Edge Replacement Grammars: A Formal Language Approach for Generating Graphs

Revanth Reddy[1]*, Sarath Chandar[2]*, Balaraman Ravindran[1,3]

[1]Department of Computer Science and Engineering, Indian Institute of Technology Madras
[2]Mila, Université de Montréal
[3]Robert Bosch Centre for Data Science and AI, Indian Institute of Technology Madras

Motivation

- A generative model that closely mimics the structural properties of a given set of graphs has utility in a variety of domains.
- The model can be used to anonymize graph data, by generating graphs similar to the original graphs.
- If we are able to fit the model more accurately, we can just save the model instead of the entire graph data.
- We can do graph classification by determining the notion of likelihood of the test graph as per the given model.
- Our graph model is based on a variant of Probabilistic Edge Replacement Grammar (PERG) called Restricted PERG (RPERG).

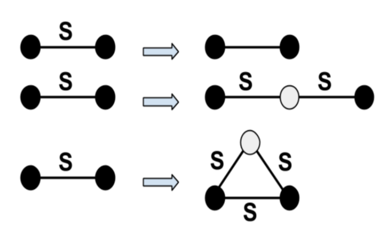
Definitions

Definition 1: An Edge Replacement Grammar (ERG) is a tuple G = $\langle \boldsymbol{N}, \boldsymbol{T}, \boldsymbol{P}, \boldsymbol{S} \rangle$ where

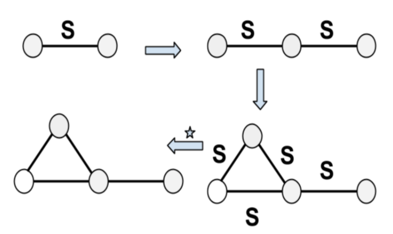
- N and T are finite disjoint set of non-terminal and terminal edge labels.
- S $\in$ N is the start edge label.
- P is a finite set of productions of the form A $\rightarrow$ R, where $\boldsymbol{A} \in$ N and R is a graph fragment with edge labels drawn from N $\cup$ T.

Definition 2: A Probabilistic Edge Replacement Grammar (PERG) consists of

- An edge replacement grammar G = $\langle \boldsymbol{N}, \boldsymbol{T}, \boldsymbol{P}, \boldsymbol{S} \rangle$
- A parameter p(A$\rightarrow$R) for each rule $\boldsymbol{A} \rightarrow \boldsymbol{R} \in \boldsymbol{P}$. This parameter is the conditional probability of choosing this rule given that the non-terminal being expanded is A with the following constraint, for any $\boldsymbol{X} \in \boldsymbol{N}$, $\sum_{A \rightarrow R: A = X} \boldsymbol{p(A \rightarrow R) = 1}$



(a) Sample ERG



(b) Sample derivation

Definition 3: Let u, v be a pair of vertices in the graph G. Let $\boldsymbol{g_1, g_2, ..., g_t}$ be the connected components obtained by removing u,v from G. A squeezing operation with respect to u, v is an operation where one of the components $\boldsymbol{g_i}$ is replaced by an edge between u, v.

When t = 1, the entire graph is squeezed into a single edge. If $\boldsymbol{t \geq 3}$ and $\boldsymbol{g_1, ..g_t}$ are isolated vertices, then squeeze operation replaces entire graph with the edge u, v. If a graph can be squeezed into a single edge, it is a ***trivial squeeze***.

Definition 4: A non-squeezable graph is a graph in which the only squeeze operation that is possible is the trivial squeeze. Star graphs and triangle are considered as the degenerate cases.

