Modeling Graphs with Vertex Replacement Grammars

Satyaki Sikdar Justus Hibshman Tim Weninger
Department of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN, USA
{ssikdar,jhibshma,tweninge}@nd.edu

***Abstract*—One of the principal goals of graph modeling is to capture the building blocks of network data in order to study various physical and natural phenomena. Recent work at the intersection of formal language theory and graph theory has explored the use of graph grammars for graph modeling. However, existing graph grammar formalisms, like Hyperedge Replacement Grammars, can only operate on small tree-like graphs. The present work relaxes this restriction by revising a different graph grammar formalism called Vertex Replacement Grammars (VRGs). We show that a variant of the VRG called Clustering-based Node Replacement Grammar (CNRG) can be efficiently extracted from many hierarchical clusterings of a graph. We show that CNRGs encode a succinct model of the graph, yet faithfully preserves the structure of the original graph. In experiments on large real-world datasets, we show that graphs generated from the CNRG model exhibit a diverse range of properties that are similar to those found in the original networks.**

***Index Terms*—vertex replacement grammar, graph model, graph generators**

I. Introduction

We consider the task of identifying the informative and interesting patterns found in graphs. Because of their ability to represent natural phenomena, graphs have been studied extensively in various computing and scientific scenarios. Arguably the most prescient task in the study of graphs is the identification, extraction, and representation of the small substructures that, in aggregate, describe the underlying phenomenon encoded by the graph. These extracted models contain the LEGO-like building blocks of real-world graphs, and their overarching goal is to enable in-depth scientific analysis and make predictions about the data.

Because of the prevalence of relevant data and the importance of this line of inquiry, there exists a large body of prior work in graph mining. Rooted in data mining and knowledge discovery, subgraph mining methods have been developed to identify frequently occurring subgraphs [12, 17]. Unfortunately, these early methods have a so-called "combinatorial explosion" problem [45] wherein the search space grows exponentially with the pattern size. This causes computational headaches and can also return a massive result set that hinders real-world applicability. Recent work that heuristically mines graphs for prominent or representative subgraphs have been developed in response, but are still limited by their choice of heuristic [48, 34, 28, 43]. Alternatively, researchers characterize a network by counting small subgraphs called graphlets and therefore forfeit any chance of finding larger, more interesting structures[37, 30, 3].

Graph generators, like frequent subgraph mining, also find distinguishing characteristics of networks, but go one step further by generating new graphs that "look like" the original graph(s). What a graph looks like includes local graph properties like the counts of frequent subgraphs, but can also include global graph properties like the degree distribution, clustering coefficient, diameter, and assortativity metrics among many others. Early graph generators had parameters that could be tuned to generate graphs with specific desirable properties. Additional work in exponential random graphs [40], Kronecker graphs [26, 7], Chung-Lu graphs [8], Stochastic Block Models (SBMs) [18], and their many derivatives [36, 31, 4, 32, 21] create a model from some example graph in order to generate a new graph that has many of the same properties as the original graph.

These graph models look for small pre-defined patterns or frequently reoccurring patterns, even though interesting and useful information may be hidden in latent and infrequent patterns. Principled strategies for extracting these complex patterns are needed to discover the precise mechanisms that govern network structure and growth.

Recent advances in neural networks have produced graph generators based on recurrent neural networks [49], variational autoencoders [42], and generative adversarial networks [6] each of which have their advantages and disadvantages, which we explore later. Generally speaking, these neural network models are excellent at generating faithful graphs but struggle to provide a descriptive (*i.e.*, explainable) model from which in-depth scientific or data analysis can be performed.

The present work describes CNRG: a **C**lustering-based **N**ode **R**eplacement **G**rammar (pronounced: "synergy") a variant of a vertex replacement grammar (VRG), which contains graphical rewriting rules that can match and replace graph fragments similar to how a context-free grammar (CFG) rewrites characters in a string. These graph fragments represent a succinct description of the building blocks of the network, and the rewiring rules of the CNRG describe the instructions about how the graph is pieced together.

Prior work has investigated the relationship between graph

theory and formal language theory by extracting Hyperedge Replacement Grammars (HRGs) from the tree decomposition of a graph [1]. The HRG framework can extract patterns from small samples of the graph and can generate networks that have properties that match those of the original graph [2]. In their typical use-case, HRGs are used to represent and generate graph patterns through hyperedge rewriting rules, where a nonterminal edge in the graph is matched with a left-hand-side (LHS) rule in the HRG and replaced with its corresponding right-hand-side (RHS). The composition of an HRG-rule is entirely dependent on the graph's tree decomposition. Unfortunately, finding an optimal tree decomposition is both NP-complete and non-unique. Heuristic tree decomposition algorithms exist but still do not scale to even moderately sized graphs. Furthermore, non-tree like graphs (*i.e.*, graphs with high treewidth) will produce large, clunky grammar rules that are difficult to interpret.

Like HRGs, VRGs have previously been used to model graph processes and generate graphs. Rather than replacing nonterminal (hyper)edges with RHS-subgraphs, a VRG replaces *vertices* with RHS-subgraphs. VRGs represent an interesting complement to HRGs, but there currently does not exist a means by which to extract a VRG from a graph automatically. Instead, graph modelers must craft these grammars by hand, which is a time-consuming process and introduces human bias into the process. We desire an automatic, scalable, and interpretable extraction algorithm that compactly models the various structures found in the graph.

The present work describes such an algorithm[1] that automatically extracts a CNRG from any graph. Critically, the extraction algorithm does not require a tree decomposition. This permits the extractor to be both scalable and immune to problems arising with non-treelike graphs. The output of the CNRG extractor is a graph model with CFG-like production rules. We show that the graph model is able to compress the graph better than state-of-the-art graph summarization models and generate graphs more faithfully than many state-of-the-art graph generation methods.

II. Preliminaries

We begin with a short introduction to the graph grammar formalism and define important terms that are used throughout the remainder of the present work.

Labeled multigraphs. A labeled multigraph is a 4-tuple $H = \langle V, E, \kappa, L \rangle$ 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 edges. By default, each edge has a multiplicity value of 1. Although the CNRG model can be used for directed graphs, the present work treats all graphs as undirected for clarity of prose and illustration. We use the terms node and vertex interchangeably in the present work.

