Attributed Graph Modeling with Vertex Replacement Grammars

Satyaki Sikdar
ssikdar@nd.edu
University of Notre Dame

Neil Shah
nshah@snap.com
Snap Inc.

Tim Weninger
tweninge@nd.edu
University of Notre Dame

Abstract

Recent work at the intersection of formal language theory and graph theory has explored graph grammars for graph modeling. However, existing models and formalisms can only operate on homogeneous (*i.e.*, untyped or unattributed) graphs. We relax this restriction and introduce the Attributed Vertex Replacement Grammar (AVRG), which can be efficiently extracted from heterogeneous (*i.e.*, typed, colored, or attributed) graphs. Unlike current state-of-the-art methods, which train enormous models over complicated deep neural architectures, the AVRG model is unsupervised and interpretable. It is based on context-free string grammars and works by encoding graph rewriting rules into a graph grammar containing graphlets and instructions on how they fit together. We show that the AVRG can encode succinct models of input graphs yet faithfully preserve their structure and assortativity properties. Experiments on large real-world datasets show that graphs generated from the AVRG model exhibit substructures and attribute configurations that match those found in the input networks.

CCS Concepts

• **Mathematics of computing** → **Graph algorithms**; **Random graphs**; • **Theory of computation** → **Random network models**.

Keywords

graph models, assortativity, graph generation, attributed graphs

1 Introduction

One of the principal goals of graph mining is to discover patterns found in real-world graphs and use them to make predictions about various relational systems. Many such systems consist of differently typed components; it has long been argued that models which treat graphs as homogeneous, without distinguishing different types of objects and links, ignore important information. By leveraging these rich semantics as *attribute labels* on nodes and edges, researchers
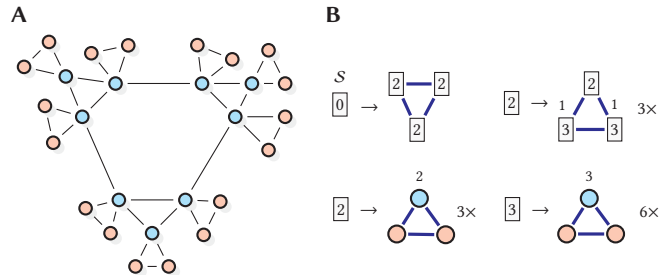
Figure 1: (A) An attributed graph with colors representing node attributes. (B) An Attributed Vertex Replacement Graph Grammar (AVRG) extracted from (A). Each grammar rule contains an LHS and RHS. Rule frequencies are indicated with a ×.

have developed attributed graph analysis techniques to better model such data (*e.g.*, via heterogeneous information networks), leading to improvements in downstream inference tasks [44, 45]. Because graphs encode interaction information about many real-world relational systems, learning how to generatively model such graphs offers a promising window into their dynamics, spurring a rich line of work [10, 20, 23, 36], inspiring our present one.

Present Work. Our work[1] lies at the intersection of attributed graph mining and graph generation. Specifically, we build upon a class of local topology-aware generators called Vertex Replacement Grammars (VRGs). Having their origins in formal language theory, where objects of different types (*e.g.*, characters, glyphs, words, parts of speech), graph grammars offer a promising avenue in modeling attributed graphs like the small example illustrated in Fig. 1(A). Attributed Vertex Replacement Grammar (AVRG) builds on recent advances in graph grammars [1, 18, 41], which were designed only to model and generate homogeneous graphs, to handle *attributed* graphs, increasing its applicability to a broad class of modern problems.

Graph neural networks (GNNs) comprise a wide swath of models with similar goals in attributed graph modeling. These models train deep neural network architectures using loss functions and stochastic gradient descent. GNNs offer excellent modeling capacity in downstream machine learning tasks but can be difficult to train, have enormous parameter spaces, and do not readily permit human inspection. Despite performing well on link prediction and node classification tasks, some GNN architectures fail to preserve the topological fidelity of the input graphs [40]. AVRGs, on the other hand, can capture and model the subtle intricacies in both topological and attribute spaces without requiring supervision, deep neural architectures, and training.

---

[1]The source code is available at https://github.com/satyakisikdar/Attributed-VRG

Simply put, the goal of the present work is to extract graph rewriting rules, like those illustrated in Fig. 1(B), that are representative of the topology and attribute configuration of the original graph. Our key idea is to use an exemplar input graph to build a grammar of graph rewriting rules with patterns that can be applied stochastically to generate new, diverse graphs that meet desirable attribute and topology-related graph properties. Both formally and through a running example, we introduce and describe the **Attributed Vertex Replacement Grammar** (AVRG) formalism and show the following:

(1) AVRGs can extract interesting and frequently appearing attributed subgraphs from large real-world graphs,
(2) AVRGs can generate graphs that match the topology and attribute assortativity of the original graph, and
(3) the ability of AVRGs to handle a wide variety of assortative and disassortative graphs.

Although graph grammars appear to be a natural fit for modeling attributed graphs, this task presents many challenges. Our paper describes many of the modeling decisions made as a result and their consequences. We also demonstrate the power and flexibility of AVRGs with two case studies–the former focuses on the ability to extract meaningful patterns in Sec. 4.5, and the latter on testing the generative ability of the model on well-known attributed graphs with various degree and attribute assortativity values in Sec. 6.1. We find that AVRG models provide a succinct yet faithful representation of input graphs–often better than attribute-based probabilistic models and graph neural network models at a far smaller computational cost and model size.

2 Related work

Attributed graphs are at the core of most modern graph mining literature. Considerable prior work focuses on clustering [29, 58], classification [54], anomaly detection [39], similarity [5, 43], and search [48] tasks on attributed graphs. The end goal of such methods is usually *descriptive* or *inferential*. On the one hand, descriptive methods tend to focus on discovering, enumerating, or counting patterns in given graph data [3, 16], which may be attributed [46, 55] or dense [19, 28]. On the other hand, inferential methods involve statistical reasoning over graph data to make educated inferences (*e.g.*, on nodes, edges, or graphs) using examples [13, 15, 21, 24, 26, 30, 47, 51, 54, 59, 59, 60].

Other works study graphs through a *prescriptive* lens by characterizing their formation and generative processes, usually from a mathematical perspective. This subfield, called graph generation, typically involves positing generative processes that aim to produce graphs with desirable (usually topological) statistical properties that match real-world phenomena. These include exponential random graphs [36], Kronecker graphs [23], Chung-Lu graphs [10], and Stochastic Block Models (SBMs) [20] involves learning parameters via likelihood maximization based on a given input graph, which can be used to generate new graphs mimicking the input graph's properties. Several very recent works also involve neural graph generation with recurrent neural networks [57], variational autoencoders [42], and generative adversarial networks [7]. However, most graph generation literature focuses on non-attributed graphs. The

few exceptions [25, 31, 32] adapt Chung and Lu's model for attributed graphs; yet, they ignore local graph topology, which is highly diverse across nodes in real graphs [14] with implications for tasks like anomaly detection [4], social contagion and diffusion [9, 50], and engagement prediction [56].