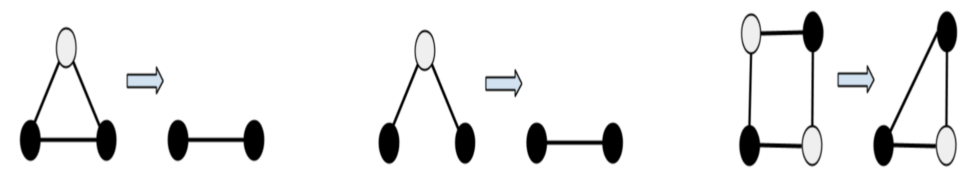


Figure: Examples for Squeezing

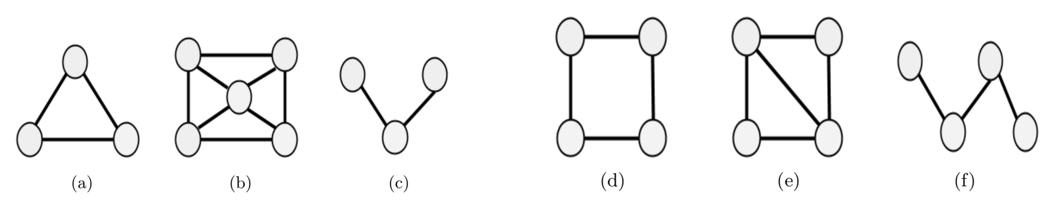


Figure: a,b,c are non-squeezable while d,e,f are squeezable

Definition 5: A Restricted Probabilistic Edge Replacement Grammar (RPERG) is a PERG such that for every rule $\boldsymbol{A} \rightarrow \boldsymbol{R} \in$ RPERG, R is a non-squeezable graph fragment.

Learning the Grammar

If $\boldsymbol{c_D(A \rightarrow R)}$ is the count of the occurrences of the sub-graph R in the data D, then the maximum likelihood estimation of the parameters of the model is given by

$$p_{ML}^{A \rightarrow R} = \frac{c_D(A \rightarrow R)}{\sum_{R': A \rightarrow R'} c_D(A \rightarrow R')}$$

The learning problem now has been reduced to getting the counts of non-squeezable components in a graph.

Algorithm 1 *Learn RPERG*

Input: Set of Graphs D = $\{\boldsymbol{g_1, g_2, ..., g_n}\}$.
Output: RPERG

```
function MAIN(Set of Graphs D)
    Stack ← empty stack
    for each graph g_i do
        GET_COMPONENTS(g_i)
        while Stack is not empty do
            g ← Stack.pop()
            find a split pair (a,b) in g
            if ∃ no split pair then
                C(A → g)+ = 1
            else
                g_1, g_2 ←Obtained by splitting g at (a,b)
                if edge(a,b) ∉ g then
                    Add edge(a,b) to g_2
                end if
                for g' in g_1, g_2 do
                    GET_COMPONENTS(g')
                end for
            end if
        end while
    end for
end function
```

```
function GET_COMPONENTS(Graph g)
    CV ← cut vertices in g
    for each v_k in CV do
        n ← no. of bi-connected components connected by v_k
        C(A → star(n)) += 1
    end for
    S ← set of all bi-connected components in g
    for each s_i in S do
        Stack.push(s_i)
    end for
end function
```

Generative Model

We assume that the network is homogeneous and the links are un-weighted. Since we have only one type of link, number of nonterminal labels is one. The learning algorithm will consider all edges in the given graph to be non-terminal edges.

Algorithm 2 *Generative Model*

Input: RPERG
Output: A Graph

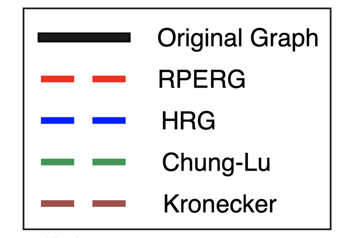
```
Graph G = NULL
Add a non-terminal edge to G
while desired graph size is not reached do
    Randomly pick a non-terminal edge A in G.
    Sample a rule A → R from RPERG and replace A with R in G.
end while
Convert all non-terminal edges in G to terminal edges.
```

Results

We tested the proposed model by fitting it onto several real life graph datasets. The networks vary not only in the number of vertices and edges, but also in the clustering coefficient, diameter, degree distribution and many other graph properties. We compare the properties of the approximate graphs generated from RPERG, HRG, Chung-Lu and Kronecker graph models.

Dataset	Nodes	Edges	Diameter	Clust. Coeff.
Arxiv	5242	14496	17	0.529
Routers	6474	13895	9	0.252
Enron	36692	183831	11	0.497
DBLP	317080	1049866	21	0.632

