# Inferring the Structure of Graph Grammars from Data

**Shailesh P. Doshi**
CORAL Lab, CSEE Department,
University of Maryland, Baltimore County,
MD-21250, USA.
Email: *sdoshi1@cs.umbc.edu*

**Fang Huang**
CORAL Lab, CSEE Department,
University of Maryland, Baltimore County,
MD-21250, USA.
Email: *fhuang2@.cs.umbc.edu*

**Dr. Tim Oates**
CORAL Lab, CSEE Department,
University of Maryland, Baltimore County,
MD-21250, USA.
Email: *oates@.cs.umbc.edu*

**Abstract**

Graphs can be used to represent such diverse entities as chemical compounds, transportation networks, and the world wide web. Stochastic graph grammars are compact representations of probability distributions over graphs. We present an algorithm for inferring stochastic graph grammars from data. That is, given a set of graphs that, for example, correspond to a set of chemical compounds, all of which have some desirable property, the algorithm uncovers the structure shared by the graphs and represents it in the form of a stochastic graph grammar. The inferred grammar assigns high probability to the graphs from which it was learned and low probability to other graphs. We report results of preliminary experiments in which inferred graph grammars are compared to target grammars used to generated training data.

## 1. Introduction

Graphs are used to represent chemical compounds [Dehaspe et al., 1998], world wide web [Guillaume and Latapy, 2002], networks, system states, objects, images [Dupplaw and Lewis, 2000], entity relationship diagrams, internal states of machines, the aspects to be learned in concept [Gonzalez et al., 2001, Yoshida and Motoda, 1995] and rule learning, etc. The reason why graphs have such a wide range of application is that many complex situations can be easily represented as graphs. Graph models give a decomposed depiction of complex systems [Gonzalez et al., 2001]. One of the most interesting applications of graphs is representation of relational databases [Getoor et al., 2001] and gene mapping [Craven and Shavlik, 1993]. Graph models also provide a framework for understanding and developing complex learning algorithms. Graph grammars are compact grammatical representations of sets of graphs or probability distributions over graphs. In this paper we make use of a fitness function for inferring graph grammars and learn these grammars from data

Section 2 describes Graph Grammars. Section 3 considers the related work and our contribution to Graph Grammars. Section 4 presents a learning algorithm and its experimental results. Finally, section 5 concludes the paper.

## 2. Graph Grammars

Graph grammars provide a natural generalization of formal language theory based on strings and the theory of term rewriting. The extensive applicability of graph grammars is for the fact that graphs naturally depict complex situations on a perceptive degree. Graph grammars provide a methodology for accurate mathematical models of local transformation on graphs. The key component of graph grammars is a finite set of productions; which generally is a triple (M, D, E) where M and D are the respective mother and daughter graphs, and E is an embedding mechanism. Applying a production corresponds to replacing M with D in the graph G. The embedding rules E advise how to connect D to M. However as opposed to strings, there is just not a single canonical way of defining graph grammars. The two most basic selections for graph rewriting are node replacement and hyperedge replacement. In the case of node replacement graph grammars, a node of a given graph is replaced by a new subgraph which is connected to the remainder of the graph by new edges depending on how the node was connected to it. In this section, we just gave a brief and informal introduction to graph grammars. In the following section, we draw out the analogy between string grammars and graph grammars. Graph Grammars in many ways are analogous to string grammars and thus a lot of known properties of string grammars directly map to graph grammars too.

Context-Free Graph Grammars (CFGGs) are similar to context-free language grammars. Productions have fragments of labeled graphs as their left-hand and right-hand sides. Just as language grammars define sets of strings, graph grammars define sets of graphs. To generate a graph from a language of graph grammar, choose the graph containing a single node labeled with the starting symbol. A node labeled with a non-terminal is selected and replaced by the graph on the right-hand side of the production that has the selected non-terminal node as its left-hand side of the production. This process is repeated until the graph contains only terminal labeled nodes. Graph generation is more complex than string generation because a node is replaced by a new subgraph, which is connected to the remainder of the graph by new edges, depending on how the node was connected to it (neighborhood controlled embedding).