A recent addition to the class of graph generators are hyperedge [1, 2] and node replacement [18, 41] grammars. Graph grammars contain graphical rewriting rules that match and replace graph fragments, similar to how a context-free string grammar rewrites characters in a string. These graph fragments represent a succinct description of the network's building blocks, and the graph grammar's rewiring rules describe the instructions about how the graph is pieced together. Unfortunately, existing grammar formalisms do not provide for attributes, nor do they capture complex attribute mixing patterns during graph generation, making them unsuitable for attributed graphs [1, 18, 41].

3 Attributed Vertex Replacement Grammars

3.1 Preliminaries

Labeled Attributed Multigraphs. A labeled attributed multigraph is a 5-tuple $H = \langle V, E, \kappa, L, \mathcal{A} \rangle$ where V is the set of vertices; $E \subseteq V \times V$ is the set of edges; $\kappa : E \mapsto \mathbb{Z}^+$ is a function assigning multiplicity to edges; L is the set of labels on nodes; and $\mathcal{A} : V \mapsto L$ is a function mapping nodes to attribute values in L. By default, each edge has a multiplicity of 1. Although our model can be used for directed graphs, the present work treats all graphs as undirected for clarity of prose and illustration. Attributes can be internal, based on the graph topology like node degree, or external, based on externally observed parameters, such as age or gender. We only focus on discrete attributes for simplicity, but these models can usually be extended relatively easily to handle continuous values. We use $|\mathcal{A}|$ to denote the number of unique attribute values, so for the graph in Fig. 1(A), $|\mathcal{A}| = 2$ because it contains two different types/colors of nodes: *blue* and *pink*.

Assortativity. Assortativity functions permit us to quantify the nature of mixing patterns of attributes in networks. In this work, we adopt the definitions proposed by Newman [26]. High assortativity values represents homophily, the *birds-of-a-feather* principle [24], where nodes are likely to connect to nodes with similar attributes. Typically graph modelers are interested in degree assortativity. That is, the assortativity of a network where node degrees were considered to be attributes. In fact, in much of the related literature, the term *assortativity* is generally taken to imply degree assortativity. However, in the present work, we endeavor to be precise when referring to the domain. Table 1 describes 12 datasets that are used as the experimental corpora in the present work. Each dataset is labeled with the number of nodes $|V|$, edges $|E|$, and number of unique attributes $|\mathcal{A}|$; as well as the degree assortativity ρ_{deg} and external attribute assortativity ρ_{attr}.

Normalized Dasgupta Cost (NDC). We judge the goodness of a dendrogram $\mathcal{D}$ obtained from hierarchical graph clustering algorithms using the (NDC) metric defined below [12].

$$NDC(\mathcal{D}, G) = \frac{\sum_{(u,v) \in E} |\text{leaves}(\text{LCA}(u, v, \mathcal{D}))|}{|V| \cdot |E|} \qquad (1)$$

where $|\text{leaves}(\text{LCA}(u, v, \mathcal{D}))|$ is the number of leaves in the subtree rooted by lowest common ancestor (LCA) of nodes u and v. A good dendrogram $\mathcal{D}$ is expected to have a low cost because connected nodes in the graph should share an LCA lower down in the tree.

3.2 The AVRG Formalism

An Attributed Vertex Replacement Grammar (AVRG) is a 4-tuple $G = \langle \Sigma, \Delta, \mathcal{P}, \mathcal{S} \rangle$ where Σ is the alphabet of node labels; $\Delta \subseteq \Sigma$ is the alphabet of *terminal* node labels; $\mathcal{P}$ is a finite set of productions rules ($\mathcal{R}$) of the form $X \rightarrow (R, f)$, where X is a left-hand side (LHS) consisting of a *nonterminal* node (*i.e.*, $X \in \Sigma \setminus \Delta$) with a size ω; and the tuple (R, f) represents the right-hand side (RHS) of the production rule, where R is a labeled attributed multigraph with terminal and possibly nonterminal nodes, and $f \in \mathbb{Z}^+$ is the rule frequency, *i.e.*, the number of times it appears in the grammar; and $\mathcal{S}$ is the starting graph $\boxed{0}$, a nonterminal of size 0.

Fig 1(B) shows an example AVRG with four rules. Each rule contains an LHS and RHS. Squares containing numbers are nonterminal-nodes that can be replaced by a rule's RHS. Colored nodes on the RHS subgraphs represent terminal nodes from the input graph. Every node in R is labeled by the number of boundary edges adjacent to it in the original graph. The sum of boundary degrees on the RHS is defined as ω and used to label the LHS.

Incorporating attributes in the formalism introduces new challenges in both extraction and generation; we elaborate next.

4 Extracting AVRGs

At a high level, our goal in the extraction process is to identify small but meaningful building blocks of a graph while also keeping track of how they can be pieced together. One way to find such building blocks would be to exhaustively enumerate all possible subgraphs up to a certain size, but this quickly becomes infeasible for even medium-sized graphs. We opt to use a dendrogram obtained from a hierarchical clustering algorithm to help us find the building blocks to make our model scalable for large graphs. This works well in practice because real-world graphs are usually self-similar across multiple scales, and therefore their nodes can be decomposed into clusters in a hierarchical fashion [34]. Once we discover the blocks, we extract grammar rules and contract both the graph and the dendrogram. We repeat this extraction and contraction process until the graph is empty. It is important to note that we only run the clustering algorithm once at the beginning and not every time the graph gets updated.

4.1 Selecting a *Good* Clustering Algorithm

We select 7 popularly used hierarchical graph clustering algorithms: conductance based bisection (Conductance) [17], spectral clustering (Spectral) [27], Hyperbolic Hierarchical Clustering (HyperHC) [8], Leiden [49], Louvain [6], Infomap [37], and recursive label prop-agation (LabelProp) [33] algorithms. We run these algorithms on the graphs described in Tab. 1 and obtain dendrograms. For every dendrogram, we calculate its Normalized Dasgupta Cost as defined in Eq. 1, group on clustering algorithm (*i.e.*, average across datasets), and plot the mean cost and 95% confidence interval (in blue) in Fig. 5. We adopted the formalism from Sikdar et al. to calculate the description length [41] and the inverse compression ratio, *i.e.*, the
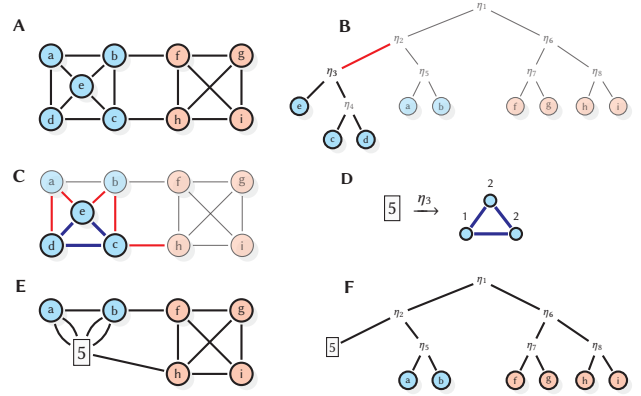

Figure 2: One iteration of the rule extraction and graph compression process. (A) Graph H. (B) $\eta^* = \eta_3$ in $\mathcal{D}$. (C) $V_{\eta_3} = \{c, d, e\}$, boundary edges are drawn in red. (D) Extracted grammar rule. (E) New, reduced graph H' with the new nonterminal X of size 5. (F) Updated dendrogram $\mathcal{D}'$.

ratio of the description length of a grammar and the input graph, such that lower is better. The mean inverse compression ratio and 95% confidence interval are plotted (in red) in Fig. 5 alongside the Normalized Dasgupta cost. We find that Conductance, Louvain, and Leiden perform best on average, so we down-select these three clustering algorithms for the next steps.

