

# Inferring Probability Distributions of Graph Size and Node Degree from Stochastic Graph Grammars

Sourav Mukherjee \*

Tim Oates †

## Abstract

Many important domains are naturally described relationally, often using graphs in which nodes correspond to entities and edges to relations. Stochastic graph grammars compactly represent probability distributions over graphs and can be learned from data, such as a set of graphs corresponding to proteins that have the same function. In this paper we show that a stochastic graph grammar can be used to efficiently compute the probability mass functions of the number of nodes, the number of edges, and the degree of a node selected uniformly at random from a graph sampled from the distribution defined by the graph grammar. Both the expectation and variance of these quantities can be computed from the mass functions. Empirical results with standard synthetic grammars and a grammar from the domain of AIDS research demonstrate the accuracy of our methods.

**Keywords.** Graph grammars, stochastic grammars, relational data mining, graph mining.

## 1 Introduction

Many important domains are naturally described relationally, often using graphs in which nodes correspond to entities and edges to relations. For example, a variety of graph-based representations of proteins exist [18], with nodes representing atoms, amino acids, or elements of secondary structure. In Web 2.0 applications such as Facebook and MySpace, nodes represent users and edges correspond to friendship relations. Graph mining algorithms, typically frequent subgraph mining algorithms, are often applied to relational data to discover motifs that characterize sets of related graphs, such as proteins known to have similar function [7] (e.g., inhibiting the AIDS virus) or social networks known to involve specific types of criminal activity.

In this paper we describe methods for efficiently computing statistics about distributions over graphs, such as the expected number of nodes and edges in a graph sampled from a distribution, given a generative model known as a stochastic context-free graph gram-

mar. Just as stochastic string grammars compactly represent probability distributions over strings, stochastic graph grammars compactly represent probability distributions over graphs. The main difference between the two is that production right-hand sides in graph grammars consist of graphs with terminal and non-terminal edges rather than linear sequences of terminal and non-terminal symbols.

Stochastic graph grammars are probability models suitable for data mining in domains where the data is relational and best represented as graphs. They are compact models for large (often infinite) sets of graphs. Given their generative nature, graph grammars are ideally suited for describing relational structures that grow over time, such as social networks and knowledge bases populated from document streams. Moreover, because of their generative nature, graph grammars can be readily used to generate samples. Besides, graph grammars are suitable for modeling important structural properties of the application domain such as hierarchies and recursion. Like Bayes nets and graphical models, stochastic graph grammars allow their structures to be specified by domain experts and their parameters to be subsequently learned using expectation-maximization (EM), although it is also possible to infer their structures from training data, as mentioned in the next paragraph.

Our approach relies on the existence of a graph grammar consisting of productions (structure) and production probabilities (parameters). Given the structure, the probabilities may be learned using the PEGG algorithm [13]. Although the problem of learning the structure of graph grammars from positive examples is still open, algorithms do exist for learning restricted classes of grammars. For example, Kukluk et al. [10] have proposed an algorithm for learning edge replacement graph grammars in which every hyperedge has degree 2. We demonstrate that having access to the generating grammar, either learned from data or hand-coded by an expert, makes it possible to (in some cases) compute precise, as opposed to estimated, values for properties of the distributions over graphs defined by the grammars.

The rest of this paper is organized as follows. Sec-

---

\*Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore 21250, Maryland, USA. Corresponding author. Email: sourav1@umbc.edu

†Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore 21250, Maryland, USA. Email: oates@cs.umbc.edu

tion 2 motivates this paper. Section 3 formally defines the terms and concepts we use throughout this paper. Section 4 presents a brief survey of related work; Section 5 highlights the contributions of this paper. In Section 6, we present techniques for estimating probability distributions of the number of nodes, the number of edges, and the degree of a node selected uniformly at random from a graph, given the underlying graph grammar. Section 7 addresses issues related to the implementation of our algorithms. Section 8 analyzes the time-complexity of our algorithms. Section 9 gives experimental results. Finally, Section 10 concludes the paper and points out future directions.

