

$$\begin{aligned} \frac{\partial J_\theta^{(i,j)}}{\partial \theta} &= \frac{\partial}{\partial \theta} \log \sigma(s_{ij}) + \sum_{j' \sim p_n(j')}^{\nu} \frac{\partial}{\partial \theta} \log \sigma(-s_{ij'}) = \\ &= \sigma(-s_{ij}) \frac{\partial s_{ij}}{\partial \theta} + \sum_{j' \sim p_n(j')}^{\nu} \sigma(s_{ij'}) \frac{\partial s_{ij'}}{\partial \theta}. \end{aligned} \tag{5}$$

We considered embedding to be successful if graph edges could be restored with $F_1 > 0.99$. This means that obtained representation explains

more than 99% of graph edges under the model, while rest 1% may be outliers.
We also discovered that dimensionality of embedding space d is crucial parameter of embedding. Minimal d such that F_1 reaches 0.99 for a particular graph could be viewed as a "complexity" of this graph under the embedding model. We found that it varies for different graphs (see Fig.2).

4.2 Distribution approximating + sampling

At this stage we have a vector representation of nodes of an input graph $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$ such that it can be restored from them with $F_1 > 0.99$. The next task is to model the distribution of node vectors $\boldsymbol{r}_i \sim \mathcal{R}$, such that new node vectors sampled from $\mathcal{R}$ would produce (using reconstruction procedure from Sect.4.1) graphs with similar properties. This step handles creation of a graph generative model and provides randomization and variability of size of generated graphs. In order to experimentally compare graphs of different size, we use cosine similarity of their 3-GP vectors and "eyeball" similarity of the form of their degree distributions.
Since the input graph is embedded, node vectors encode information about graph structure. We assume that key statistical graph properties are reflected in *distribution* of $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$, rather than in individual vectors. Therefore a model captured the distribution $\mathcal{R}$ such that $\boldsymbol{r}_i \sim \mathcal{R}$ would also contain these properties. Furthermore if we sample a new set of node vectors from $\mathcal{R}$ and construct a new graph by the same procedure, it will also demonstrate these properties. This idea is justified well by an example of random dot product graphs, where node vectors distribution analytically determines graph properties [21]. Another benefit of this approach is that number of vectors sampled and therefore size of generated graph may be varied.
To model a distribution $\mathcal{R}$ we suggest the following method called GN based on gaussian noise. GN just memorizes the whole set of vectors $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$ and adds to each one gaussian noise with magnitude ϵ. To draw a sample from $\mathcal{R}_\epsilon$ it randomly samples $i \in \{1..|N|\}$ and returns $\boldsymbol{r}_i + \boldsymbol{g}$, $\boldsymbol{g} \sim \mathcal{N}(0, diag(\epsilon, ..., \epsilon))$. This is in fact kernel density estimation with a normal kernel. In case of large graphs storing all N vectors may be excessive, so one could randomly sample a smaller subset from them.

Size of scaled graphs If we scale a graph with n_0 nodes and e_0 edges by a factor of x, it should have $n_x \approx x n_0$ nodes. What number of edges e_x should it have, or in other words what is $e(n)$ law? In our approach the law $e \sim n^2$ can be proved theoretically for graphs generated by one model despite of probability distribution model:

Theorem 1. *Let $\mathcal{R}$ be a probability distribution, $s_{ij} = s(\boldsymbol{r}_i, \boldsymbol{r}_j)$ be a real-valued function. If $\boldsymbol{r}_i \sim \mathcal{R}$ and edge $i \to j$ is defined by condition $s_{ij} > t_G$, then if sample n vectors the number of edges $e \propto n^2$.*

Proof. Since $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ come from the same distribution edge probability between two randomly chosen nodes $P(s_{ij} > t_G) = p$ is a constant and is determined only by $\mathcal{R}$. Fraction of pairs connected with edges doesn't depend on the number of node-vectors sampled n, i.e. $\mathbb{E}(e/n^2) = p = const$. Hence $e \propto n^2$. $\square$

As long as real graphs may exhibit different laws of growth, we may treat an ERGG-dwc imitation of a real graph not as its future state, but as its scaled version.