4.2 Selecting a Suitable Internal Tree Node

Let $T = \{\eta_1, \cdots, \eta_k\}$ be the set of internal nodes in the dendrogram $\mathcal{D}$. We observe that the leaves of a subtree rooted at an internal node $\eta_i \in T$ correspond to a subset of vertices of the H, *i.e.*, $V_{\eta_i} \subseteq V$. For example, in Fig. 2(A), $V_{\eta_3} = \{c, d, e\}$ and let H_η be the subgraph induced by V_η on H. Considering each internal node is not particularly helpful, especially in large graphs, given the overlap in the subgraphs induced by a tree node and its ancestors. Therefore, our goal is to determine a strategy that selects a particular tree node η^* from T that creates small but meaningful grammar rules. So, let s be a function $s : T \mapsto \mathbb{R}$ which assigns scores to individual tree nodes, and $T^* \subseteq T$ be the set of the *lowest* scoring tree nodes, *i.e.*, $T^* = \arg\min_{\eta \in T}\{s(\eta)\}$. Now, depending on the scoring function and the state of the current dendrogram, T^* may contain more than one *optimal* tree node, in which case we need a selection policy.

The most straightforward policy would be to pick a tree node at *random* from T^*. A slightly more sophisticated policy would consider scoring functions that operate based on the subtree's size. In this approach, we introduce a new parameter $\mu \in \mathbb{Z}^+$. Our goal is to extract rules where the RHS subgraphs have exactly (or nearly) μ nodes. This aligns with our goal of finding small but meaningful rules from the input graph. A simple scoring function that enforces the size-based approach would be $s(\eta_i) = |V_{\eta_i}| - \mu$.

Once we pick the best tree node $\eta^* \in T^*$, we proceed to extract a grammar rule $\mathcal{R}$ that we then add to grammar G.

4.3 Extracting an AVRG Rule

Let $H_{\eta^*} = (V_{\eta^*}, E_{\eta^*})$ be the subgraph induced by V_{η^*} on H, where we copy over the corresponding attribute values from V to V_{η^*},
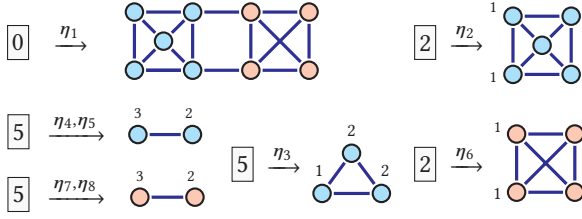
Figure 3: Set of all grammar rules obtained from the dendrogram $\mathcal{D}$ for graph H from Fig. 2(A). Note some tree nodes like η_4 and η_5 produce the same rule.

and $E_{\eta^*} = \{(u,v) \mid u,v \in V_{\eta^*} \wedge (u,v) \in E\}$. Let $E_{\text{cut}}^* = \{(u,v) \mid u \in V_{\eta^*} \wedge v \in V\}$ be the set of edges, called *boundary edges*, that span from H_{η^*} to H, and let $\mathbf{b} = [b_1 \ b_2 \ \cdots \ b_{|V_{\eta^*}|}]^T$ be the vector of *boundary degrees* for each node $v \in V_{\eta^*}$. The boundary degree (b_i) of a node v_i stores the number of boundary edges to which v_i is adjacent, *i.e.*, $b_i = |\{(u,v) \mid (u,v) \in E_{\text{cut}}^* \wedge (u = v_i \vee v = v_i)\}|$, and therefore $\sum \mathbf{b} = |E_{\text{cut}}^*| = \omega$. We include the boundary degree as an additional attribute for each node in H_{η^*}.

We now have all the ingredients needed to create a grammar rule. Let $\mathcal{R} = X \rightarrow (R, f)$ be an AVRG rule, where X is a nonterminal of size ω, R is the graph H_{η^*}, and the frequency $f = 1$. Now, if $\mathcal{R}$ is a new rule, we add it to the grammar G; otherwise, we update the frequency of the existing rule in G that is *isomorphic* to $\mathcal{R}$. For example, we obtain 6 unique grammar rules from the dendrogram $\mathcal{D}$ in Fig. 2(B) which are shown in Fig. 3.

Rule Isomorphism.   Two grammar rules $\mathcal{R}_1$ and $\mathcal{R}_2$ are isomorphic if and only if (a) their left-hand side nonterminals are of the same size, *i.e.*, $\omega_1 = \omega_2$, and (b) the RHS attributed graphs are isomorphic. We use the VF2 algorithm for this calculation [11].

4.4 Updating the Data Structures

After extracting the grammar rule $\mathcal{R}$, we update both the current graph H and dendrogram $\mathcal{D}$ to prevent the same set of nodes from directly participating in future rules.

Updating the Graph.   In the current graph H, first we remove all the nodes in V_{η^*} and its corresponding edges. Then, we introduce a new nonterminal node X^* labeled with ω (from the rule extraction step). We connect X^* to the rest of the graph through the set of cut edges E_{cut}^* in R where the endpoints in V_{η^*} are redirected to X^*. Note, this may lead to the creation of multi-edges in the new graph H', which is now strictly smaller than H.

Updating the Dendrogram.   With a new (smaller) H', it may be prudent to re-run the clustering algorithm and generate a new dendrogram. However, our initial experiments found that re-clustering is time-consuming and rarely results in significant changes to the dendrogram. Instead, we modify $\mathcal{D}$ by replacing the subtree rooted at η^* with nonterminal node X^* labeled with ω. Finally, we set $H \leftarrow H'$ and repeat the tree node selection and rule extraction processes until the dendrogram is empty.