We work with context-free (or confluent) node replacement graph grammars. Node replacements are controlled by productions or replacement rules of the grammar. In context free node replacement graph grammars, the result of the replacement does not depend on the order in which they are applied. Here we basically work with C-edNCE graph grammars. NCE stands for neighborhood controlled embedding, d stands for "directed graphs" and the e means not only nodes but edges of the graphs are also labeled. The C in the beginning is for confluent or context free grammar. An edNCE graph grammar is a tuple $G = (\Sigma, \Delta, \Gamma, \Omega, P, S, E)$ where $\Sigma$ is the set of alphabets of node labels, $\Delta \subseteq \Sigma$ is the set of alphabets of terminal node labels, set $\Gamma$ is of alphabets of edge labels, set $\Omega \subseteq \Gamma$ is of alphabets for final edge labels, P is the finite set of productions, $S \in \Sigma - \Delta$ is the initial nonterminal. A production is of the form $X \rightarrow (D, C)$ with $X \in \Sigma - \Delta$ and $(D, C) \in GRE_{\Sigma,\Gamma}$. E is the set of Embedding rules. C-edNCE grammars, also called VR (Vertex Replacement) grammars seem to have more generating power than HR (Hyperedge replacement) grammars.

Stochastic Context-Free Graph Grammars (SCFGGs) are an extension of SCFGs for representing formal languages. The only difference being that

graph grammars can be used to generate and parse graphs rather than strings. Let L(G) be the language generated by a SCFG. L(G) would consists of all the strings of terminal symbols derivable from the starting symbol of the grammar (typically S). For a string $\alpha \in L$ (LG), the probability of a parse tree of α is the product of all the probabilities of the productions involved in its construction. The probability of the string α is the sum of all the probabilities of its individual parses. SCFGGs are CFGGs with probabilities assigned to each production. They define a probability distribution over graphs and can be used to generate graphs according to that distribution and also to determine the probability of a given graph. In our graph grammar learning, we used node replacement grammars with neighborhood controlled embedding. Below is a simple example of a SCFGG.

S -> A-X [1.0] {}
X -> B-X [0.2] {(A, B) (A, X) (B, B) (B, X)}
X -> C-X [0.3] {(A, C) (A, X) (B, C) (B, X) (C, C)}
X -> D [0.5] {(A, D) (B, D) (C, D)}

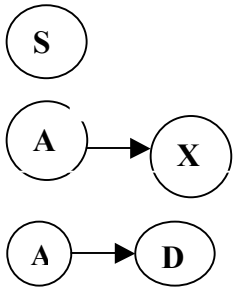


Fig. 1

Figure 1 shows the application of probabilistic selection of rules to generate the graph. In the first pass, S is expanded to give A →X with the application of S→A-X [1.0]. In the next pass, the production with the higher probability is selected to give A→D with the application of X → D [0.5], {(A, D) (B, D) (C, D)}. In the grammar, S is the starting symbol, the real number in [ ] is the probability associated with each production rule, the tuples in {} are the neighborhood controlled embedding rules. In the embedding rule, the first symbol is the neighboring node in the mother graph; the second symbol is the node in the daughter graph, which means there is a new edge between such two nodes in the replaced graph.

## 3. Related Work

Graph Grammars are an active area of research [Rozenberg, 1997]. A wealth of work is being done in the deterministic representation of graph grammars but there exists no known work on learning of graph grammars from data. Cook [Cook and Holder, 2000] discovers frequent substructures in graphs, but they do not take into consideration the embedding rules or production probabilities as in graph grammars. The representation of probabilistic modeling in hyperedge replacement grammars is done by Mosbah [Mosbah, 1992], but not learning of graph grammars was attempted. There's a lot of related work on learning probabilistic string grammars from data, [Keller and Lutz, 1997, Stolcke, 1994] and we draw on similar work.

### 3.1. Our Contribution

There are no methods known that learn graph grammars from data. The primary aim of the presented work is to learn graph grammars on the basis of training data, i.e. graphs. This paper describes the robustness of graph grammars based on the Minimum Description Length (MDL) principle [Cook and Holder, 1994, Derthick, 1991].