4.3 Attributing

Now we are able to generate controllable size directed graphs structurally similar to a given one. The final step is to handle edge weights and community labels assigning in a generated graph. Their coherence with graph topology should be preserved. How do we make communities in new graphs and endow them with weights in a proper way?

Community structure We view a community structure of the graph as its nodes labelling: each node i has a (possibly empty) set of community labels $\mathcal{C}_i$ it belongs to. We suggest to inherit these labels in a generated graph using GN sampling method: if a node k of a new graph was sampled from node i, it has the same labels $\mathcal{C}'_k = \mathcal{C}_i$. In this way, given uniform sampling of nodes in GN, communities in a new graph become proportionally scaled images of the original ones.

Weights In order to assign edge weights in a generated graph we also inherit original weights using GN sampling method. For edge (k, l) of a new graph, if adjacent nodes k, l were sampled from nodes i, j of the original graph, we assign corresponding weight $w'_{kl} = w_{ij}$.
The question left is what if the original graph $G(N, E)$ doesn't have an edge (i, j)? One reason is incorrect embedding of (i, j). Since the fraction of edges $(i, j) \notin E$ after successful embedding is less than 1%, their weights wouldn't affect the results much. In this case a random weight may be chosen: $w_0 \sim \mathcal{U}(\{w_{ij}\}_{(i,j)\in E})$.
Another reason is the noise added to the sampled node vectors. In this case we can consider such an edge a weak connection, which motivates choice of default weight as minimal possible weight: $w_0 = \min_{(i,j)\in E} w_{ij}$.
In order to check correctness of weights assignment, we scaled weighted graphs with communities and used modularity measure designed for directed weighted graphs with communities [7]. High value of this metric is a kind of evidence that communities are more densely (accounting also for edge weights) connected within than between each other. Obtaining modularity values as high as in the original graph in experiments supports the hypothesis about weights assigning (see Sect.5.1).
Complexity of ERGG-dwc is $O((|E|\nu \log \frac{|E|}{|N|} + x^2|N|^2)d)$, details are omitted due to space limit.

5 Experiments

When developing our ERGG-dwc algorithm we used a set of small and medium size directed graphs from different domains and also artificial graphs (see table 2 for their parameters). Graph embedding is implemented in C++ using `pthreads` library for thread parallelization.

Table 2: Left: directed graphs; right: directed weighted graphs with detected communities. Graphs parameters: name on plots, number of nodes N, number of edges E, modularity [7] Q, embedding space dimensionality d such that $F_1 > 0.99$.

graph	**N**	**E**	**d**
Karate[2]	34	78	3
Yeast[3]	688	1079	9
VAST[4]	400	1562	12
Foods[5]	128	2106	8
TW [16]	146	1309	12
Kron[6] [17]	2187	11675	24
Words[3]	2704	8300	17
ER[7] [17]	800	8000	26
G+ [16]	1243	106485	62

graph	**N**	**E**	**Q**	**d**
Protein[3]	95	213	0.6630	7
Resid[8]	217	2672	0.5106	14
VAST[4]	400	1562	0.5743	12
Airport[9]	1574	28236	0.1247	31
LFR[10] [12]	1000	14396	0.7209	26

5.1 ERGG-dwc steps elaborating

Embedding method parameters We investigated how F_1 depends on feature space dimensionality d (see Fig.2). All graphs successfully reach $F_1 = 0.99$ at some d (see table 2), which we called their "complexity". The value d corresponding to $F_1 = 0.99$ for a particular graph can be determined via binary search: graph is iteratively embedded with different d and F_1 is approximately estimated for each. This usually takes 5-7 iterations.

Distribution approximating and sampling Here we fixed an embedding for each test graph such that $F_1 > 0.99$ and experimented with approximating of embedding vectors distribution.
An input here is a set of representation vectors $\boldsymbol{r}_i = \begin{bmatrix}\boldsymbol{u}_i & \boldsymbol{v}_i & Z_i\end{bmatrix}^T$ (of length $2d+1$) for $i = 1..|N|$ and threshold t_G. Distribution approximating algorithm is applied to fit a given set $\{\boldsymbol{r}_i\}_{i=1}^{|N|}$ and then used to sample new $m = \lfloor x|N| \rfloor$ vectors. Corresponding nodes are connected with edges

[2] `http://support.sas.com/documentation/cdl/en/procgralg/68145/HTML/default/viewer.htm#procgralg_optgraph_examples07.htm` Edges were considered directed.