In Fig. 2 we continue the running example and illustrate the grammar rule extraction and graph and dendrogram contraction processes. The input graph (A) has 9 nodes, 16 edges, and each node is colored blue or pink. A dendrogram drawn from clustering
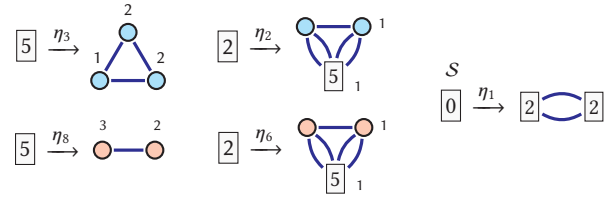

Figure 4: Grammar Rules for our example graph H using the dendrogram $\mathcal{D}$ in Fig. 2(B). The tree nodes η_3, η_2, η_8, η_6, and η_1 where selected in order to obtain these rules. Note that ω on the LHS is equivalent to the summation of the number of broken edges on the RHS, illustrated as numbers above or to the right of each RHS-node; $f = 1$ for all rules.
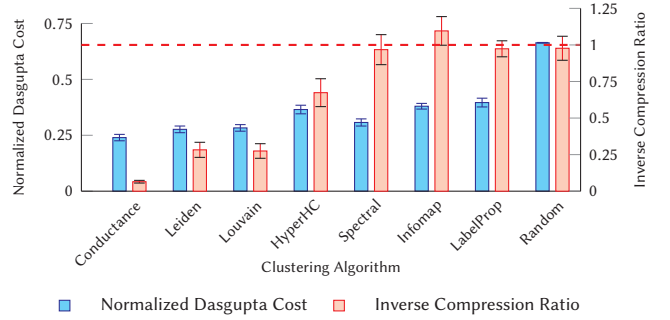

Figure 5: Mean-average Normalized Dasgupta Cost and Inverse Compression Ratio grouped by clustering algorithm. Error bars represent 95% confidence interval. Lower is better for both axes. Red dashed line at y=1 indicates no model compression. We pick Conductance, Leiden, and Louvain algorithms for further consideration.

of the graph is drawn in (B) from which η_3 is selected by using the size-based scoring function described in Sec. 4.2 with μ=3. The subgraph representing the subtree rooted at η_3 is highlighted in (C) and contains 5 boundary edges (drawn in red). (D) illustrates a new grammar rule constructed with size-five LHS and an RHS copied from the highlighted subgraph under η_3. Finally, the extracted subgraph and selected subtree are contracted and replaced by the nonterminal node from the LHS of the newly created rule in (E) and (F). This process then repeats until the dendrogram is empty, yielding us the set of rules shown in Fig. 4.

Before discussing generation, we examine AVRGs extracted from two real graphs with distinct attribute mixing patterns.

4.5 Case Study I: Examining AVRG Rules

Cora.   The citation network consists of about 2500 papers spread across 7 different fields of Machine Learning, which are represented by colors on the nodes. In Fig. 6(A), we observe that not all classes are represented equally; for example, 27% nodes are red while only 5% nodes are blue. The edge mixing patterns based on the node attributes in Fig. 6(B) reflect strong homophily, with most edges spanning between papers (nodes) in the same field (color), as evidenced by high values on the main diagonal as well as the high
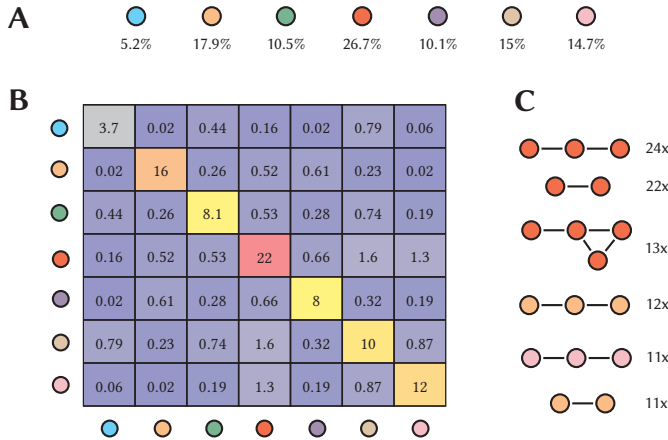
Figure 6: Cora. (A) Proportion of nodes in each class (color). (B) Attribute mixing matrix, each cell represents the percentage of edges with the corresponding colored endpoints. High numbers in the main diagonal and lack of off-diagonal edges indicate strong homophily. (C) The six most frequent rule RHS subgraphs of the extracted AVRG.
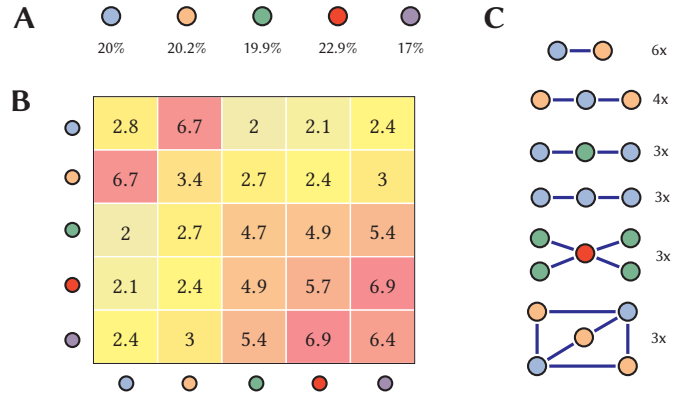

Figure 7: Chameleon. (A) Proportion of nodes in each class (color). (B) Attribute mixing matrix, each cell represents the percentage of edges with the corresponding colored endpoints. Almost uniform numbers in the matrix indicates heterophily. (C) The six most frequent rule RHS subgraphs of the extracted AVRG.

value of $\rho_{\text{attr}} = 0.764$. We expect the RHS subgraphs of the grammar rules extracted from Cora to capture this phenomenon. Indeed, in Fig. 6(C), we find that each of the 6 most frequent subgraphs consists of nodes belonging to the same class (color). Moreover, the top three rules all involve the red nodes, which is expected since 22% of the edges in the network span between red nodes.

Chameleon. The Chameleon Wikipedia network consists of about 2500 webpages on Wikipedia, which fall under the topic *Chameleon*. The nodes are classified into 5 (almost) quantiles based on their average monthly traffic, whose distribution is shown in Fig. 7(A). The edge mixing patterns, in this case, is quite intriguing. We see that nodes lying in the same quantile are not necessarily likely to be more connected to each other than others, resulting in a more uniform distribution of edges as seen in Fig. 7(B), resulting in the low attribute assortativity score of 0.032. The 6 most frequent RHS subgraphs in the grammar rules reflect this disparity too; all but one rule consists of nodes from different classes, as seen in Fig. 7(C).

Thus, it is clear that the grammar rules reflect the mixing patterns observed in the input graph, which empirically validates our extraction procedure's efficacy. Additionally, we can use the grammar rules to generate graphs which we describe next.

5 Graph Generation with AVRGs

The grammar G encodes information about the original graph H, which can be used for various purposes. One compelling way to determine the faithfulness of a graph model is to use it to generate graphs. How closely do these generated graphs resemble the original one? Is the model able to capture the nuances of mixing patterns across both degrees and attributes? To answer these questions, we first describe a stochastic generative process that repeatedly applies rewriting rules to generate graphs.