Clustering-based Node Replacement Grammars (CNRGs). A CNRG 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 of the form $X \to (R, f)$, where X is the LHS consisting of a nonterminal node (*i.e.*, $X \in \Sigma \setminus \Delta$) with a size ω, and the tuple (R, f) represent the RHS, where R is a labeled multigraph with terminal and possibly nonterminal nodes, and $f \in \mathbb{Z}^+$ is the frequency of the rule, *i.e.*, the number of times the rule appears in the grammar, and $\mathcal{S}$ is the starting graph which is a nonterminal of size 0. This formulation is similar to node label controlled (NLC) grammar [41], except that the CNRG used in the present work does not keep track of specific rewiring conditions. Instead, every internal node in R is labeled by the number of boundary edges to which it was adjacent in the original graph. The sum of the boundary degrees is, therefore, equivalent to ω, which is also equivalent to the label of the LHS.

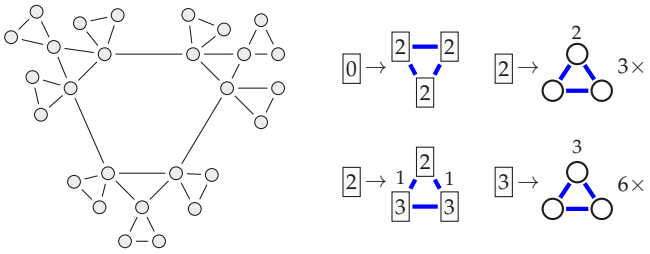

Fig. 1: (A) An example graph can be decomposed into a CNRG. (B) An extracted CNRG containing four distinct rules, each with an LHS and RHS. The LHS is a single nonterminal node drawn as a square labeled with size ω (drawn inside the node). The RHS is a subgraph with nonterminal nodes drawn as squares and labeled (illustrated inside the node), terminal nodes labeled with the number of boundary edges (drawn on top of the node), and connecting edges (which do not have labels in this example). The production rules on the right have $f = 3\times$ and $f = 6\times$ indicating that they occur 3 and 6 times respectively.

A CNRG can be extracted from any graph or hypergraph and may not be unique. That is, one graph may produce many different CNRGs. The goal of the present work is to extract CNRGs that capture the high-order structure of the graph. The example in Fig. 1 shows an example graph and an example grammar that can be extracted from it. In this example, the original graph appears to have a regular structure akin to a recursively arranged triangle of triangles. The extracted grammar represents this triangle of triangles pattern, which is represented by the grammar rules.

Like their HRG cousins [1], the extracted CNRG may also be used to generate graphs that are similar to (or contain similar high-level structures as) the original graph.

Model size. One way to compare the conciseness of a grammar is by analyzing its size. For this task, we define a description length (abbreviated as DL) for graphs and grammars following prior work. Given a labeled multigraph H

[1]Source code can be found at the Github repository.

defined above, we compute its size in the following way. Let $\lg(\cdot)$ denote $\log_2(\cdot)$. Our approach is similar to that of Cook and Holder [9] except that (i) we use Elias γ [10] encoding instead of the Quinlan & Rivest encoding [38], and (ii) we directly encode the multiplicity matrix M instead of encoding a binary adjacency matrix A and its associated multiplicity matrix M separately. First, $\lg|V|$ and $\lg|L|$ bits are required to encode the number of vertices and the number of labels in H respectively. Hence, the total number of bits required to encode all the labeled vertices (v) is $v = \lg|V| + |V| \cdot \lg|L|$ bits. Second, let M be a $|V| \times |V|$ multiplicity matrix where $M_{ij} = \kappa(i,j)$ for $(i,j) \in E$, and 0 otherwise. We add 1 to each element of M to use the γ-code, which can only encode positive integers. Hence, the total number of bits required to encode all the labeled edges (e) is $e = \lg|E| + \lg|L| \cdot \sum_{ij} |\gamma\text{-code}(M_{ij})|$ bits. Therefore, the description length $(DL(H))$ of the graph H is $DL(H) = (v + e)$ bits.

Like the graph H, the CNRG G is also given a description length. Each rule (P) is of the form $X \rightarrow (R, f)$, where X is a nonterminal of size ω, R is a labeled (multi)graph, and f is the frequency. We encode the nonterminal size ω and the frequency f using the γ-code. Mathematically, the description length $(DL(l_P))$ of the LHS is given by $DL(l_P) = |\gamma\text{-code}(\omega)| + |\gamma\text{-code}(f)|$ bits.

Similarly, we define a description length $(DL(r_P))$ for the RHS. The labeled (multi)graph R is encoded similar to H; additionally, we have to include the γ-encoding of the individual boundary degrees (abbreviated as b_deg) of the nodes in V_R. So, we have $DL(r_P) = |\gamma\text{-code}(R)| + \sum_{v \in V_R} |\gamma\text{-code}(\text{b\_deg}(v))|$ bits. Therefore, the description length $(DL(G))$ of the CNRG G is given by $DL(G) = \sum_P (DL(l_P) + DL(r_P))$ bits.

With these definitions formally stated we can more-concretely restate the task: given a (multi)graph H, we seek to extract a CNRG G that succinctly and thoroughly encodes H. A byproduct of extracting such a graph grammar is that the production rules may also serve as a succinct representation of the constituent structures found in the original graph.

III. Extracting Vertex Replacement Grammars

As discussed earlier, many possible CNRGs can represent the same original graph. An optimal CNRG ought to represent the original graph succinctly (*i.e.*, with as few bits as possible) and faithfully (*i.e.*, without losing any information). Unfortunately, such an optimal lossless compression is not possible in all cases. Instead, we assume that H can be clustered hierarchically [39] and that regular substructures can be extracted as rules.

The remainder of this section describes the details of several CNRG extraction methods and uses the minimum description length principle to extract a grammar.

A. Hierarchical Graph Clustering

We begin with a labeled (multi)graph H. We first compute a dendrogram from H using a hierarchical clustering algorithm. We explored the Leiden method [46], the Louvain method [5], recursive spectral bipartion [15], and hierarchical spectral k-means [33]; however, any hierarchical clustering method may be used here.

