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, IIT Madras
[2]Mila, Université de Montréal
[3]Robert Bosch Centre for Data Science and AI, IIT Madras

Motivation

Why do we need graph generative models?

- Graph generative models can have utility in a wide range of domains.
- Model can be used to anonymise graph data.
- If we able to fit more accurately, we can just save the model instead of the entire graph.
- Graph classification can be done by determining the notion of likelihood of a test graph as per the given model.

Motivation

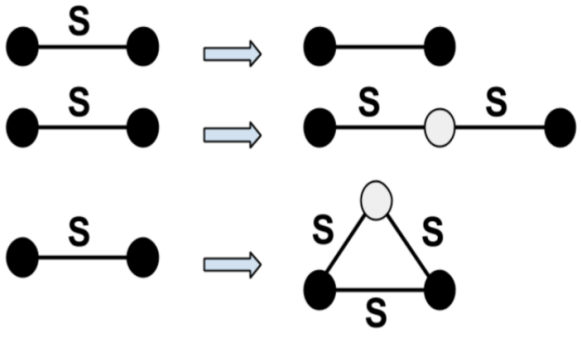
- Graph grammars are an extension of the concept of grammars on strings.
- However, it becomes difficult to model graphs using graph grammars since the RHS of the rules can be any subgraph.
- We want restrict the RHS to certain types of graphs only so that the productions can be learnt.

Our graph model is based on a variant of Probabilistic Edge Replacement Grammar (PERG) called Restricted PERG (RPERG).

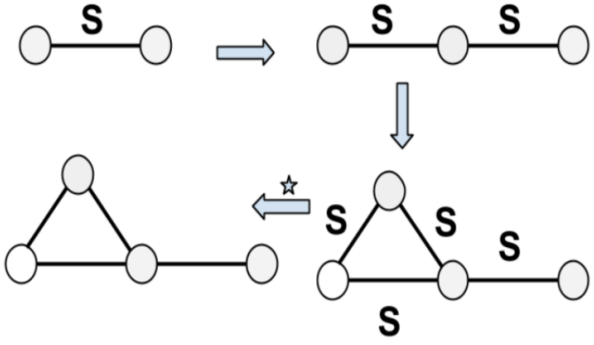
Definitions

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.



(a) Sample ERG



(b) Sample derivation

Definitions

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 $\boldsymbol{X} \in \boldsymbol{N}$, $\sum_{\boldsymbol{A} \rightarrow \boldsymbol{R}: \boldsymbol{A} = \boldsymbol{X}} \boldsymbol{p}(\boldsymbol{A} \rightarrow \boldsymbol{R}) = \boldsymbol{1}$

Squeezing Operation

Definition 3: Let u, v be a pair of vertices in the graph G. Let $g_1, g_2, \ldots, g_t$ 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 g_i is replaced by an edge between u, v.

When t = 1, the entire graph is squeezed into a single edge. If $t \geq 3$ and $g_1, ..g_t$ 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***.

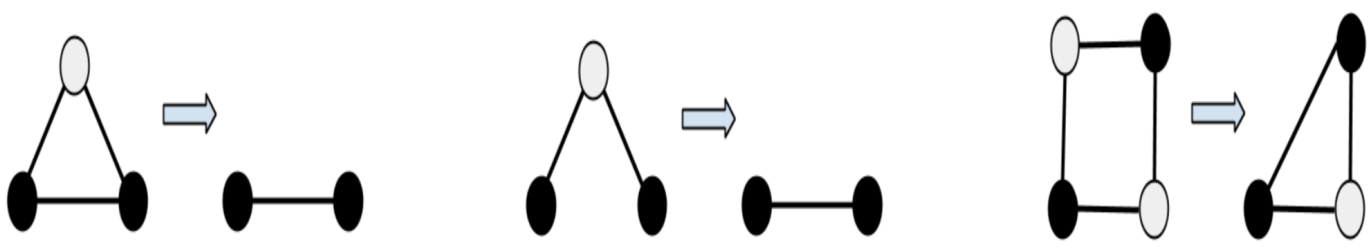


Figure: Examples for Squeezing

Non-Squeezable graphs

Definition 4: A non-squeezable graph is a graph in which the only squeeze operation that is possible is the trivial squeeze.