The MDL principle: The Minimum Description Length principle was introduced by Rissanen in 1989 and it states that the optimal theory is the one that minimizes the Description Length of a complete data set into consideration. In the algorithm described later, we find the most frequent substructure and then replace it by a nonterminal. The details of the algorithm are described in the later section.

$$MDL = \min \{DL(G'/S) + DL(S) + DL(E/S)\}$$

Where DL(S) is the description length of the substructure selected to be replaced by the nonterminal, DL(G'/S) is the description length of the graph replaced with the selected substructure S, and DL(E/S) is the description length of all the probabilistic embedding rules generated given the substructure S is selected, taking into account the length required to encode values of the probabilities of every individual production.

Stochastic Context-Free Grammars (SCFGs) are a variant of context-free grammars (CFGs) in which each production is associated with a probability, a real number in [0, 1]. The set of production probabilities is referred to as the parameters of the SCFGs. The sum of the probabilities of all the productions expanding the same non-terminal must be one for the SCFG to be proper. SCFGGs define a probability distribution over graphs and can be used to generate graphs according to that distribution and to determine the probability of a given graph which maybe new or unseen. Probabilistic graph grammars provide a tool for describing how production rules are applied.

Here the considered grammars are SCFGGs, the probability of the derivation will be the product of all the probabilities of all the productions used in the derivation. Since the grammar might be ambiguous, the probability of the derived graph is the sum of all probabilities of the derivations generating the graph. And hence you can estimate the probabilities of a context free set of graphs. To explore the relationship between a probabilistic context free set of graphs and its grammar, all the distinct derivations generating any instance or element of the set must be known.

When generating graphs, production is chosen randomly (or in a specified manner) independently of all other productions in the derivation and thus the probability of the derived graph is obtained by multiplying the probabilities of the productions used in all of the derivation. But since graph grammars might be ambiguous, a terminal graph has multiple derivations and thus the probability is the sum of all distinct derivations. This estimation of the probabilities is therefore computationally very expensive. Therefore we estimate the probabilities of the embedding rules of the production grammar, where each embedding for a given production is treated independently. We describe an algorithm that learns probabilistic graph grammars. The algorithm described below is able to handle both directed and undirected graph and chain graphs (these are graphs which have both

directed and undirected edges) to identify a substructure with variable number of edges with labels and directions on them.

## 4. The Learning Algorithm

The approach is based on discovering a frequent structure [Cook and Holder, 1994] in the input graph that minimizes the description length of the graph given that the frequent structure is selected and then estimating probabilities on the embeddings and thus the productions. The algorithm requires an input of a set of graphs in text differentiated with distinct graph ids (graph number in a set of graphs). The graph representation is typical with labeled nodes and labeled edges along with the graph ids. The edges can be directed or undirected with no restrictions in the graphs.

The algorithm performs iterative search on the input, which is a set of graphs. The search for the most frequent substructure is directed with the aim to minimize the description length of the input. Every iteration generates a compressed version of the input graph and a production with probabilities on its embedding rules. At this stage of the algorithm, we look for subgraphs containing just 2 nodes that occur frequently and make that as the right hand side of the productions. This iterative process continues till the description length of the compressed graph with the given probabilistic production is less than the desired fraction of the description length of the previous input graph or all the nodes in the graph are merged to a single node. The algorithm has the ability to be customized to accommodate user specified limits on the number of production rules generated given the fact that the description length is less than the threshold selected at the previous iteration.

For every iteration, the search starts with the estimation of frequencies of all the uniquely labeled vertices (node labels) and then maintaining a table of these frequencies. The NodeCount module does this and selects the top entries of this table. The TwoNode module is a search module which takes a single node and expands it in all the directions giving a two node (vertex) structure along with the edge label between them and the direction of the edge ( the no. of edges can be more than one). This TwoNode module is applied to all the occurrences of the selected node appearing in the set of the input graph file. The TwoNode module generates a table of two_node structures along with their respective frequencies. Here the top structures are selected (according to the specified limit).

The ProductionRule module operates on all the entries of the discovered structures selected. The ProductionRule module gives embedding rules for all occurrences of both of the nodes of the selected discovered structures. These embedding rules consist of all the neighbors of both the nodes of the selected discovered structures for all graphs in the set of input graphs. The embedding rules are represented as two labeled nodes and the number of edges between them with their respective edge labels and edge directions. One of the nodes of the embedding is from the set of two nodes obtained from the selected discovered structure. All these embedding rules are maintained in a table of the production rules along with their respective frequencies.