[3] `http://www.weizmann.ac.il/mcb/UriAlon/download/collection-complex-networks`

[4] `http://hcil2.cs.umd.edu/newvarepository/VAST%20Challenge%202008/challenges/MC3%20-%20Cell%20Phone%20Calls/` Edge weight is the number of calls

[5] `http://vlado.fmf.uni-lj.si/pub/networks/data/bio/foodweb/foodweb.htm`

[6] Used SNAP generator with parameters "`krongen -m:'0 0.783, 0.003, 0.733; 0.147, 0.636, 0.772; 0.028, 0.700, 0.009' -i:9`"

[7] Used SNAP generator with parameters "`graphgen -g:e -n:800 -m:8000`"

[8] `http://moreno.ss.uci.edu/data.html#oz`

[9] `https://toreopsahl.com/datasets/#usairports`

[10] Run with parameters "`-N 1000 -om 3 -on 0.5 -maxk 150 -t1 2.4 -t2 1.6`"

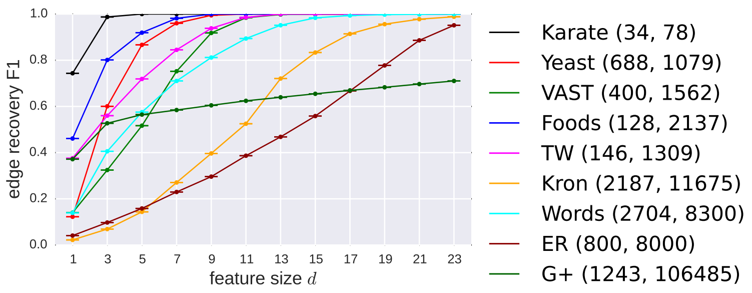

Fig. 2: Edge-recovery F_1 measure of restored graphs versus embedding vectors dimensionality d. Averaged over 5 runs.

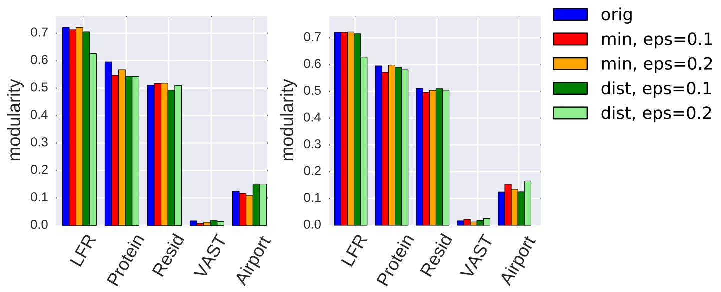

Fig. 3: Modularity of communities in ERGG-dwc generated graphs for two default edge weightings 'min' and 'dist' depending on GN noise magnitude ϵ. Scaling factors $x = 1$ (left) and $x = 4$ (right). Original communities are detected by OSLOM (except for LFR). Values are averaged over 5 runs.

according to $s_{ij} > t_G$ condition, and finally dangling nodes are removed, giving an output graph $H = (M, F)$.
We measured cosine similarity between 3-GPs of a generated graph and an original one; number of nodes M/xN, and number of edges F/x^2E in relation to their expected values, with different scaling factors x. We found that GN with $\epsilon \in [0.1; 0.2]$ performs best in terms of these metrics.