## 2 Motivation

In this section, we point out a few application scenarios for our work:

1. *Chemical Databases* Given a database of molecular structures represented as graphs, one may learn the structure of the underlying graph grammar using [10], and subsequently, the parameters of that grammar using [13]. However, the problem of utilizing the learned grammar (beyond mere inference of new graphs) has not been addressed satisfactorily, to the best of our knowledge. Using the techniques presented in this paper, we can learn the distributions of the number of nodes (atoms), the number of edges (bonds), as well as the number of bonds per atom (valencies). Our techniques may also be applied in compound design, where we tune the parameters of the underlying grammar until graphs generated by it have a desired distribution.
2. *Social Networks* In the domain of social networks, our techniques can be employed to compute degree distributions, expected number of members and links in a network and so on. Moreover, since a social network is a dynamic structure, our techniques can help determine the effect of structural constraints (expressed as grammar productions and production probabilities) on the distribution of the generated networks.
3. *Anomaly Detection* Anomaly detection might be useful in identifying malicious communities in a social network. If the size and degree distributions of innocuous networks are learned, then this information can be used to identify anomalous networks that may be potentially harmful.

In the next section, we present definitions used in the rest of the paper.

## 3 Preliminaries

In this section, we review the basic definitions and concepts pertaining to stochastic graph grammars. The presentation will follow that given in [13]. Throughout this discussion, we will use the notation  $G = (V, E)$  to refer to a graph with  $V$  being the set of vertices, and  $E$  being the set of edges.

**DEFINITION 3.1.** *Let  $G = (V, E)$  be a graph. A hyperedge is an ordered subset of its vertices  $V$ . Alternatively, a hyperedge of degree  $n$  can be thought of as a mapping  $H : \{1, 2, \dots, n\} \rightarrow V$ .*

*A hypergraph is a graph that can, in addition to edges, also have hyperedges.*

**DEFINITION 3.2.** *A (hyperedge replacement) stochastic context-free graph grammar (SCFGG) is defined as a tuple  $(S, N, T, P, p)$  where:*

- $N$  is the set of non-terminal symbols,
- $T$  is the set of terminal symbols, disjoint from  $N$ ,
- $S \in N$  is a special non-terminal called the start symbol,
- $P$  is a set of productions,
- $p$  is a probability function defined on the set of productions, such that the sum of the probabilities of all productions with the same left hand side equals 1.

*In a hyperedge replacement SCFGG, terminals are used to denote graphs without hyperedges, while non-terminals are used to label hyperedges. Every non-terminal  $Z$  has a positive integer  $D_Z$ , called the degree of  $Z$ , associated with it. A hyperedge  $H$  may be labeled with a non-terminal  $Z$  only if the degree of  $H$  is  $D_Z$ . A production is an ordered pair  $(H, \alpha)$ , written as  $H \rightarrow \alpha$ , where  $H$  is a non-terminal and  $\alpha$  is a hypergraph.*

A SCFGG can be viewed as a generative model: we start with the start symbol  $S$ , and at each step we replace any non-terminal  $H$  with a hypergraph  $\alpha$  such that there is a production  $H \rightarrow \alpha$ . This process is continued until we arrive at a graph that has no non-terminal symbols. When a hyperedge  $H$  in a graph  $G$  is replaced using the production  $H \rightarrow \alpha$ ,  $G$  is called the host graph, and  $\alpha$  is called the subgraph. Finally, the probability of such a derivation is defined as the product of the probabilities of its productions.

The term *embedding rules* is used to describe how the subgraph is placed in the host-graph while replacing a non-terminal. In general, embedding rules for

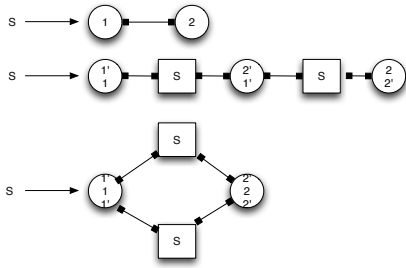


Figure 1: A grammar for generating series parallel paths. Note that non terminal hyperedges are labeled with rectangles, with the non-terminal symbols written inside them. Also, the vertices of the hyperedge are enumerated using primed integers:  $1'$ ,  $2'$ . Finally, the subgraphs have vertices labeled 1, 2 indicating how they will be embedded in the host-graph.