Each time the ProductionRule module is applied on an occurrence of discovered structure, the selected discovered occurrence is replaced by a nonterminal and this nonterminal is connected with all the neighbors of both the nodes of the selected discovered structure preserving the edges with their

labels and directions. These nonterminal are selected from a set of nonterminals, one distinct nonterminal for each of the selected discovered structure. The output of the ProductionRule module is a table of the distinct nonterminals for the selected discovered structures with the discovered structures themselves and their embedding rules along with their probabilities, all of these represented as graph structures.

For each of the selected Node by the NodeCount module, and for each of the discovered structure for the above, the DL module is applied and the discovered structure for its respective node which gives the minimum DL is selected along with its output graph (set of graphs). This is done by the MDL module. The output of the MDL module at each iteration serves as an input for the next iteration. For each MDL operation, the file of selected production rules, i.e. the grammar file is appended. The DL module takes input as a graph and calculates its description length.

Below is a very simple example:

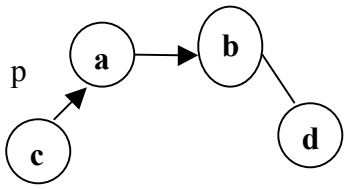


Fig. 2.

The above structure is converted into the following figure. Below is the graph grammar representation of the example above. At each iteration, there is some addition in the set of Productions and Tuples.

$\sum$ = {a, b, c, d, X, Y, Z}
$\Delta$ = {a, b, c, d}
$\Gamma$ = {r, p}
$\Omega$ = {r, p}
P = {}
S = {X}
E = {}

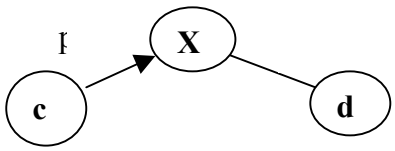


Fig. 3

P = {(a, r/r, b, in)}
E = {(a, p/p, c, out, p1), (b, #/#, d, *, p2)}

Figure 4 is transformed with one nonterminal X and the rest of the nodes are terminal nodes.

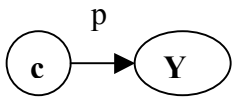


Fig. 4

P = {X: (a, r/r, b, in), Y: (X, #/#, d, *, p2)}
E = {(a, p/p, c, out, p1), (b, #/#, d, *, p2), (X, p/p, c, out, p3)}


Fig. 5

P = {X:(a, r/r, b, in), Y:(X, #/#, d, *), Z:(Y, p/p, c, out,)}
E = {(a, p/p, c, out, p1), (b, #/#, d, *, p2), (X, p/p, c, out, p3)}

There is only one node in the graph and thus the final tuple gives the grammar and the complete set of these tuples defines a language.

### 4.1. Experimental Results

A number of experiments were conducted on the algorithm to test its consistency and reliability. The algorithm was tested on individual graph and a set of variable number of graphs ranging from the size of 1, 5, 10,..., 25. We have used a set of graphs to generate the probabilistic grammar with the MDL fitness function, then again generated the graphs with the extracted grammar and used this extracted grammar to generate a set of graphs. We continue this cycle to generate the graphs from the grammar and then the extracted grammar from these graphs. These cycle runs were done to test the stability of the grammars and the graphs generated from these grammars over multiple phases of compressions and generations. These runs of cycle were done with variable number of graphs at each run.

The grammar to generate graphs was also tested for the probabilities on the production rules. These probabilities were varied for different runs and the graphs obtained were observed for similarity. The results showed that the number of substructures discovered in the graphs were actually proportional to the production probabilities that generated these substructures. The tests conducted on a varying number of similar graphs showed that the quantity of drop in the DL is proportional to the number of similar graphs. The more is the number of similar graphs; more is the drop in the DL. But in the case of graphs with very few similar substructures, there is a very marginal reduction in the DL. This essentially means that the MDL is a dependable fitness function.