Community labels and edge weights For experiments here we used several directed weighted graphs from various domains and applied a community detection method (OSLOM [13]), and also one synthetic graph generated by LFR [12] (see table 2, right). We also fixed embeddings with $F_1 > 0.99$ and applied GN inheriting community labels and edge weights as described in 4.3. We compared modularity of generated communities in the generated graphs for two methods of default weight choice, minimal weight ('min') and random weight from the distribution ('dist'). We varied noise magnitude ϵ and scaling factor x.
We found that when $\epsilon \in [0; 0.2]$ modularity remains approximately the same in average both for $x = 1$ and $x = 4$, while higher $\epsilon = 0.3$ makes

it lower especially significant at $x = 4$ (not plotted). This again proved $\epsilon \in [0.1; 0.2]$ to be optimal values, therefore we used them further.
Comparison of default weighting schemes showed that both of them perform almost identically in terms of modularity on different graphs (see Fig.3). at $\epsilon = 0.1$, while at $\epsilon = 0.2$ on large graphs 'dist' scheme leads to larger modularity loss than 'min' scheme.

5.2 Variability evaluation

In order to apply ERGG-dwc for significance testing of various network mining tools, generated graphs should be not only similar to the original one but also differ from each other. Variability of graph imitations produced by ERGG-dwc should be wide enough to model the natural variability across real networks. For assessing the variability of ERGG-dwc we considered how different graph statistics vary across graphs from one domain and compared to corresponding variances of different ERGG-dwc imitations of one graph of them. For that we chose a set of twitter-ego nets and picked 15 graphs close in number of nodes ($|N| \in [170; 180]$) and number of edges ($|E| \in [2000; 3000]$) as a dataset. We analyzed in-degree, 3-GP, and clustering coefficient distributions for these graphs and also for 15 ERGG-dwc imitations of one of them: Fig.4 demonstrates similar variabilities for both sets.

5.3 Significance testing

Here we demonstrate how to perform significance testing of several network mining tools using ERGG-dwc. For that we chose a set of Google+ ego-nets and picked 8 graphs close in number of nodes ($|N| \in [450; 500]$). For each graph we generated 5 imitations and ran an algorithm on the graph (black triangles at Fig.5) and on the imitations (blue triangles at Fig.5). We measured modularity for community detection methods OSLOM and Infomap[11] and running time for diameter computing algorithm, see Fig.5 (green triangles correspond to modularity of communities generated by ERGG-dwc).
Looking at the plots one can conclude that OSLOM produces insignificant results in terms of modularity, while Infomap's result are much more significant.
We also tested performance of diameter computing algorithm (Fig.5, 2 plots on the right). We also used 5 Google+ ego-nets and 5 ego-nets from Twitter ($|N| \in [170; 180]$), generating 15 ERGG-dwc imitations for each graph. Everything is as expected for Twitter, but more time is needed to figure out the reasons of Google+ results.

5.4 Comparison to SKG

Now we compare the work of ERGG-dwc with SKG [14] in terms of various graph features. We fitted US Airports graph[12] (Airport) by our algorithm and SKG. Fitting was done by Kronfit with default parameters.

[11] `http://www.mapequation.org/code.html`
[12] `https://toreopsahl.com/datasets/#usairports`

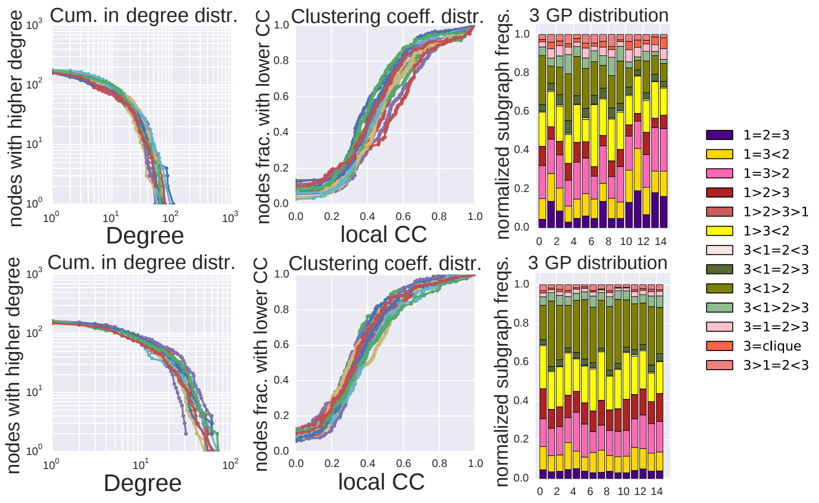

Fig. 4: Variability evaluation for cumulative in-degree distribution, cumulative clustering, 3-GP. Top row: set of 15 twitter-ego graphs of size $|N| \in [170; 180]$, $|E| \in [2000; 3000]$, bottom row: 15 ERGG-dwc imitations of one of them.