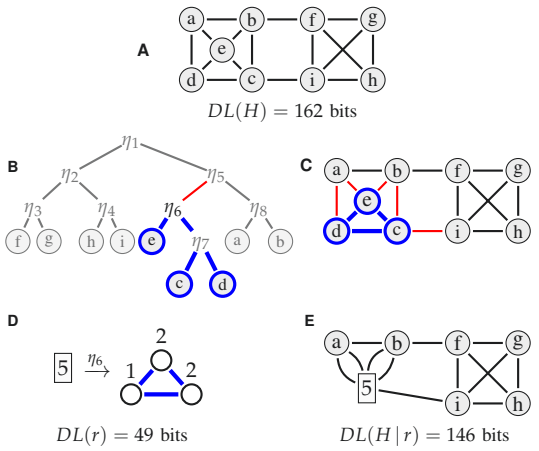

Fig. 2: (A) Original graph H. (B) Dendrogram created from a hierarchical clustering algorithm, leaves of this dendrogram are nodes of H. (C) Subtree η_6 is selected. Leaf nodes and edges of the induced subgraph are drawn in blue; boundary edges are drawn in red. (D) Rule η_6 extracted from the H. LHS is a nonterminal labeled by ω=5; RHS is the induced subgraph of the nodes in η_6 labeled with their boundary condition (*i.e.*, number of boundary (red) edges present). (E) New graph H' with η_6 removed and replaced by a nonterminal node $\boxed{5}$.

As a running example, we introduce a 9-node, 16-edge undirected graph in Fig. 2(A). Applying the recursive spectral clustering algorithm on this graph results in the dendrogram shown in Fig. 2(B). Non-leaf nodes of the dendrogram are represented as η_i, and the leaves are nodes from the original graph. We see that the dendrogram computed from the example graph correctly separates the left and right sides of the graph. A similar dendrogram is produced when other clustering algorithms are applied.

B. Rule Extraction

Given an initial dendrogram D computed by applying a hierarchical clustering algorithm on H, the next step is to generate a graph grammar G. The summary of the rule extraction process is as follows: (a) create production rules from D, (b) find the best scoring rule and add it to the grammar G, (c) contract the respective subgraphs to create a reduced graph H', and update D to reflect those changes. Finally, set $H \leftarrow H'$ and repeat until D is empty.

Creating a Grammar Rule. Each internal node $\eta \in D$ corresponds to a grammar rule $r_\eta : X \rightarrow (R, f)$. Let V_η represent the leaf nodes in the subtree rooted at η, which correspond to nodes in graph H. Let b_η represent the set of *boundary edges*, *i.e.*, edges in H which have exactly one endpoint in V_η, and let $\omega = |b_\eta|$. b_η is used to compute the boundary degrees of the nodes in V_η.

We set X to be a nonterminal node of size ω as the LHS of the new production rule. The RHS of the new production

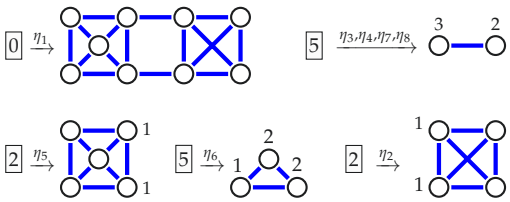

Fig. 3: All possible rules that can be extracted from the dendrogram in Fig. 1B labeled by their corresponding subtrees $\eta_{1\cdots8}$.

rule in the CNRG formalism is a labeled multigraph $R \subseteq H$ with rule frequency f. Let $R = \langle V_R, E_R, \kappa_R, L_R \rangle$ where $V_R = V_\eta$; $E_R = \{(u,v) \mid u \in V_\eta \wedge v \in V_\eta \wedge (u,v) \in E\}$; $\kappa_R(e) = k$, where k is the multiplicity of edge $e \in E_R$; $L_R = \{\text{internal node, internal edge}\}$. If this newly generated rule already exists in the grammar, then the frequency of that rule will be incremented by 1 (instead of storing duplicate rules). Note that this leads to the creation of a many-to-one mapping between the non-leaf nodes of the dendrogram and the rules. Finally, each subtree, and consequently, each rule, is assigned a score (s_η) which is used for selection. The details of the scoring functions are discussed in Sec. III-C.

Returning to the running example, Fig. 3 shows all possible rules that can be constructed from the dendrogram introduced in Fig. 2(B). Considering every possible production rule at every step becomes computationally intractable for medium and large-sized graphs. We also observe that certain internal nodes towards the top of the dendrogram cover many leaf nodes and therefore tend to create production rules with large RHSs. Production rules with large RHSs do not align with our aim of finding small, but topologically meaningful building blocks of the graph. So, to prune the search space and restrict the size of the RHS of the rules, we introduce a subtree restriction parameter μ that removes subtrees larger than μ from consideration.

Selecting the Best Scoring Rule. From all rules r_η, we pick the rule r^*_η with the *minimum* score and add it to the G, updating the necessary alphabet Σ, terminal nodes Δ, and production rules $\mathcal{P}$ as needed. Note that multiple subtrees of the dendrogram may correspond to the same rule. For example, in Fig. 3 subtrees η_3, η_4, η_7, and η_8 all correspond to the same rule.

Updating the Data Structures. Once a production rule is created from the dendrogram, the next step is to create H' by contracting H by removing the RHS subgraph and inserting the new nonterminal node.

Let $H' = H$ initially. For a selected η^*, we remove V_η from H', and insert a new nonterminal node X labeled with ω (from the first step). We connect X to the rest of the graph through the set of boundary edges in R where edges that were connected to V_η are redirected to connect to X. Note, this may lead to the creation of multi-edges in the new graph. H' is now strictly smaller than H and contains new nonterminal nodes.

With a new (smaller) H', it may be prudent to re-run the clustering algorithm and draw a new dendrogram. However, in our initial experiments, we found that re-clustering is time consuming and rarely results in significant changes to the dendrogram. Instead, we simply modify D by replacing the subtrees in η^* with nonterminal nodes X labeled with ω. Scores are also updated as needed based on the new graph.

Finally, we set $H \leftarrow H'$ and repeat this process until the dendrogram is empty.

C. Scoring Functions

The choice of scoring function directly impacts the choice of η^*, which directly impacts the extracted CNRG. Again note that we ignore all η where $|V_\eta| > \mu$. The simplest case is to set $s_\eta = |V_\eta| - \mu$. But this simple case results in many ties which need to be broken. For this task we consider three policies:

- *Random tiebreaking.* Pick η^* at random from candidates equi-distant to μ.
- *Greedy DL.* Break ties by picking η^* that minimizes the overall DL of the grammar. Minimizing the DL of the grammar is akin to finding a rule that already exists in the grammar, or by selecting the rule that has the smallest description length among all candidates according to the description length calculation described in Sec. II. This is more computationally expensive than other policies because it requires the DL computation for each candidate η. Among η's with equal DL, ties are broken arbitrarily.
- *Greedy Level.* Break ties by picking η^* that is at the highest level in the dendrogram. This results in the creation of fewer rules, because a larger portion of the dendrogram, and consequently the graph, is contracted at each step. Among subtrees with equal level, ties are broken arbitrarily.
- *Greedy level + DL.* Break ties by picking η^* using the Greedy Level policy first and then by using the DL.