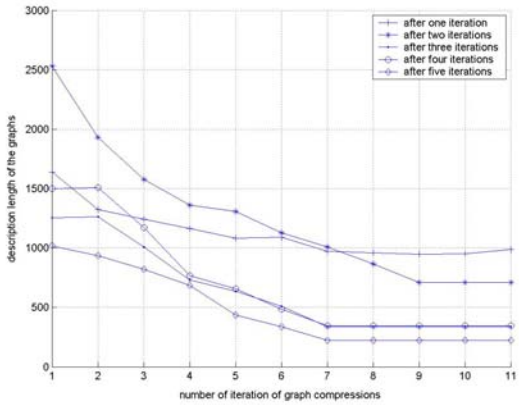


Fig. 6

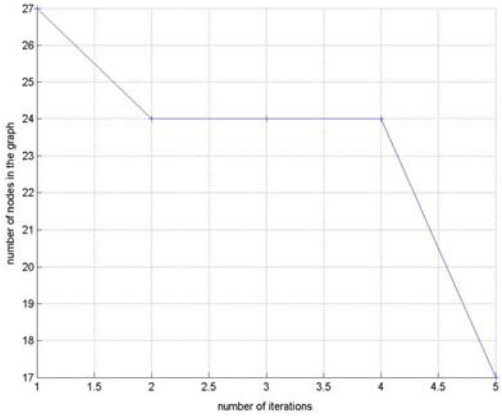


Fig. 7

Figure 6 shows the drop in the description length of the graph after each iteration for variable number of graphs. The results imply that more the number of graphs with some similarity among them, the larger is the drop in the description length of these graphs. Thus the fitness of the MDL principle is quite robust.

There are also some quantitative measures done to compare the generated graphs and the grammars extracted for a few iterations. Given the grammar, the graphs generated were compared for their size using the number of nodes of these graphs as a measure. Figure 7 shows that the number of nodes is fairly consistent. With the number of iterations increasing, there is a slight drop in the number of nodes because the size of the generated graph depends on the number of productions used to generate the graph. The drop for the number of the production rules is mentioned below.

Figure 8 was plotted for the number of production rules generated after each iteration against the number of iterations for graph sets with variable number of graphs. It shows that the number of productions tends to drop over the number of iterations because of the MDL principle. As the description length of the production and its embeddings increase, the size of the grammar generated from the graphs tends to fall so as to have the optimal encoding length for the compressed graphs, and thus the grammar. It is also evident from the figure that the rate of drop of the number of productions is fairly constant. This is due to the MDL principle.

Figure 9 shows the experimental results of a set of graphs with very few similar structures. This set of graphs was probabilistically generated with many productions with a fairly equal probability on each of the productions. Some of the graphs generated from the above grammar had very few similar structures and thus when they were used for compression to generate the grammar, the number of productions and their respective embeddings were also in a larger number.

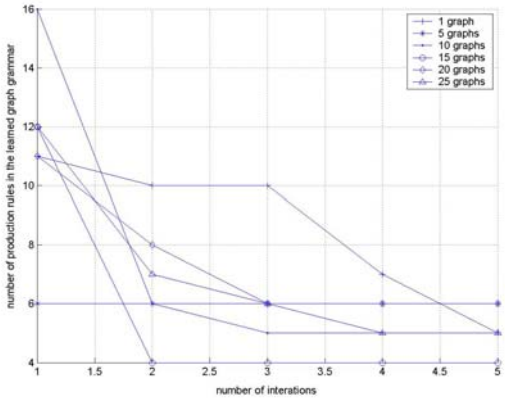


Fig. 8

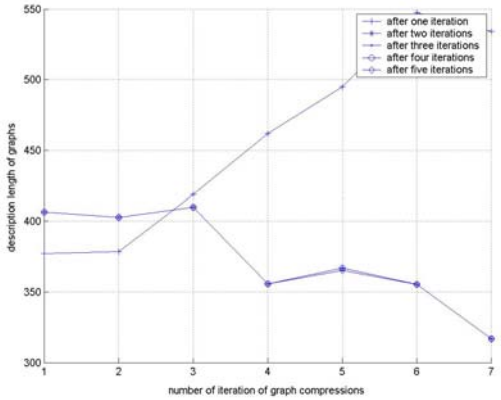


Fig. 9

This accounts for the fact that the description length of these productions and the embeddings contributed to a huge amount towards the description length of the compressed graphs and thus for the grammar generated.