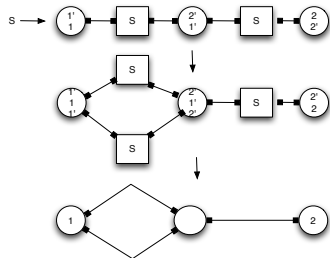


Figure 2: Derivation of a series-parallel graph using the grammar in Figure 1

hyperedge-replacement (HR) graph grammars tend to be much simpler than those for node-replacement graph grammars, in which non-terminals label nodes rather than hyperedges. Let the host-graph  $G$  have a hyperedge  $H$  of degree  $n$ . Then the vertices constituting  $H$  must be labeled as  $1', 2', \dots, n'$ . Also, for every production  $H \rightarrow \alpha$ ,  $\alpha$  must have  $n$  vertices labeled  $1, 2, \dots, n$ . When  $H$  is replaced by  $\alpha$ , node  $j$  in the subgraph must be glued to node  $j'$  in the host-graph. This is the only embedding rule this paper will assume.

As an illustrative example, we consider the grammar shown in Figure 1, which is a grammar for all series-parallel graphs. For this grammar, let the productions have probabilities  $1 - p$ ,  $p_1$  and  $p_2$  respectively, where  $p = p_1 + p_2$ . We use this grammar to illustrate our estimation techniques in later sections.

As an example of a derivation, consider Figure 2. A derivation always starts with a production that has the start symbol  $S$  in the left-hand side. In the series-parallel grammar, there are three such productions, and one of them is selected according to their respective probabilities. In this example, the second production

(the “series” production) is chosen first. The resulting hypergraph now has two non-terminals, both labeled  $S$ . We replace one of them by the right hand side of the third production (the “parallel” production). In the resulting hypergraph, we have three non-terminals, all labeled  $S$ , and we replace all three by the right hand side of the first production (the “terminal” production), resulting in a graph with no hyperedges. The probability of the derivation is the product of the probabilities of the productions used in it. Thus, the probability of this derivation is given by  $P = p_1 \times p_2 \times (1 - p)^3$ . Also, nodes labeled 1 and 2 in the final graph are called *external nodes*, as they would be used for gluing this graph to a host graph, had the start symbol for this derivation been the label of a hyperedge in that host graph. All nodes that are not external nodes are referred to as *internal nodes*. Thus, in any grammar  $G$ , the number of external nodes in any hypergraph derived from a non-terminal  $Z$  is equal to the degree of  $Z$ , i.e.,  $D_Z$ . Note that the degree of a non-terminal is completely different from the degree of a node in a graph, which is simply the number of edges incident on it.

In the next section, we briefly survey work done in graph grammars and related areas.

#### 4 Related Work

Many data mining and machine learning algorithms assume the data to have a “feature vector” representation, where each instance is of the form  $(\mathbf{x}, y)$ , where the input,  $\mathbf{x}$ , is a collection of features, and the output  $y$  is predicted by the algorithms. However, domains such as bioinformatics, chemical databases, social networks, etc. deal with highly structured relational data, better represented as graphs than as feature vectors. Thus, it has become increasingly important to mine information from graph databases. The approaches adopted so far may be put under the following broad categories:

**4.1 Kernel Methods** To perform tasks such as graph classification using support vector machines (SVM), researchers have proposed various kinds of graph kernels in the recent past. For example, Bunke and Shearer [2] have proposed a graph kernel based on the size of the maximal common subgraph (MCS). However, finding the MCS takes exponential time. This has motivated the definition of graph kernels based on simpler substructures such as random walks [6], subtree kernels [3] and so on. However, we are not aware of any work that uses the above methods to extract useful information about the underlying data distribution (except, of course, a decision surface for classification). In this paper, we show that graph grammars can indeed be used to reveal useful information about the distribution.

**4.2 Graph Mining and Graph Grammars** An important task in graph mining is frequent substructure discovery. In [19], Yan et al. have proposed the gSpan algorithm for detecting frequent subgraphs. While frequent subgraph mining discovers frequent substructures, graph grammars take the mining process one step further, by offering generative models for graph languages, where such substructures are frequent. Before considering graph grammars, we briefly review research done in learning string grammars from training data.

Recall that learning a grammar consists of learning both the structure (the productions) and the parameters (the production probabilities) of that grammar. In the context of string grammars, Lari and Young [11] presented an algorithm for learning the production probabilities, given the productions and training examples. The problem of learning the structure (the productions) has also been addressed, both for hidden Markov models (HMMs) [1, 14, 16] that can learn only regular grammars, and for the more general context free grammars [4, 16].