Any new graph H' starts with $\mathcal{S}$, a size 0 nonterminal. An AVRG extracted using the extraction process outlined above is guaranteed

to have a single size-0 starting nonterminal node because the grammar extraction only terminates when the root of the dendrogram covers all the nodes in the graph. Then grammar rules are applied stochastically according to f on available nonterminal nodes.

Precisely, each step of the generation process consists of three sequential parts: (a) choosing a nonterminal node from the current graph H', (b) choosing a suitable grammar rule from the grammar G, and (c) replacing the selected nonterminal in H' with the RHS subgraph of the selected rule, and rewiring it to the rest of the graph. This process continues until no more nonterminals remain or until some predetermined graph size is reached.

5.1 Selecting a Grammar Rule

We pick a nonterminal X of size ω at *random* from H' and find the set of grammar rules with the same LHS nonterminal, *i.e.*, of size ω. From that set of rules, we select a rule $\mathcal{R}$ stochastically according to frequency f and connect the RHS into H'. This process prevents overfitting to the input graph and enables the model to flexibly generate diverse graphs that still resemble the original.

5.2 Choosing a Rewiring Policy

How the RHS is connected to the existing graph matters. The degree of the chosen nonterminal in H' will always match its size, *i.e.*, ω. So, deleting X from H' will break exactly ω edges (called broken edges) that then need to be rewired to nodes in the RHS.

Recall that each node in the RHS subgraph of R was labeled with boundary degree b_i during extraction, and the sum of all boundary degrees is equal to ω. We will use this helpful property to guide the edge rewiring process. Doing so helps us preserve the local connectivity patterns across the different regions of the graph, thereby increasing the fidelity of the output. We consider three possible rewiring policies in the current work: (1) random rewiring, (2) mixing matrix, and (3) local greedy.
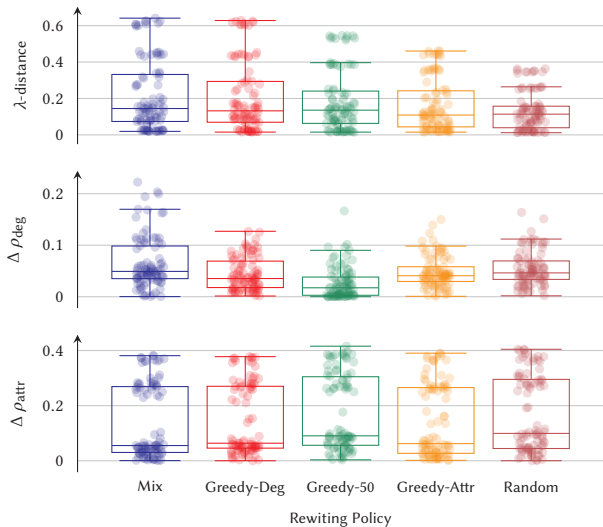
Figure 8: Rewiring policies over three metrics. Lower is better. We select the Mixing Matrix rewiring policy.

Random Rewiring. Each node i in the RHS subgraph of R receives exactly b_i edges at random from the set of broken edges.

Mixing Matrix. If we consider graph generation as a kind of reverse extraction process, from the top of the dendrogram moving downwards, then we can think of each RHS as introducing a new region/community of the graph–first at a macro-level and progressively refining the graph structure one rule at a time.

As their name implies, terminal nodes and their induced subgraph stay constant once they are introduced. We, therefore, distinguish two sets of terminal nodes: (1) terminal nodes in H', and (2) terminal nodes in the RHS subgraph of a rule that is being applied.

From among the terminal nodes in H', we select those which have edges that are broken. Then from the terminal nodes in the RHS of the rewriting rule, we select terminal nodes that have non-zero boundary degrees. Once we have identified these two sets, we use a Chung-Lu style edge rewiring policy that matches edges based on the attribute mixing matrix from the input graph. For instance, if red nodes are twice as likely to connect to other red nodes as blue nodes in the input graph, we will use that probability to match red and blue nodes similarly during the rewiring process for the broken edges between the two terminal node sets.

Local Greedy. We compute an error function for each possible rewiring between two terminal nodes and greedily pick a rewiring configuration that minimizes the local error:

$$\epsilon = \beta \cdot |\rho'_{\mathrm{deg}} - \rho_{\mathrm{deg}}| + (1 - \beta) \cdot |\rho'_{\mathrm{attr}} - \rho_{\mathrm{attr}}| \qquad (2)$$

where ρ_{deg} and ρ'_{deg} is the degree assortativity before and after the application of the rewiring policy, likewise for ρ_{attr}. We select $\beta \in \{0, 0.5, 1\}$, which mediates the weight between the two, and call the policies Greedy-Attr, Greedy-50, and Greedy-Deg respectively.

Selecting a Rewiring Policy. Among the many rewiring policies available to our model, it is crucial to choose one that generates graphs that contain mixing patterns that are faithful to the original graph. Based on the results from previous experiments, we choose

the Leiden clustering and a size-based node selection strategy with $\mu = 5$. Then we test the five rewiring policies with ten repetitions on each dataset in Table 1.

Figure 8 illustrates the individual results for each trial and a corresponding box and whisker plot for λ-distance [53], $\Delta\rho_{\mathrm{deg}}$, and $\Delta\rho_{\mathrm{attr}}$ *i.e.*, absolute difference between the degree and attribute assortativity of the generated and original graphs, respectively (lower is better). Given the lack of a clear winner among the rewiring policies, we choose the mixing matrix one in the remainder of this work, given its flexibility.

5.3 Observing the Generation Dynamics

Next, we inspect the graph generation process. Observing the growth of the generator provides a unique look at its underlying dynamics. Despite the temptation, it is important to *not* think of this generation process as a temporal one, but rather as a top-down filling-out. As new nodes and edges are added, the size of the graph grows, and we expect that the underlying topological and attribute mixtures will converge near the correct values.

Four views of graph generation on the Texas graph are shown in Fig. 9. In this example, the AVRG was extracted from the Texas graph using Leiden clustering. Rules were extracted using the size-based scoring policy ($\mu = 5$), breaking ties randomly. In this example, the graph generator used the mixing matrix rewiring policy and was requested to grow to 250 nodes instead of 183, *i.e.*, we intentionally specified the new graph to be larger than the original. The difference between blue and red points shows the difference between the entire generated graph (treating terminal and nonterminal nodes as the same) versus just the terminal graph (ignoring nonterminal nodes and edges adjacent to nonterminal nodes). These values will eventually converge. When there are no more nonterminals, graph generation ceases.

Green horizontal lines in Fig. 9 show the graph metrics from the original graph (except λ-distance, where 0 means perfect match). In this example, we asked the generative model to produce a graph with an extra 67 nodes. As rewriting rules are applied, the number of nodes gradually approaches 250. Likewise, the graph metrics converge to near the values of the input graph. The graph metrics remain relatively stable once the input size is reached–indicated by the vertical black line. This example illustrates an important property of AVRGs: they can generate graphs of any large size, yet they continue to resemble the input graph.