Figure 9 shows the variations in DL of the structurally different graphs after each iteration. The DL after the first iteration actually increased due to the presence of the large number of productions generated. When these productions were further iterated for generation and then compression and repeating the above cycle for a few number of times, the results observed were interesting. These results showed that the drop in the DL for these graphs becomes small and steady after the first iteration. Also after the first iteration, the number of nodes of the generated graphs and the number of extracted productions from these generated graphs had a consistent behavior with respect to the experimental results mentioned above.

## 5. Conclusion

Graphs are a natural representation for a variety of problems, and graph grammars can be used to compactly represent structure shared by a set of graphs. In this paper we described what we believe is the first algorithm for learning stochastic graph grammars from data. The viability of the algorithm was demonstrated in a set of experiments in which a known graph grammar was used to generate training data (i.e. sets of graphs) and our algorithm was used to recover the generating grammar from the data.

In the future we will develop more sophisticated methods for learning the parameters of stochastic graph grammars, such as a version of the Inside-Outside algorithm for graphs that uses Expectation Maximization. Also, we will investigate the use of additional operators in the search for grammar structure, such the merging operator used in Bayesian model merging approaches to the inference of stochastic string grammars.

## 6. References


[Cook and Holder, 2000] D. J. Cook and L. B. Holder. Graph-Based Data Mining. In *IEEE Intelligent Systems*, 15(2): 32-41, 2000.

[Cook and Holder, 1994] D. J. Cook and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. In *Journal of Artificial Intelligence Research*, pages 231-255, 1994.

[Craven and Shavlik, 1993] M. W. Craven and J. W. Shavlik. Machine Learning Approaches to Gene Recognition. In *IEEE Expert*, 9(2): 2-10, 1993.

[Dehaspe *et al.*, 1998] L. Dehaspe, H. Toivonen and R. D. King. Finding Frequent Substructures in Chemical Compounds. In *Proceeding of the Fourth International Conference of Knowledge Discovery and Data Mining*, pages 30-36, 1998.

[Derthick, 1991] M. Derthick. A Minimal Encoding Approach to Feature Discovery. In *Proceedings of the National Conference on Artificial Intelligence*, pages 565-571, 1991.

[Dupplaw and Lewis, 2000] D. Dupplaw and P. H. Lewis. Content- Based Image Retrieval with Scale-spaced object trees. In *M. M. Yeung, B. L. Yeo and C. A. Bouman, editors, Proceedings of Storage and Retrieval for Media Databaes*, volume 3972, pages 253-261, 2000.

[Getoor *et al.*, 2001] L. Getoor, N. Friedman, D. Koller and B. Taskar. Probabilistic Models of Relational Structure. In *International Conference on Machine Learning*, Williamstown, MA, June 2001.

[Gonzalez *et al.*, 2001] J. Gonzalez, L. B. Holder and D. J. Cook. Graph-Based Concept Learning. In *Proceedings of the Florida Artificial Intelligence Research Symposium*, 2001.

[Guillaume and Latapy, 2002] J. Guillaume and M. Latapy. The Web Graph: An Overview. 2002.

[Rozenberg, 1997] Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, 1997.

[Keller and Lutz, 1997] B. Keller and R. Lutz. Evolving Stochastic Context-Free Grammars from Examples Using a Minimum Description Length Principle. In *Workshop on Automata Induction, Grammatical Inference Language Acquisition*, ICML097.

[Mosbah, 1992] M. Mosbah. Probabilistic Graph Grammars. In *Proceedings of the 18th International Workshop, WG '92 on Graph-Theoretic Concepts in Computer Science.*

[Stolcke, 1994] A. Stolcke. Bayesian Learning of Probabilistic Language Models. PhD thesis, University of California, Berkley, 1994.

[Yoshida and Motoda, 1995] K. Yoshida and H. Motoda. Clip: Concept Learning from Inference Pattern. In *Journal of Artificial Intelligence*, 75(1): 63-92, 1995.