Previous work suggests that crude two-part MDL [14] is a useful principle for selecting model parameters [22, 9]. Therefore, the next policies to select η^* mimic this. Specifically, let $s_\eta = DL(r_\eta) + DL(H \mid r_\eta)$, which is the sum of the DL of the rule and the DL of H compressed by r_η respectively.

Based on this idea, our next task is to calculate $DL(H \mid r_\eta)$. One important consideration is the case where multiple subtrees map to the same rule. Again consider the example from Fig. 3 where the subtrees η_3(f, g), η_4(h, i), η_7(c, d), and η_8(a, b) are all encoded in the same rule r. With this in mind, two strategies are evident to us: *local MDL* and *global MDL*. In the local MDL strategy, we calculate the scores of each η independently, without regard to other subtrees which result in identical rules. In the global strategy, we recognize that identical rules can be compressed together and therefore calculate $DL(H \mid r_\eta)$ such that all isomorphic r_η's are compressed and stored simultaneously. In global MDL strategy, η^* is not a single rule, but rather a set of isomorphic rules that are compressed together.

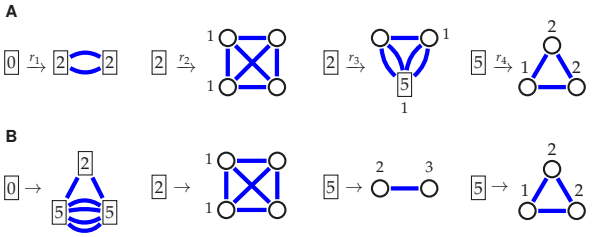

Fig. 4: CNRGs obtained from Fig. 2(B) with $\mu = 4$ using (A) the Local MDL strategy and (B) the Global MDL strategy.

We hypothesize that the global MDL strategy will perform best, but requires significantly more time to select η^*. Fig. 4 shows complete CNRGs extracted using the Local (A) and Global (B) MDL strategies. The differences in this small example are subtle. It is unclear which is better.

IV. Generating Graphs from Vertex Replacement Grammars

The grammar G encodes information about the original graph H in a way that can be used to generate new graphs. How similar are these newly generated graphs to the original graph? Do they contain similar structures and similar global properties? In this section, we describe how to repeatedly apply rules to generate these graphs.

We use a stochastic graph generating process to generate graphs. Simply put, this process repeatedly replaces nonterminal nodes with the RHSs of production rules until no nonterminals remain.

Formally, a new graph H' starts with $\mathcal{S}$, a single nonterminal node labeled with 0. From the current graph, we randomly select a nonterminal and probabilistically (according to each rule's frequency) select a rule from G with an LHS matching the label ω of the selected nonterminal node. We remove the nonterminal node from H', which breaks exactly ω edges. Next, we introduce the RHS subgraph to the overall graph randomly rewiring broken edges respecting the boundary degrees of the newly introduced nodes. For example, a node with boundary degree of 3 expects to be connected with exactly 3 randomly chosen broken edges. This careful but random rewiring helps preserve topological features of the original network. After the RHS rule is applied, the new graph $\hat{H}$ will be larger and may have additional nonterminal nodes. We set $H' = \hat{H}$ and repeat this process until no more nonterminals exist.

An example of this generation process is shown in Fig. 5 using the rules from Fig. 4(A). We begin with $\boxed{0}$ and apply r_1 to generate a multigraph with two nonterminal nodes and two edges. Next, we (randomly) select the nonterminal on the right and replace it with r_2 containing four terminal nodes and 6 new edges. There is one remaining nonterminal, which is replaced with r_3 containing two terminal nodes, one nonterminal node, and 5 edges. Finally, the last nonterminal node is replaced with r_4 containing three terminal nodes and three edges. The edges are rewired to satisfy the boundary degrees, and we see that $\hat{H} = H$. In this way, the graph generation algorithm creates new graphs. The previous example conveniently picked rules that would lead to an isomorphic copy of the original graph; however, a stochastic application of rules and random rewiring of broken edges is likely to generate various graph configurations.

V. Methodology and Results

Our next task is to evaluate the CNRG model size and its graph generation performance. For size, we measure how the CNRG's description length compares with other graph models. For performance, we measure the accuracy of the stochastic graph generator by comparing the generated graphs with the original graph.

The goal of the first part of this section is to explore the parameter space for CNRG extraction and generation performance. After we select appropriate parameters, we will compare against existing methods.

A. Datasets

Datasets were selected based on their variety and size. Our implementation of the CNRG extractor is memory bound at $O(|V| + |E|)$, but it is computationally very fast. The computational complexity of the extractor varies with the choice of clustering algorithm and extractor policy; the graph generation is in $O(|V| + |E|)$. The CNRG extractor can scale to extremely large graphs. Alternative graph models are unable to scale to the largest available graphs, so we selected graphs that could be compared against existing models.

We selected five medium-sized graphs from various sources. They are listed in Tab. I and were downloaded from KONECT [24] and SNAP [25].

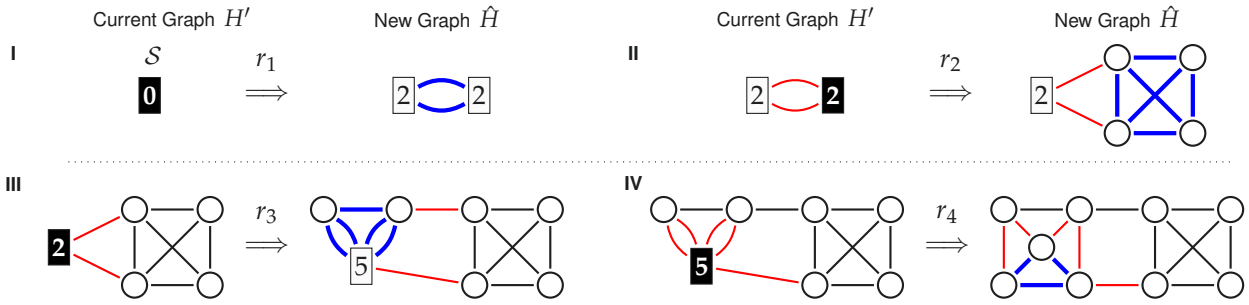

Fig. 5: Generation algorithm. An application of the rules (in tree-order according to D) will regenerate G.