6 Graph Generation Results and Findings

We performed a thorough set of experiments that compared the graph generation performance of the AVRG model against several state-of-the-art graph models. Note that, just like in other graph grammar models, applying grammar rules in a tree-ordering over the dendrogram will permit the generation of an isomorphic graph to the input graph. However, this property of AVRGs is not particularly helpful in generating null graphs. Instead, our goal was to generate new graphs and determine if these new graphs shared similar topological and attribute properties with the input graphs.

Methodology. We compared AVRGs against seven state-of-the-art graph generators. Many existing graph generators cannot handle attributed graphs, and many attributed graph generators do not
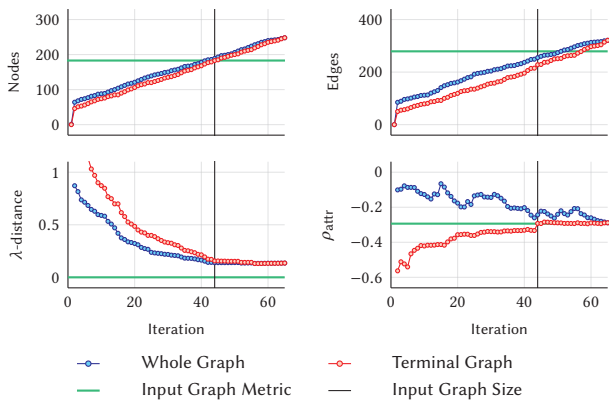
Figure 9: The growth of a graph (Texas) during generation. The green line indicates the corresponding value in the original graph. The generated graph can be made to grow to an arbitrary size; the vertical black line represents the size of the input graph. This example shows how the metrics of the generated graphs slowly approach and eventually converge near ideal value as the generated graph approaches the original graph's size.

Table 1: Dataset summary statistics. $|V|$: # of nodes, $|E|$: # of edges, $|\mathcal{A}|$: # of unique attribute values, ρ_{deg}: degree assortativity coefficient, ρ_{attr}: attribute assortativity coefficient.

Name	$|V|$	$|E|$	$|\mathcal{A}|$	ρ_{deg}	ρ_{attr}
Polbooks	105	441	3	−0.128	0.723
Football	115	613	12	0.162	0.608
Texas	183	279	5	−0.270	−0.294
Cornell	183	277	5	−0.249	−0.079
Wisconsin	251	450	5	−0.193	−0.173
Airports	1,183	22,459	4	0.168	0.356
Polblogs	1,222	16,714	2	−0.221	0.811
Citeseer	2,110	3,668	6	0.007	0.664
Chameleon	2,277	31,371	5	−0.200	0.032
Cora	2,485	5,069	7	−0.071	0.764
Squirrel	5,201	198,353	5	−0.227	0.007
Film	7,600	26,659	5	−0.047	0.003

scale to large sizes. The AVRG model can easily scale to extremely large graphs, but we opt to compare against other attributed models necessitating smaller graphs in this evaluation. Specifically, we compared against graph neural network models NetGAN [7], Graph Autoencoders [22], and Variational Autoencoders [42]. We also compared with classical, non-neural models such as CELL [35], the Degree Corrected Stochastic Block Model (DC-SBM) [20], AGM [32], and the base Chung-Lu model [10]. We used the best-performing hyperparameter settings for each of the baseline models.

All experiments were repeated ten times. The mean average of each metric for each model is reported. The 95% confidence intervals are not shown for clarity but usually lie within 5% of the mean. The AVRG model used Leiden clustering algorithm to build the dendrogram, the size-based node selection strategy with $\mu = 5$, and the Mixing Matrix rewiring policy for all results in this section.
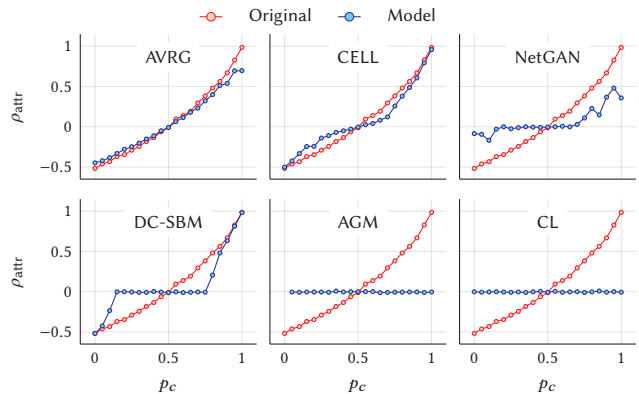

Figure 10: Results on matching simulated scale-free graphs: AVRG best hugs the assortativity curve (CELL is a close contender) across the spectrum, while other models struggle.

6.1 Case Study II: CABAM Simulations

The CABAM model simulates scale-free graphs with varying attribute assortativity via a modified, attribute-aware preferential attachment process [38]. We generate graphs with 500 nodes and degree distribution power-law exponent ∼3, and vary the interclass assortativity (p_c) in increments of 0.05. Figure 10 shows that for a majority of the attribute assortativity space, AVRG with the Hyperbolic hierarchical clustering (Leiden produces slightly worse results) can produce graphs matching the true values. We observe that AVRGs and CELL are the standout performers, both tightly hugging the assortativity curve. Other graph models, including NetGAN (surprisingly), do poorly on this task. These results demonstrate that AVRG is capable of handling graphs across the spectrum of attribute assortativity.

6.2 Real-World Graphs

We evaluated the generation quality of the models by comparing both topological and attribute similarities with the original graphs.

We choose λ-distance to compare the topological fidelity instead of comparing degree and PageRank distributions because it is known to be more sensitive to topological variations [53]. We also computed the absolute difference between degree and attribute assortativity values to quantify similarity in mixing patterns. Finally, we counted all *connected* and *colored* 2, 3, and 4-node graphlets (*e.g.*, edges, triangles, squares, kites) [52]. We used the inverse of Pearson's correlation coefficient to compare each colored graphlet count vector (the inverse is taken so that lower is better−consistent with our other measures) as a way of measuring higher-order attribute mixing patterns. Because of the enormous number of possible colored graphlets, some of these counts could not be computed for the larger graphs.

The results on all 12 input graphs are aggregated in Tab. 2. Some models failed due to a combination of memory and model fitting issues. However, we can make the following observations.

CELL and AVRG are the standout performers, both consistently finishing in the top 2. CELL, being the state-of-the-art, performed

Table 2: Graph generation performance. Lower is better. 95% confidence intervals are not shown for clarity. The best and second best performing models are marked in boldface and underline respectively. Failures are marked with −. λ-dist: topological similarity, $\Delta\rho_{\mathbf{deg}}$ and $\Delta\rho_{\mathbf{attr}}$: difference in degree and attribute assortativities. 1 - r: Inverse Pearson Correlation Coefficient of colored graphlet counts. AVRG ranks in the top-2 for at least one metric across all datasets except Cora.