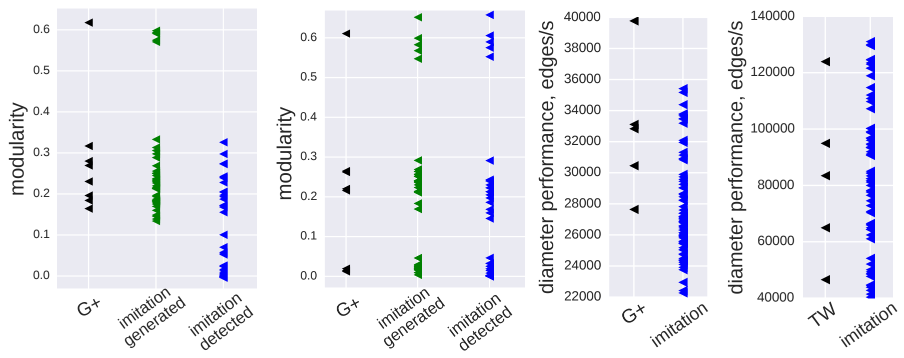

Fig. 5: Significance tests. From left to right: OSLOM CD, Infomap CD, performance of diameter computing Google+ ego-nets and Twitter ego-nets. 8 ego-nets of Google+ ($N \in [450; 500]$) and 8 ego-nets of Twitter ($N \in [170; 180]$) are used.

Degree distribution and spectral properties are approximated slightly better via ERGG-dwc than SKG. 3-GP and clustering properties are not captured by SKG at all, while ERGG-dwc matches them well. SKG generated graph doesn't match number of nodes and hence average degree, but has lower diameter 5 that is significantly smaller than the original 8. Reciprocity metric (proportion of reciprocal edges) is 0.02 versus original 0.78 and 0.65 for ERGG-dwc. This may be explained by the fact that edges $i \to j$ and $j \to i$ are sampled independently even having same probability and thus reciprocal edge $i \leftrightarrows j$ is much less probable.

5.5 Discussion

In the experiments we showed that directed graphs from various domains can be successfully embedded into low-dimensional space by means of our modified embedding method COMBO. Embedding quality is evaluated in a special sense specific for RGG task: edges of graph can be restored with high F_1 (> 0.99). Optimal dimensionality d of the space depends on graph and is currently found by repetitive trials. Faster determination of optimal d for a given graph could be a future work.
Unfortunately, in our experiments we found that at high scaling factors x the form of degree distribution is not preserved. As it was shown theoretically the number of edges $|E|$ is proportional to $x^2|N|^2$.
Furthermore, GN method provides a simple way to inherit weights and community labels from the original graph. Experiments suggest that this labelling method preserves high modularity of generated communities in scaled graphs. These graphs may be used as benchmarks for testing community detection algorithms. Another future direction is advancing this naive method of labelling: current approach may cause staircase effects in edge weight and community size distributions at high scaling factors x.
Besides showing the closeness of generated graphs to the original one in terms of degree distributions and 3-GP, we found their variability wide enough to model a graph domain. This means that ERGG-dwc can be used to generate datasets for significance testing.
We also compared quality of fitting a real graph for ERGG-dwc and SKG in terms of several graph statistics and found that ERGG-dwc reproduces most of them much closer to the original. Although WCC distribution reveals many small connected components in ERGG-dwc graph, it may be not critical since the largest WCC is large enough.
Finally, we evaluated performance of ERGG-dwc and confirmed $O(|N|^2)$ generation time. One way to speed-up edge generation is to optimize finding all pairs (i, j) with $s_{ij} > t_G$. Some techniques used in nearest neighbor search related problem could be employed to overcome an exhaustive search. Another idea is to reduce the search space to pairs corresponding to existing graph edges, which makes finding edges F in $O(x^2|E|)$ steps.