TABLE I: Datasets

Name	$\|V\|$	$\|E\|$
EuCore Emails	986	16,687
PolBlogs	1,222	16,717
OpenFlights	2,905	15,645
ArXiv GrQc	4,158	13,428
Gnutella	6,299	20,776
WikiVote	7,066	100,736
PGP	10,680	24,316

B. Selecting CNRG Parameters

To measure model size, we must first select from the many parameters of the extraction model: clustering algorithm, boundary information, extractor selection heuristic, and RHS size (μ).

The methodology is as follows. We extract a CNRG for each combination of the clustering algorithm, scoring function, and $\mu \in \{2,3,\ldots,10\}$, which equates to 300 different CNRG models for each dataset. To permit statistical tests and confidence intervals, this process is repeated five times for a total of 1,500 CNRG models for each graph.

For k-way recursive Spectral algorithm, we use $k = \sqrt{n/2}$ [29]. The random hierarchical clustering method split the graph into two (nearly) equally-sized but random clusters in a top-down fashion.

Model Size. We define the size of the CNRG as the number of rules present in the grammar and its overall complexity. The number of rules is simply the count of the number of distinct production rules. Usually, the grammar size is sufficient to make decisions about the model. Smaller is better.

The description length (DL) measures the size and complexity of the grammar. CNRGs extracted from graphs of different sizes should not be compared in absolute terms – a large graph will almost certainly have a larger CNRG than a small graph. In order to perform an apples to apples comparison across different dataset sizes, we measure model size using the reciprocal compression ratio: $\mathrm{DL}(G)/\mathrm{DL}(H)$, where $\mathrm{DL}(G)$ is the description length of the CNRG and $\mathrm{DL}(H)$ is the description length of the original graph. Lower is better.

Model Performance. We define the performance of a model as its ability to generate a graph $\hat{H}$ that is similar to the original graph H. There are many ways to compare $\hat{H}$ with H. In the present work we use the spectral distance (λ-distance) [47] and DeltaCon [23].

The λ-distance compares the spectrum of a graph, which is typically defined as the set of eigenvalues $s = \{\lambda_1, \lambda_2, \ldots, \lambda_{|V|}\}$ are are ordered by their magnitude $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{|V|}$. The graph spectrum permits a distance to be calculated:

$$\lambda\text{-distance}(\hat{H}, H) = \sqrt{\sum_i (\hat{s}_i - s_i)^2},$$

where the list of eigenvalues may be zero-padded if they are not the same size.

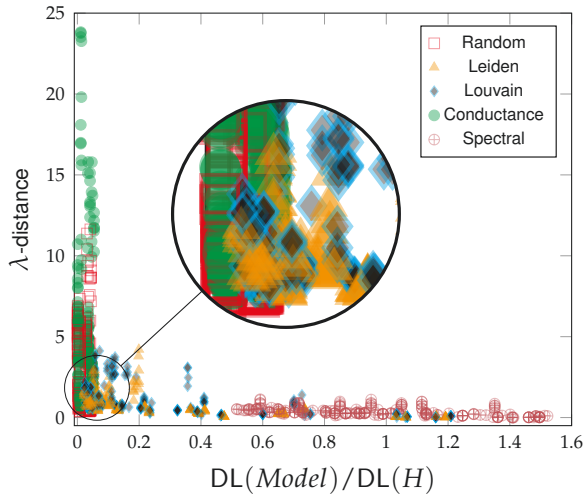

Fig. 6: λ-distance (lower is better) and compression ratio (lower is better) for all runs (all μ, clustering method, η^* selection policy) on all datasets. Results that are consistently in the bottom-left corner are best. Leiden performs the best consistently. This figure is best viewed in color.

DeltaCon measures the difference in node affinities using a belief propagation algorithm. The use of belief propagation implicitly models the diffusion of information throughout the graph and should be able to measure global and local graph structures.

In addition, we count the number of three and four node graphlets [3] that are present in the graph and directly compare these counts. The graphlet correlation distance (GCD) is also used to measure the rank correlation of graphlet orbital counts between nodes in each graph [37, 30, 16].

Because these are all distance metrics, lower is better.

Selecting a Clustering Method. First, we consider the selection of a clustering method. We used Random, Leiden, Louvain, recursive spectral bipartition (*i.e.*, Conductance), and hierarchical spectral k-means (*i.e.*, Spectral) clustering methods. Each clustering method was applied to each dataset using all available datasets, μ-values and η^* selection policies. Each unique configuration was repeated 5 times.

The model size and λ-distance for each graph is plotted in Fig. 6. We observe that Spectral clustering results in remarkably good graph generation, but bad compression. Conversely, Conductance clustering results in remarkably good graph compression, but bad graph generation performance.

As is typical, we generally observe a trade-off between compression and model performance. The Leiden clustering method appears to perform the best in both metrics consistently; so we select Leiden clustering for further analysis.

Selecting an η^* policy. Our next task is to find the η^* selection policy that performs best. Using only the Leiden clustering method, we group all runs (across all μ values) and plot the mean reciprocal compression ratio and λ-distance in Fig. 7. 95% confidence intervals are drawn as error bars.

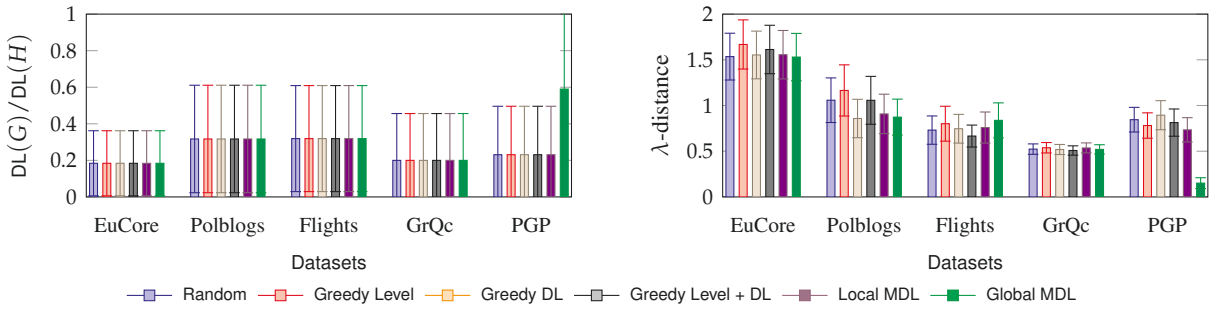

Fig. 7: Mean model size (left) and graph generation performance (right) for each η^* selection policy using Leiden clustering. No clear winner is observed. This figure is best viewed in color.

We observe that the choice of η^* selection policy has minimal effect on the model size and the generation performance. We select Greedy Level + DL and Local MDL because they performed (slightly) better than the other methods, but also because they are much faster to compute than the Global MDL.