	Polbooks				Football				Texas				Cornell			
	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r
AGM	0.087	0.062	**0.058**	0.165	–	–	–	–	–	–	–	–	–	–	–	–
CL	<u>0.069</u>	0.075	0.729	1.172	0.178	0.176	0.618	1.192	0.265	0.097	0.276	0.894	0.226	0.091	0.058	0.723
DC-SBM	0.194	<u>0.037</u>	0.115	1.154	0.254	0.142	0.197	1.129	0.221	0.051	0.253	1.366	0.208	0.056	0.052	1.305
NetGAN	0.192	**0.030**	0.112	**0.101**	0.189	0.159	0.170	0.261	0.203	0.049	0.173	0.822	0.153	<u>0.023</u>	<u>0.037</u>	0.453
GAE	3.014	0.093	0.280	–	2.430	0.208	0.444	–	5.584	0.592	0.326	–	5.916	0.671	0.062	–
GVAE	3.826	0.081	0.259	–	3.454	0.199	0.503	–	6.433	0.754	0.302	–	6.235	0.754	0.093	–
CELL	**0.058**	0.164	<u>0.069</u>	0.161	0.125	0.086	0.103	0.198	<u>0.132</u>	0.038	<u>0.048</u>	0.591	0.085	**0.021**	**0.032**	**0.234**
AVRG	0.229	0.080	0.171	<u>0.148</u>	**0.051**	**0.056**	**0.022**	**0.075**	0.118	0.036	0.030	0.504	**0.081**	0.031	0.042	<u>0.346</u>

	Wisconsin				Airports				Polblogs				Citeseer			
	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r
AGM	–	–	–	–	<u>0.185</u>	0.211	<u>0.048</u>	**0.021**	0.224	0.142	**0.024**	0.006	0.022	**0.021**	**0.012**	**0.054**
CL	0.195	0.070	0.147	0.748	**0.154**	0.226	<u>0.359</u>	0.654	<u>0.150</u>	0.159	0.814	1.125	**0.015**	0.026	0.669	1.050
DC-SBM	0.150	<u>0.035</u>	0.139	1.137	0.523	<u>0.025</u>	<u>0.048</u>	0.650	<u>0.291</u>	**0.023**	0.032	1.220	<u>0.018</u>	<u>0.022</u>	0.314	1.023
NetGAN	0.198	<u>0.035</u>	0.115	0.461	0.589	0.148	0.152	**0.021**	0.323	0.154	0.367	0.186	0.032	0.051	0.602	0.545
GAE	6.595	<u>0.451</u>	0.182	–	13.518	0.029	0.154	–	16.025	0.167	0.218	–	20.260	0.027	0.612	–
GVAE	7.216	0.547	0.190	–	13.685	0.226	0.183	–	16.079	0.142	0.222	–	20.764	0.059	0.609	–
CELL	**0.111**	0.025	<u>0.029</u>	**0.110**	0.194	**0.011**	**0.038**	0.650	**0.107**	<u>0.026</u>	<u>0.028</u>	**0.003**	0.024	0.058	<u>0.200</u>	<u>0.073</u>
AVRG	<u>0.136</u>	**0.025**	**0.020**	<u>0.244</u>	0.447	0.041	0.063	**0.021**	0.357	0.083	0.283	0.123	<u>0.018</u>	0.034	0.263	0.101

	Chameleon				Cora				Squirrel				Film			
	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r	λ-dist	$\Delta\rho_{\text{deg}}$	$\Delta\rho_{\text{attr}}$	1 - r
AGM	0.196	0.125	0.036	–	0.026	0.032	**0.013**	<u>0.255</u>	0.521	0.083	0.008	–	0.047	0.020	<u>0.004</u>	–
CL	<u>0.174</u>	0.129	0.035	–	**0.015**	0.046	0.766	<u>1.114</u>	0.472	0.095	0.007	–	0.039	0.019	**0.003**	–
DC-SBM	0.361	0.017	<u>0.012</u>	–	0.024	<u>0.020</u>	0.292	1.078	0.835	<u>0.034</u>	<u>0.002</u>	–	0.030	0.008	<u>0.004</u>	–
NetGAN	0.332	0.129	<u>0.029</u>	–	0.058	<u>0.036</u>	0.710	1.089	1.041	0.219	0.008	–	–	–	–	–
GAE	–	–	–	–	22.364	0.066	0.688	–	–	–	–	–	–	–	–	–
GVAE	–	–	–	–	21.860	0.058	0.686	–	–	–	–	–	–	–	–	–
CELL	0.260	**0.003**	0.017	–	<u>0.022</u>	**0.004**	<u>0.132</u>	**0.067**	**0.303**	**0.030**	**0.001**	–	<u>0.023</u>	**0.001**	<u>0.004</u>	–
AVRG	**0.072**	<u>0.010</u>	**0.007**	–	0.026	0.042	0.374	0.263	<u>0.356</u>	0.067	0.006	–	0.025	<u>0.006</u>	**0.003**	–

admirably across all the datasets. AVRG does particularly well on disassortative graphs like Texas, Wisconsin, and Chameleon, in capturing topological and attribute mixing patterns. NetGAN, DC-SBM, AGM, and CL perform similarly well, while the autoencoders fail to reproduce the graphs faithfully.

7 Conclusions and Future Work

The present work describes the attributed vertex replacement grammar (AVRG) model inspired by the context-free grammar formalism widely used in compilers and natural language processing. We described how an AVRG can be extracted from a hierarchical clustering of an attributed graph and then show that the model successfully encodes local topological structures. Starting with an empty graph, if we apply grammar rules stochastically, the model can generate a new graph. We show that the newly generated graphs are similar to the input graph, both topologically and in attribute mixing patterns.

Despite their straightforward design (*i.e.*, lacking supervision and a rich neural architecture), we demonstrate that AVRGs can often surpass state-of-the-art deep graph neural network generators' performance. We further demonstrate that the attributed graph rewriting rules encoded by the grammar can be interpreted to understand the local mixing patterns of the graph and how these substructures are pieced together.

These results open an exciting avenue for further exploration into attributed rewriting rules. One of the principal strengths of AVRGs is their ability to find the same rule repeatedly. For attributed graphs, combining identical grammar rules via rule isomorphism becomes less likely. As the number of unique attributes grows, the model size is likely to grow as well. Combining similar rules using graph kernel methods or other formalisms borrowed from formal language theory may be a particularly compelling avenue for further research. Additional work is also needed to extract actionable knowledge and interpret the simple patterns encoded in the grammar. Such tools could also be used to better understand specific graph datasets such as knowledge graphs, biological networks, and other complex systems containing a rich mixture of attributes and their patterns. We also intend to apply these techniques to solve additional downstream tasks like link prediction, graph summarization, and other critical graph modeling tasks.