(a) Dataset Statistics



(b) Color Coding

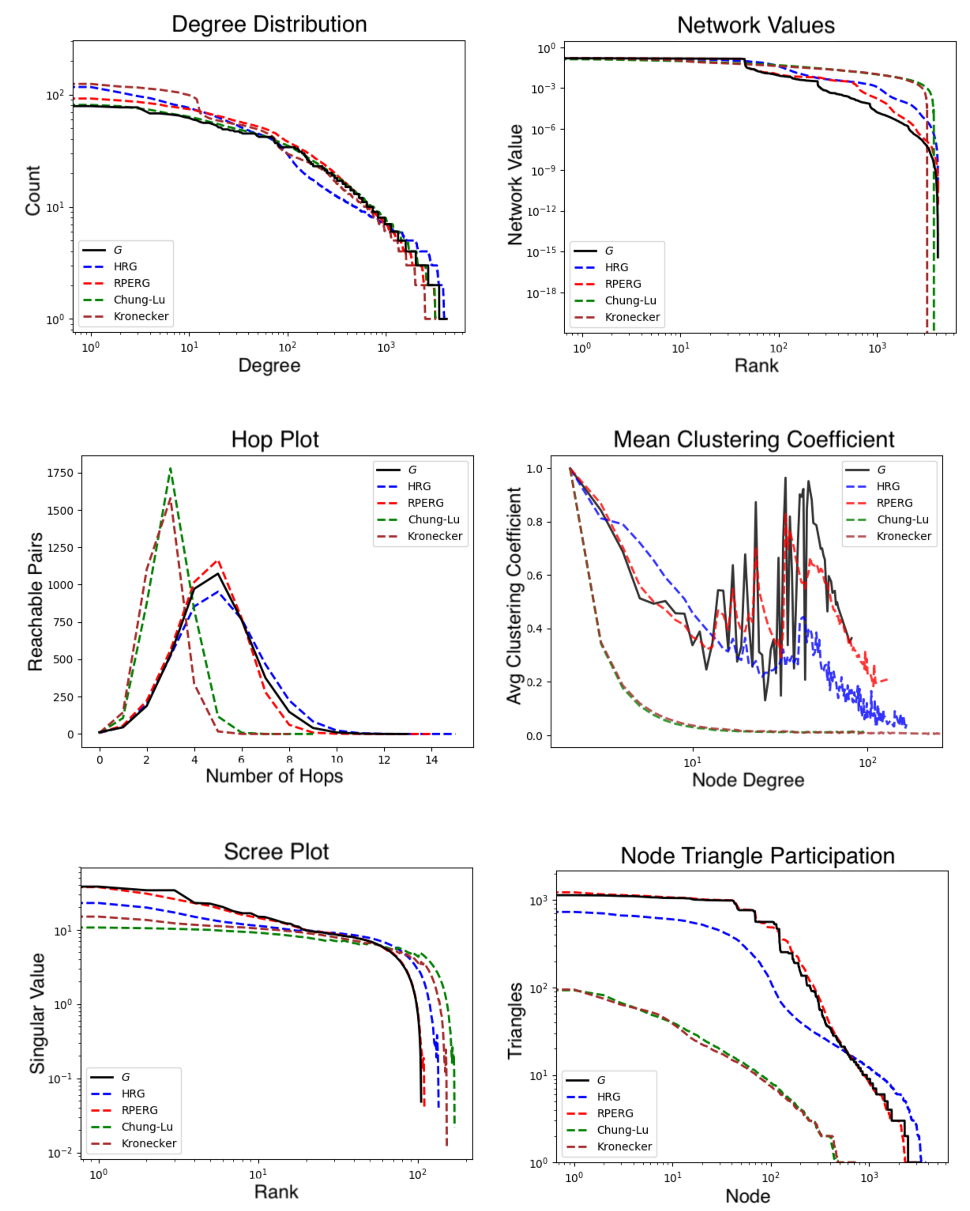


Figure: Arxiv GR-QC

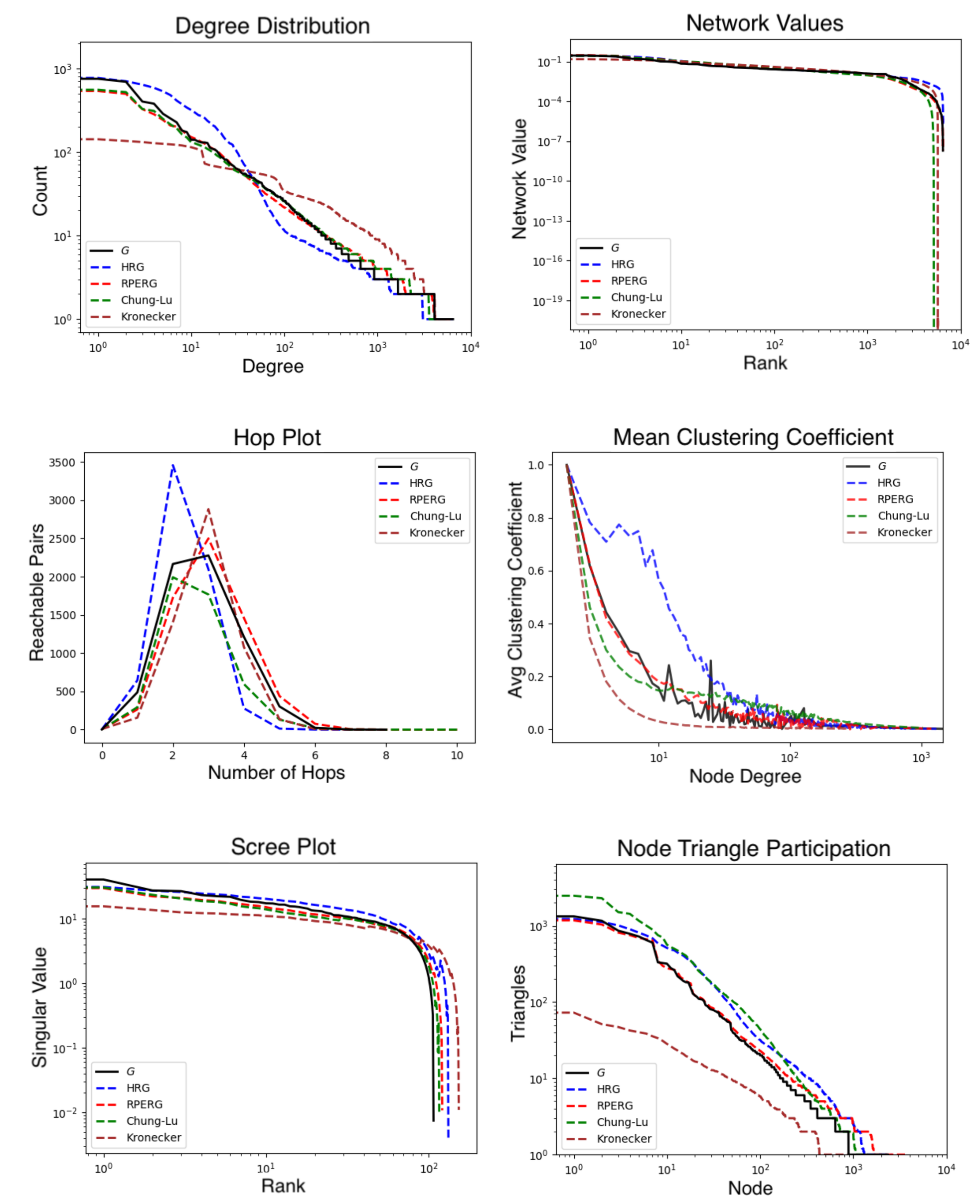


Figure: Internet Routers

To more concretely compare the eigenvectors, the cosine distance between the eigenvector centrality of the original graph and the models' generated graphs is calculated.

Graphlet Correlation Distance (GCD) computes the distance between two graphlet correlation matrices by measuring the frequency of the various graphlets present in each graph, i.e the number of edges, wedges, triangles, squares, 4-cliques, etc., and comparing the graphlet frequencies between two graphs.

Dataset	RPERG	HRG	Chung-Lu	Kronecker
Arxiv	**0.0025**	0.0161	0.3496	0.3406
Routers	**0.0247**	0.0411	0.0379	0.0614
Enron	**0.00007**	0.0002	0.0052	0.0676
DBLP	**0.0079**	0.0649	0.5854	0.4997

(a) Cosine Distance

Dataset	RPERG	HRG	Chung-Lu	Kronecker
Arxiv	**1.086**	1.094	1.792	2.071
Routers	**1.293**	1.404	1.975	2.776
Enron	**0.487**	0.525	1.319	2.83
DBLP	**0.409**	1.602	1.738	2.821

(b) GCD Values

Conclusion

- Even though the grammar is context free, it is able to capture most of the statistical properties of the graph.
- We observe that our algorithm is easily parallelizable as we can run the algorithm simultaneously on multiple graphs.
- In future work, we can try to model preferential attachment by converting the grammar into a context sensitive grammar.
- Tackling graphs with multiple types of links (heterogeneous links) is also a challenging problem.