Selecting a μ Value. Next, we compare model size and generation performance results for various values of μ. Recall that μ is an upper bound for the number of nodes that appear within a subtree of η^*; *i.e.*, μ is the maximum number of nodes that can appear in any extracted RHS.

We select μ by following the pattern as before. Using the Leiden clustering method and Greedy Level + DL and Local MDL η^* selection policies, we plot the mean reciprocal compression ratio and λ-distance in Fig. 8. 95% confidence intervals are drawn as error bars.

We observe little difference between the η^* selection policies. However, the size-to-performance trade-off becomes evident again as μ varies from small to large. Small values of μ more complex models but more accurate models, while larger values produce less complex models but less accurate models; however, there are quickly diminishing returns as μ increases.

We select a $\mu = 4$ because it appears to generate reasonably small models with reasonable accuracy.

In summary, based on the decisions highlighted in this section we select a parameterization for the CNRG that uses Leiden clustering, the Greedy Level + DL for the η^* selection policy, and $\mu = 4$. We will use these values throughout the remainder of the present work unless otherwise specified.

C. Graph Model Size

Next, we compare the model size of CNRG, parameterized as above, in bits against three other graph models: the Vocabulary-based summarization of Graphs (VoG) [22], SlashBurn [27], and SUBDUE [19]. Like the CNRG model, VoG, SlashBurn, and SUBDUE maintain an encoding of the graph, but their models are constructed in very different ways. VoG summarizes graphs using a fixed vocabulary of structures. SlashBurn recursively splits a graph into hubs and spokes connected only by the hubs. SUBDUE creates a node-grammar model, similar in principle to the CNRG model, by finding substructures that maximally reduce the size (bits) of the graph after each selection. These models are useful for graph summarizing and graph understanding, but do not generate

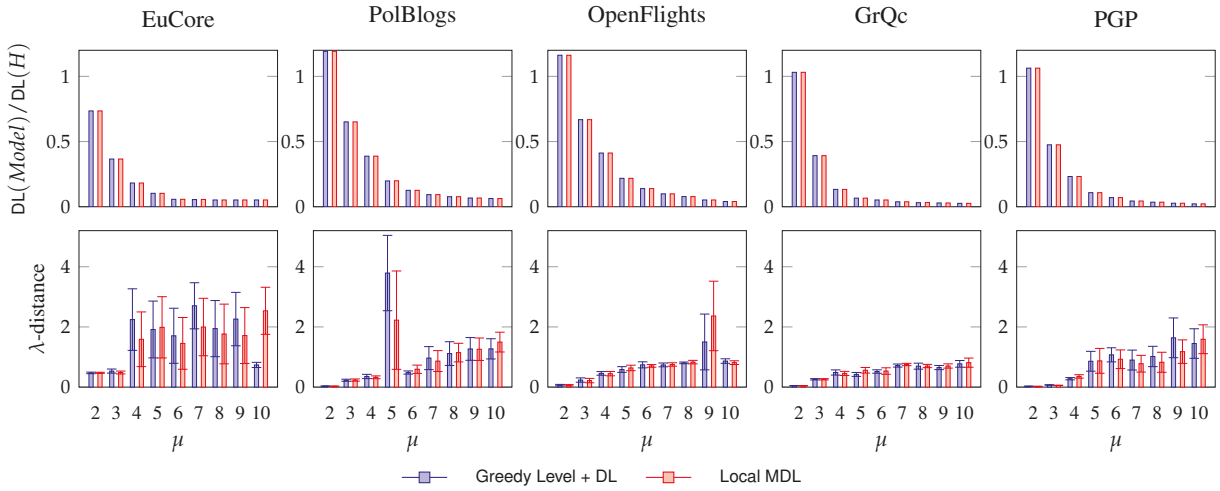

Fig. 8: Mean model size (top), and graph generation performance (bottom) for each μ using Leiden clustering. We select μ=4 as having the best size-to-performance tradeoff. This figure is best viewed in color.

TABLE II: Model Size Comparison. Lower is better.

Graph	$DL(Model)/DL(H)$			
	SUBDUE	SlashBurn	VoG	CNRG
Karate	3.546	1.119	1.080	**0.704**
Dolphins	4.348	1.336	1.026	**0.43**
LesMis	3.546	1.05	**0.875**	0.924
EuCore	–	5.54	0.986	**0.182**
PolBlogs	–	0.873	0.881	**0.388**
OpenFlights	–	0.888	0.869	**0.412**
GrQc	–	1.154	0.851	**0.133**
PGP	–	1.196	0.911	**0.232**
Gnutella	–	1.045	0.967	**0.306**
WikiVote	–	0.839	0.843	**0.525**

graphs; thus, they can only be compared to CNRGs by their model size. Each of the models was run with their default settings.

SUBDUE was unable to process the even the smallest of our graph datasets, so we included three graphs: Karate, Dolphins, and LesMis, representing well known small graphs, in Tab. II. These results show that CNRG almost always produces the best model sizes among the other models. This confirms our hypothesis that CNRG compresses the original graph better than the state-of-the-art methods.

D. Graph Generation Performance

Here we show that the CNRG model represents not only a succinct encoding of the original graph but also a faithful one as well. Keeping a tree ordering over production rules in the CNRG will permit a generation close or isomorphic to the original graph. This is an interesting, but not particularly useful outcome of the CNRG model. Instead, we ask how well the CNRG model generates new graphs. Are these graphs similar to the original graph? How does the CNRG accuracy compare to other graph models at generating graphs?

Graph generators have been studied intently for several years. The idea being that we only truly understand a graph if we can generate it faithfully. Practically speaking, graph generators are often used to create null models for statistical purposes. In a similar vein, graph generators are frequently used to find anomalous patterns in real-world graphs.

Setup. We compare CNRG graph generation against many of the state-of-the-art graph generators. We consider the properties that characterize some real-world networks and compare the distribution of graphs generated using the Kronecker graph model [26], the Block Two-Level Erdős-Rényi (BTER) model [21], Chung-Lu's configuration model [8], the degree corrected Stochastic Block Model (DC-SBM) [18], and the stochastic Hyperedge Replacement Grammar (HRG) model [1, 2].