Acknowledgments

References

[1] Salvador Aguiñaga, David Chiang, and Tim Weninger. 2018. Learning Hyperedge Replacement Grammars for Graph Generation. *TPAMI* (2018).

[2] Salvador Aguiñaga, Rodrigo Palacios, David Chiang, and Tim Weninger. 2016. Growing Graphs from Hyperedge Replacement Graph Grammars. In *CIKM*. ACM.

[3] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *ICDM*. IEEE.

[4] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*. Springer.

[5] Rami Al-Rfou, Bryan Perozzi, and Dustin Zelle. 2019. Ddgk: Learning graph representations for deep divergence graph kernels. In *WWW*. 37–48.

[6] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *JSM* (2008).

[7] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. NetGAN: Generating Graphs via Random Walks. In *ICML*.

[8] Ines Chami, Albert Gu, Vaggos Chatziafratis, and Christopher Ré. 2020. From Trees to Continuous Embeddings and Back: Hyperbolic Hierarchical Clustering. *arXiv preprint arXiv:2010.00402* (2020).

[9] Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. 2014. Can cascades be predicted?. In *WWW*.

[10] Fan Chung and Linyuan Lu. 2002. The average distances in random graphs with given expected degrees. *PNAS* (2002).

[11] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *PAMI* (2004).

[12] Sanjoy Dasgupta. 2016. A cost function for similarity-based hierarchical clustering. In *STOC*.

[13] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. 135–144.

[14] Yuxiao Dong, Reid A Johnson, Jian Xu, and Nitesh V Chawla. 2017. Structural diversity and homophily: A study across more than one hundred big networks. In *KDD*. 807–816.

[15] Hongchang Gao and Heng Huang. 2018. Deep attributed network embedding. In *IJCAI*.

[16] Gösta Grahne and Jianfei Zhu. 2005. Fast algorithms for frequent itemset mining using fp-trees. *TKDE* (2005).

[17] Lars Hagen and Andrew B Kahng. 1992. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* (1992).

[18] Justus Hibshman, Satyaki Sikdar, and Tim Weninger. 2019. Towards interpretable graph modeling with vertex replacement grammars. In *BigData*. IEEE.

[19] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *KDD*.

[20] Brian Karrer and Mark EJ Newman. 2011. Stochastic blockmodels and community structure in networks. *Phys. Rev. E* (2011).

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *ICLR* (2016).

[22] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[23] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker graphs: An approach to modeling networks. *JMLR* (2010).

[24] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annu. Rev. Sociol.* (2001).

[25] Stephen Mussmann, John Moore, Joseph J Pfeiffer, and Jennifer Neville III. 2014. Assortativity in Chung Lu random graph models. In *Workshop on Social Network Mining and Analysis*. ACM.

[26] Mark EJ Newman. 2003. Mixing patterns in networks. *Physical review E* (2003).

[27] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *NeurIPS*. 849–856.

[28] Hamed Nilforoshan and Neil Shah. 2019. Slicendice: mining suspicious multi-attribute entity groups with multi-view graphs. In *DSAA*. IEEE.

[29] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. 2014. Focused clustering and outlier detection in large attributed graphs. In *KDD*.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*.

[31] Joseph J Pfeiffer, Timothy La Fond, Sebastian Moreno, and Jennifer Neville. 2012. Fast generation of large scale social networks while incorporating transitive closures. In *Workshop on Privacy, Security, Risk and Trust*. IEEE.

[32] Joseph J Pfeiffer III, Sebastian Moreno, Timothy La Fond, Jennifer Neville, and Brian Gallagher. 2014. Attributed graph models: Modeling network structure with correlated attributes. In *WWW*. 831–842.

[33] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E* (2007).

[34] Erzsébet Ravasz and Albert-László Barabási. 2003. Hierarchical organization in complex networks. *Phys. Rev. E* (2003).

[35] Luca Rendsburg, Holger Heidrich, and Ulrike Von Luxburg. 2020. NetGAN without GAN: From Random Walks to Low-Rank Approximations. In *International Conference on Machine Learning*. PMLR.

[36] Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. 2007. An introduction to exponential random graph (p*) models for social networks. *Social networks* (2007).

[37] Martin Rosvall, Daniel Axelsson, and Carl T Bergstrom. 2009. The map equation. *The European Physical Journal Special Topics* (2009).

[38] Neil Shah. 2020. Scale-Free, Attributed and Class-Assortative Graph Generation to Facilitate Introspection of Graph Neural Networks. In *SIGKDD Workshop on Mining and Learning with Graphs*.

[39] Neil Shah, Alex Beutel, Bryan Hooi, Leman Akoglu, Stephan Gunnemann, Disha Makhija, Mohit Kumar, and Christos Faloutsos. 2016. Edgecentric: Anomaly detection in edge-attributed networks. In *ICDMW*. IEEE.

[40] Satyaki Sikdar, Daniel Gonzalez, Trenton Ford, and Tim Weninger. 2020. The Infinity Mirror Test for Graph Models. *arXiv preprint arXiv:2009.08925* (2020).

[41] Satyaki Sikdar, Justus Hibshman, and Tim Weninger. 2019. Modeling graphs with vertex replacement grammars. In *ICDM*. IEEE.

[42] Martin Simonovsky and Nikos Komodakis. 2018. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*. Springer.

[43] Yu Su, Fangqiu Han, Richard E Harang, and Xifeng Yan. 2016. A fast kernel for attributed graphs. In *SDM*. SIAM, 486–494.

[44] Yizhou Sun and Jiawei Han. 2013. Mining heterogeneous information networks: a structural analysis approach. *SIGKDD Explorations* (2013).

[45] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB* (2011).

[46] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient subgraph matching on billion node graphs. *VLDB* (2012).

[47] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*.

[48] Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. 2007. Fast best-effort pattern matching in large attributed graphs. In *KDD*.

[49] V. A. Traag, L. Waltman, and N. J. van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports* (2019).

[50] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. 2012. Structural diversity in social contagion. *PNAS* (2012).

[51] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR* (2018).

[52] Sebastian Wernicke and Florian Rasche. 2006. FANMOD: a tool for fast network motif detection. *Bioinformatics* (2006).

[53] Richard C Wilson and Ping Zhu. 2008. A study of graph spectra for comparing graphs and trees. *Pattern Recognition* 41, 9 (2008), 2833–2841.

[54] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? *ICLR* (2019).

[55] Xifeng Yan and Jiawei Han. 2002. gspan: Graph-based substructure pattern mining. In *ICDM*. IEEE.

[56] Carl Yang, Xiaolin Shi, Luo Jie, and Jiawei Han. 2018. I know you'll be back: Interpretable new user clustering and churn prediction on a mobile social application. In *KDD*.

[57] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *ICML*.

[58] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2010. Clustering large attributed graphs: An efficient incremental approach. In *ICDM*. IEEE, 689–698.

[59] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. 2020. Graph Neural Networks with Heterophily. *arXiv preprint arXiv:2009.13566* (2020).

[60] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. *Neurips* 33 (2020).