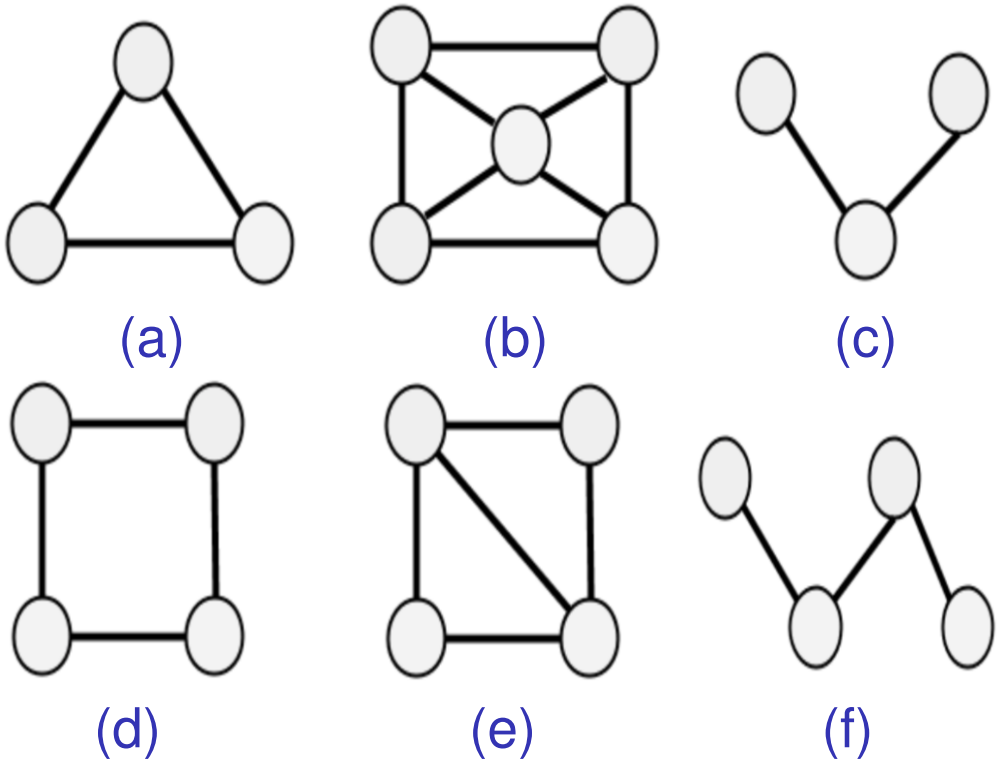


Figure: a,b,c are non-squeezable while d,e,f are squeezable

Learning the grammar

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

We assume that the network is homogeneous and the links are un-weighted. Hence, number of nonterminal labels is one.

The probability of a graph G under an RPERG is given as,

$$p(G) = \prod_{A \rightarrow R \in P} p(A \rightarrow R)^{c(A \rightarrow R)}$$

For a model built on a set of graphs D, the maximum likelihood estimation of the parameters of the model is given by,

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

Algorithm

Algorithm 1 *Learn RPERG*

function MAIN(Set of Graphs D)
  for *each graph* $\boldsymbol{g_i}$ **do**
    GET_COMPONENTS($\boldsymbol{g_i}$)
    while *Stack is not empty* **do**
      $g \leftarrow$ Stack.pop()
      if $\exists$ *no split pair (a,b) in g* **then**
        $\boldsymbol{C(A \rightarrow g)+ = 1}$
      else
        $\boldsymbol{g_1}$, $\boldsymbol{g_2}$ $\leftarrow$Obtained by splitting g at (a,b)
        for g' in $\boldsymbol{g_1}$, $\boldsymbol{g_2}$ **do**
          GET_COMPONENTS(g')
        end for
      end if
    end while
  end for
end function

function GET_COMPONENTS(Graph g)
  for *each* $\boldsymbol{v_k}$ *in cut vertices in g* **do**
    $n \leftarrow$ no. of bi-connected components connected by $\boldsymbol{v_k}$
    $C(A \rightarrow star(n))$ += 1
  end for
  $S \leftarrow$ set of all bi-connected components in g
  for *each* $\boldsymbol{s_i}$ *in S* **do**
    Stack.push($\boldsymbol{s_i}$)
  end for
end function

Graph Generation

In the learnt grammar, all the edges in the RHS of the rules will be non-terminal edges.