Like CNRGs, these other graph models learn parameters that can be used to approximately recreate the original graph or a graph of some other size such that the generated graph holds many of the same properties as the original graph. The generated graphs are likely not isomorphic to the original graph. We can, however, still judge how closely the generated

Fig. 9: Relative graphlet[2] counts as a heatmap. Color intensity in each cell indicates *disagreement* between the number of graphlets found in the generated graph and the number of graphlets found in he original graph. CNRG consistently performs the best. The grayed out columns indicate that the method failed to produce graphs. This figure is best viewed in color.

graph resembles the original graph by comparing several of their local and global graph properties.

Exponential Random Graph Models (ERGMs) are another type of graph model that learns a robust graph model from user-defined features of a graph [40]. Unfortunately, this model does not scale well and is prone to model degeneracy. Neural network graph models like GraphVAE [42] and GraphRNN [49] are currently limited in their scalability. Generative adversarial networks (GANs) have been shown to scale to medium-sized graphs and perform on-par with existed methods; however, the model size of NetGAN is many times larger than the graph size [6]. We attempted to compare these methods but were unable to because of problems with either model degeneracy or scalability.

The main purpose of node embedding models like LINE [44], node2vec [13], VGAE [20], and others [11] is to learn vector representations of the nodes. They are not well equipped to generate graphs and cannot be compared for this task.

[2]Symbols are used to represent graphlet structures; $g2_1$ is an edge, $g3_1$ is a triangle, $g3_2$ is an open triangle, $g4_{1..6}$ represent clique, chordal cycle, triangle with tail, cycle, star, and path respectively [3].

TABLE III: Graph generation performance. Graphs generated by CNRG closely match the original graph and are consistently the best or close to the best performing model. Lower is better, the best results are indicated by boldface.

	EuCore			**PolBlogs**			**OpenFlights**		
	GCD	λ-dist	DELTACON	GCD	λ-dist	DELTACON	GCD	λ-dist	DELTACON
ChungLu	0.409	**0.803**	6661	0.466	**1.234**	8020	1.1116	**0.614**	14142
HRG	0.229	8.091	7841	1.196	4.407	8872	1.2442	2.761	15860
DC-SBM	**0.180**	2.057	**5736**	0.262	4.186	8023	0.8414	3.534	11450
BTER	–	–	–	0.352	7.505	8444	0.832	4.936	13269
Kronecker	0.3164	11.802	4840	1.302	14.31	6140	1.83	10.459	**8589**
VRG	0.233	4.969	5793	**0.212**	4.276	**7436**	**0.2832**	3.581	11473
	GrQc			**PGP**			**Gnutella**		
	GCD	λ-dist	DELTACON	GCD	λ-dist	DELTACON	GCD	λ-dist	DELTACON
ChungLu	2.657	**0.389**	21607	2	**0.64**	18503	1.02	0.42	34451
HRG	1.99	4.41	**12153**	–	–	-	2	5	**20755**
DC-SBM	2.065	2.202	14456	1.39	2.29	15216	–	–	–
BTER	2.231	0.439	14066	1.61	0.832	15161	1.10	0.474	32692
Kronecker	3.87	5.468	13173	2.882	3.54	12320	3.31	5.96	22145
VRG	**1.067**	0.723	13528	**0.448**	1.329	**12257**	**0.41**	**0.20**	30616

Evaluation. We generate 5 graphs using each model on each dataset and compare each generated graph with the original. To measure how well the local structures are preserved in the generated graph, we counted the number of size-2, 3, and 4 node graphlets [3] and compared those values to the number of graphlets present in the original graph. The (mean average) difference in graphlet counts is indicated as a heatmap in Fig. 9. CNRG consistently outperforms the other models at this task.

GCD, λ-distance, and DELTACON metrics are indicated in Tab. III where bold indicates the best (mean average) performance for each dataset and metric. Across all metrics, the CNRG model performs consistently well, especially in the graphlet counts and the GCD metrics. The ChungLu model does a very good job at capturing the λ-distance; this is expected because ChungLu directly (and only) models the node degree, which is highly correlated with the eigenvalues.

VI. DISCUSSION

The present work describes CNRG, a variant of the vertex replacement grammar model inspired by the context-free grammar formalism widely used in compilers and natural language processing. We described how a CNRG can be extracted from a hierarchical clustering of a graph and then show that the model succinctly encodes the structures present in the original graph. Starting with an empty graph, if we apply CNRG rules stochastically, then the CNRG model can generate a new graph. We show that the newly generated graphs contain global and local topographical features that are similar to the original graph.

A potentially significant benefit from the CNRG model stems from its ability to directly encode local substructures and patterns in the RHSs of the grammar rules. Encoding these local graphlet-like structures is probably the reason that the CNRG model performed so well at the graphlet counting task and the GCD metric. Forward applications of CNRGs may allow scientists to identify previously unknown patterns in graph datasets representing important natural or physical phenomena [35]. Further investigation into the nature of the extracted rules and their meaning (if any) is a top priority.

We also plan to investigate differences between the grammars extracted from different types of graphs. What are the implications of finding two graphs that have a significant overlap in their extracted grammars? What about graphs that seem similar on the surface, but have little overlap in their grammar? Another area of study that we are particularly interested in is learning a temporal grammar from the dynamical processes of an evolving graph. Additional applications of CNRGs are possible on multi-level, multi-layer, and labeled graphs and their various applications.

Acknowledgements. We thank David Chiang for his guidance on this work. This research is supported by a grant from the US National Science Foundation (#1652492).

REFERENCES

[1] Aguiñaga S, Palacios R, Chiang D, Weninger T (2016) Growing graphs from hyperedge replacement graph grammars. In: CIKM, ACM, pp 469–478

[2] Aguiñaga S, Chiang D, Weninger T (2018) Learning hyperedge replacement grammars for graph generation. IEEE Trans on Pattern Analysis and Machine Intelligence pp 1–1, DOI 10.1109/TPAMI.2018.2810877

[3] Ahmed NK, Neville J, Rossi RA, Duffield N (2015) Efficient graphlet counting for large networks. In: ICDM, IEEE, pp 1–10

[4] Baldesi L, Butts CT, Markopoulou A (2018) Spectral graph forge: Graph generation targeting modularity. In: INFOCOM, IEEE, pp 1727–1735

[5] Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment 2008(10):P10008

[6] Bojchevski A, Shchur O, Zügner D, Günnemann S (2018) NetGAN: Generating graphs via random walks. In: Dy J, Krause A (eds) ICML, Stockholm Sweden, vol 80, pp 610–619

[7] Chakrabarti D, Zhan Y, Faloutsos C (2004) R-mat: A recursive model for graph mining. In: SDM, SIAM, pp 442–446

[8] Chung F, Lu L (2002) The average distances in random graphs with given expected degrees. Proceedings of the National Academy of Sciences 99(25):15879–15882