6 Conclusion

We introduced and thoroughly evaluated an approach to modeling real directed graphs without a priori knowledge about their domain and properties. To the best of our knowledge, this is the first successful attempt to employ graph embedding technique in random graph generation. The resulting graphs are statistically similar to the input one, proving that representing graphs in low-dimensional space is the right way to obtain their scaled imitations.
Scalability of our method could be improved by adjusting embedding scheme and/or edge generation process. Also, edge weights and node communities are currently treated as attributes and in fact are cloned from the input graph. Additional research is required towards embedding weights and communities along with edges. The generating scheme should be adjusted accordingly to recover weights and communities from the embedding and preserve their statistical properties.

Acknowledgements

This research was collaborated with and supported by Huawei Technologies Co.,Ltd. under contract YB2015110136.
We are also thankful to Ilya Kozlov and Sergey Bartunov for their ideas and valuable contributions.

References

1. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
2. I. Bordino, D. Donato, A. Gionis, and S. Leonardi. Mining large networks with subgraph counting. In *2008 Eighth IEEE International Conference on Data Mining*, pages 737–742. IEEE, 2008.
3. D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *SDM*, volume 4, pages 442–446. SIAM, 2004.
4. K. Chykhradze, A. Korshunov, N. Buzun, R. Pastukhov, N. Kuzyurin, D. Turdakov, and H. Kim. Distributed generation of billion-node social graphs with overlapping community structure. In *Complex Networks V*, pages 199–208. Springer, 2014.
5. R. Cohen and S. Havlin. Scale-free networks are ultrasmall. *Phys. Rev. Lett.*, 90:058701, Feb 2003.
6. S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. Pseudofractal scale-free web. *Phys. Rev. E*, 65:066122, 2002.
7. M. Drobyshevskiy, A. Korshunov, and D. Turdakov. Parallel modularity computation for directed weighted graphs with overlapping communities. In *Proceedings of the Institute for System Programming*, volume 28(6), pages 153–170, 2016.
8. P. Erd6s and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.
9. M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, volume 1, page 6, 2010.
10. O. U. Ivanov and S. O. Bartunov. Learning representations in directed networks. In *International Conference on Analysis of Images, Social Networks and Texts*, pages 196–207. Springer, 2015.
11. D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.
12. A. Lancichinetti and S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80(1):016118, 2009.
13. A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato. Finding statistically significant communities in networks. *PloS one*, 6(4):e18961, 2011.
14. J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042, 2010.

15. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.
16. J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.
17. J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.
18. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
19. E. Mossel, J. Neeman, and A. Sly. Stochastic block models and reconstruction. *arXiv preprint arXiv:1202.1499*, 2012.
20. A. A. Nanavati, S. Gurumurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjea, and A. Joshi. On the structural properties of massive telecom call graphs: Findings and implications. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 435–444, New York, NY, USA, 2006. ACM.
21. C. L. M. Nickel. *Random dot product graphs: A model for social networks*, volume 68. 2007.
22. G. Palla, L. Lovász, and T. Vicsek. Multifractal network generator. *Proceedings of the National Academy of Sciences*, 107(17):7640–7645, 2010.
23. G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos. Using graph theory to analyze biological networks. *BioData Mining*, 4(1):10, 2011.
24. B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
25. C. L. Staudt, M. Hamann, I. Safro, A. Gutfraind, and H. Meyerhenke. Generating scaled replicas of real-world complex networks. *arXiv preprint arXiv:1609.02121*, 2016.
26. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. ACM, 2015.
27. D. J. Watts and S. H. Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.
28. A. Wegner et al. Random graphs with motifs. 2011.
29. M. Winkler and J. Reichardt. Motifs in triadic random graphs based on steiner triple systems. *Physical Review E*, 88(2):022805, 2013.
30. X. Ying and X. Wu. Graph generation with prescribed feature constraints. In *SDM*, volume 9, pages 966–977. SIAM, 2009.
31. S. J. Young and E. R. Scheinerman. Random dot product graph models for social networks. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 138–149. Springer, 2007.