Algorithm 2 *Generative Model*

1: Graph G = NULL
2: Add a non-terminal edge to G
3: **while** *desired graph size is not reached* **do**
4: Randomly pick a non-terminal edge A in G.
5: Sample a rule $\boldsymbol{A} \rightarrow \boldsymbol{R}$ from RPERG and replace A with R in G.
6: **end while**
7: Convert all non-terminal edges in G to terminal edges.

Experiments

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

Table: Dataset Statistics for real world graphs.

Experiments

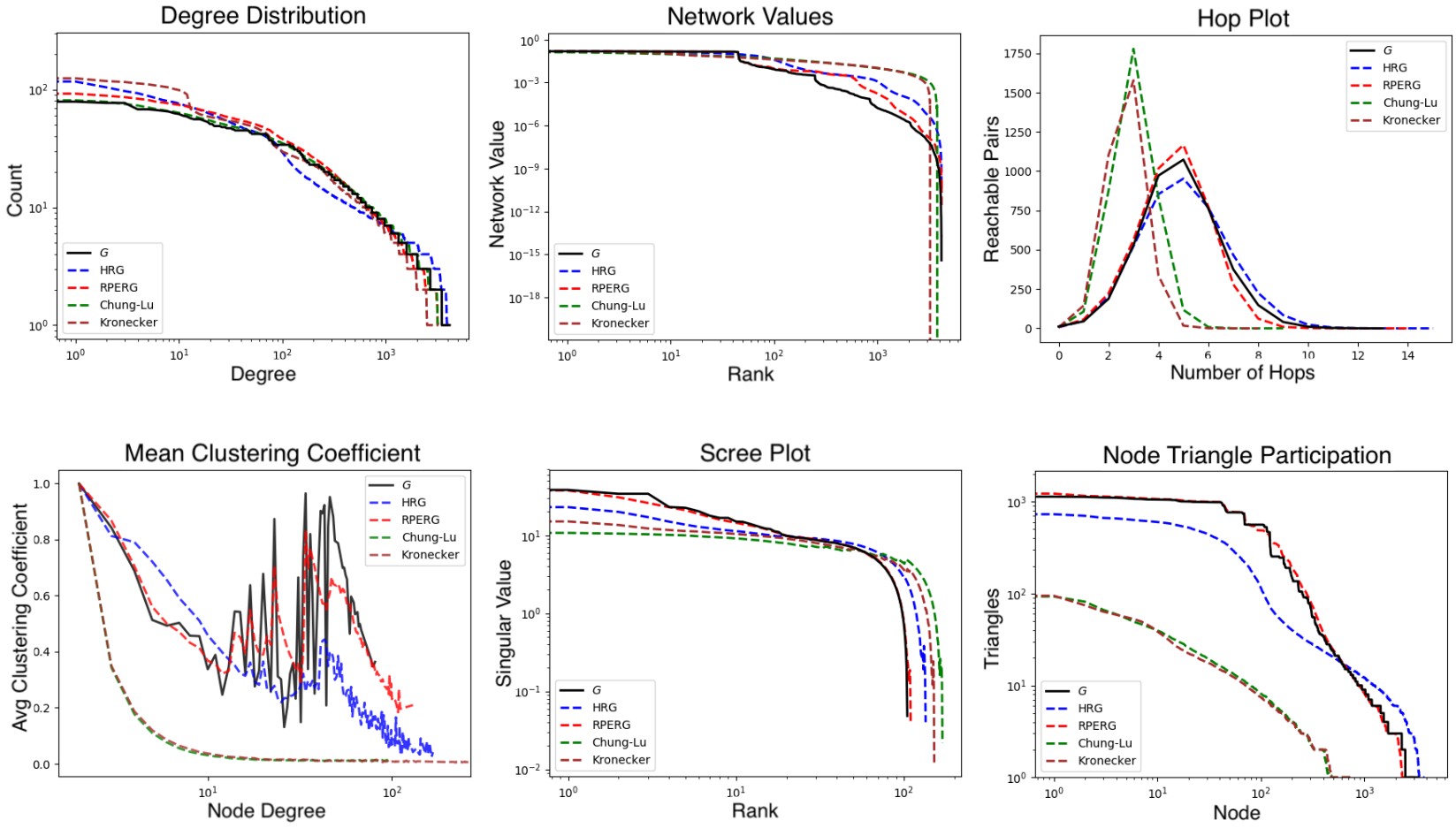


Figure: Plots for Arxiv GR-QC

Experiments

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

Table: Cosine Distance between the eigenvector centrality of original graph and generated graphs.

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

Table: Graphlet Correlation Distance 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.