[9] Cook DJ, Holder LB (1993) Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research 1:231–255

[10] Elias P (1975) Universal codeword sets and representations of the integers. IEEE Trans on Information Theory 21(2):194–203

[11] Goyal P, Ferrara E (2018) Graph embedding techniques, applications, and performance: A survey. Knowledge-Based Systems 151:78–94

[12] Grahne G, Zhu J (2005) Fast algorithms for frequent itemset mining using fp-trees. IEEE Trans on Knowledge and Data Engineering 17(10):1347–1362

[13] Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: SIGKDD, ACM, pp 855–864

[14] Grünwald PD (2007) The minimum description length principle. MIT press

[15] Hagen L, Kahng AB (1992) New spectral methods for ratio cut partitioning and clustering. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems 11(9):1074–1085

[16] Hočevar T, Demšar J (2014) A combinatorial approach to graphlet counting. Bioinformatics 30(4):559–565

[17] Jiang C, Coenen F, Zito M (2013) A survey of frequent subgraph mining algorithms. The Knowledge Engineering Review 28(1):75–105

[18] Karrer B, Newman ME (2011) Stochastic blockmodels and community structure in networks. Phys Rev E 83(1):016107

[19] Ketkar NS, Holder LB, Cook DJ (2005) Subdue: Compression-based frequent pattern discovery in graph data. In: Workshop on open source data mining: frequent pattern mining implementations, ACM, pp 71–76

[20] Kipf TN, Welling M (2016) Variational graph auto-encoders. arXiv preprint arXiv:161107308

[21] Kolda TG, Pinar A, Plantenga T, Seshadhri C (2014) A scalable generative graph model with community structure. SIAM Journal on Scientific Computing 36(5):C424–C452

[22] Koutra D, Kang U, Vreeken J, Faloutsos C (2015) Summarizing and understanding large graphs. Statistical Analysis and Data Mining: The ASA Data Science Journal 8(3):183–202

[23] Koutra D, Shah N, Vogelstein JT, Gallagher B, Faloutsos C (2016) Deltacon: principled massive-graph similarity function with attribution. ACM Trans on Knowledge Discovery from Data 10(3):28

[24] Kunegis J (2013) Konect: the koblenz network collection. In: TheWebConf, ACM, pp 1343–1350

[25] Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data

[26] Leskovec J, Chakrabarti D, Kleinberg J, Faloutsos C, Ghahramani Z (2010) Kronecker graphs: An approach to modeling networks. Journal of Machine Learning Research 11(Feb):985–1042

[27] Lim Y, Kang U, Faloutsos C (2014) Slashburn: Graph compression and mining beyond caveman communities. IEEE Trans on Knowledge and Data Engineering 26(12):3077–3089

[28] Lin W, Xiao X, Ghinita G (2014) Large-scale frequent subgraph mining in mapreduce. In: ICDE, IEEE, pp 844–855

[29] Liu Y, Shah N, Koutra D (2015) An empirical comparison of the summarization power of graph clustering methods. arXiv preprint arXiv:151106820

[30] Marcus D, Shavitt Y (2012) Rage–a rapid graphlet enumerator for large networks. Computer Networks 56(2):810–819

[31] Mussmann S, Moore J, Pfeiffer JJ, Neville III J (2014) Assortativity in chung lu random graph models. In: Workshop on Social Network Mining and Analysis, ACM

[32] Mussmann S, Moore J, Pfeiffer III JJ, Neville J (2015) Incorporating assortativity and degree dependence into scalable network models. In: AAAI, pp 238–246

[33] Ng AY, Jordan MI, Weiss Y (2002) On spectral clustering: Analysis and an algorithm. In: NeurIPS, pp 849–856

[34] Nijssen S, Kok JN (2005) The gaston tool for frequent subgraph mining. Electronic Notes in Theoretical Computer Science 127(1):77–87

[35] Pennycuff C, Sikdar S, Vajiac C, Chiang D, Weninger T (2018) Synchronous hyperedge replacement graph grammars. In: ICGT, Springer, pp 20–36

[36] Pfeiffer JJ, La Fond T, Moreno S, Neville J (2012) Fast generation of large scale social networks while incorporating transitive closures. In: Workshop on Privacy, Security, Risk and Trust, IEEE, pp 154–165

[37] Pržulj N (2007) Biological network comparison using graphlet degree distribution. Bioinformatics 23(2):e177–e183

[38] Quinlan JR, Rivest RL (1989) Inferring decision trees using the minimum description length principle. Information and computation 80(3):227–248

[39] Ravasz E, Barabási AL (2003) Hierarchical organization in complex networks. Phys Rev E 67(2):026112

[40] Robins G, Pattison P, Kalish Y, Lusher D (2007) An introduction to exponential random graph (p*) models for social networks. Social networks 29(2):173–191

[41] Rozenberg G (1997) Handbook of Graph Grammars and Comp., vol 1. World scientific

[42] Simonovsky M, Komodakis N (2018) Graphvae: Towards generation of small graphs using variational autoencoders. In: International Conference on Artificial Neural Networks, Springer, pp 412–422

[43] Sun Z, Wang H, Wang H, Shao B, Li J (2012) Efficient subgraph matching on billion node graphs. Proceedings of the VLDB Endowment 5(9):788–799

[44] Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: Large-scale information network embedding. In: TheWebConf, International World Wide Web Conferences Steering Committee, pp 1067–1077

[45] Thoma M, Cheng H, Gretton A, Han J, Kriegel HP, Smola A, Song L, Yu PS, Yan X, Borgwardt KM (2010) Discriminative frequent subgraph mining with optimality guarantees. Statistical Analysis and Data Mining: The ASA Data Science Journal 3(5):302–318

[46] Traag VA, Waltman L, van Eck NJ (2019) From Louvain to Leiden: guaranteeing well-connected communities. Scientific Reports 9(1):5233, DOI 10.1038/s41598-019-41695-z, URL https://doi.org/10.1038/s41598-019-41695-z

[47] Wilson RC, Zhu P (2008) A study of graph spectra for comparing graphs and trees. Pattern Recognition 41(9):2833–2841

[48] Yan X, Han J (2002) gspan: Graph-based substructure pattern mining. In: ICDM, IEEE, pp 721–724

[49] You J, Ying R, Ren X, Hamilton W, Leskovec J (2018) GraphRNN: Generating realistic graphs with deep auto-regressive models. In: Dy J, Krause A (eds) ICML, Stockholm Sweden, vol 80, pp 5708–5717