Now we turn to the problem of learning graph grammars from training data. The theory of graph grammars, along with several applications, has been surveyed in [15]. Oates et al. [13] have proposed an algorithm, PEGG (Parameter Estimation in Graph Grammars), for estimating the production probabilities of a stochastic context free graph grammar, given its productions. Algorithms for learning the structure of graph grammars include SubdueGL [9] for deterministic graph grammars, and its extension to stochastic grammars presented in [5]. Both these algorithms learn node-replacement graph grammars. One of the early approaches to learning deterministic hyperedge-replacement (HR) graph grammars was presented by Jeltsch and Kerowski [8]. More recently, Kukluk et al. [10] have proposed an algorithm for learning the structure of edge replacement graph grammars. The class of graph languages expressible by such grammars is restricted [15], since a hyperedge in such a grammar can only have degree 2. However, to the best of our knowledge, no satisfactorily general solution exists to the problem of learning stochastic HR graph grammars.

Moreover, applications of graph grammars have been limited primarily to inference. In this paper, we show that graph grammars can, in fact, be used to unravel useful information about the underlying graph distributions, as summarized in the following section.

## 5 Contributions of this Paper

In [17], Stolcke et al. have demonstrated that a stochastic string grammar may be used to extract useful information about the underlying distribution of strings. In

[12], Mukherjee et al. extended Stolcke's approach to graph grammars, and presented techniques for estimating the expected number of nodes, the estimated number of edges, and the average node degree in a graph distribution.

In this paper, we show that stochastic graph grammars can be used for a more detailed characterization of the underlying distribution. In particular, we present techniques for estimating the probability mass functions of the number of nodes, the number of edges, and the degree of a node selected uniformly at random, from a graph sampled from the distribution. Using these techniques, one may compute not only the expectation of these quantities, but also their variances. In the next section, we present our estimation techniques.

## 6 Estimating Probability Distributions from Graph Grammars

In this section, we show how stochastic graph grammars can be used to compute useful probability distributions pertinent to the distributions of graphs defined by the grammars. Given a graph grammar  $G$ , we define the following probability mass functions (PMFs), for every non-terminal  $Z$  in that grammar:

1.  $P_{G,Z}^{\text{node}}(n)$  denotes the probability that a graph generated by expanding the non-terminal  $Z$  has  $n$  nodes, for  $n \geq 0$ .
2.  $P_{G,Z}^{\text{edge}}(n)$  denotes the probability that a graph generated by expanding the non-terminal  $Z$  has  $n$  edges, for  $n \geq 0$ .
3.  $P_{G,Z}^{\text{degree}}(n)$  denotes the probability that a graph generated by the non-terminal  $Z$  is such that the degree of a node sampled uniformly at random from it equals  $n$ , for  $n \geq 0$ .

Generally, we will drop the subscript  $G$ , so that  $P_{G,Z}^{\text{node}}(n)$  will be abbreviated as  $P_Z^{\text{node}}(n)$ ; similarly for edges and degree.

We present techniques for estimating these probability mass functions. The underlying stochastic graph grammar is assumed to be known. Typically, one would start with a set of samples from the graph distribution, learn the underlying grammar, and then use our techniques to estimate the distributions. For learning the structure (i.e., the productions) of the grammar, one may use the techniques presented in [10]. However, this would restrict the class of grammars learned to those containing hyperedges of degree 2 only. The problem of learning the structure of hyperedge replacement graph grammars in general is still an open problem, and at the time of this writing, [10] seems to be the best available

solution. Having learned the structure, the probabilities may be learned using the PEGG algorithm given in [13].

To aid the presentation, let us first define some notation.

**6.1 Notation** Given a grammar  $G$ , let  $Z$  be a non-terminal in the grammar, such that there are  $N_Z$  production rules with  $Z$  on the left hand side, with probabilities  $p_{Z,1}, p_{Z,2}, \dots, p_{Z,N_Z}$ , satisfying  $\sum_{j=1}^{N_Z} p_{Z,j} = 1$ . Let the  $j^{\text{th}}$  such production be of the form  $Z \rightarrow \alpha_j$  where  $\alpha_j$  is a hypergraph with  $v_{Z,j}$  vertices (of which the number of internal nodes is  $v_{Z,j}^{\text{int}} = v_{Z,j} - D_Z$ ),  $a_{Z,j}$  edges, and  $h_{Z,j}$  hyper-edges, labeled  $Z_{j,1}, Z_{j,2}, \dots, Z_{j,h_{Z,j}}$ . Note that these non-terminals do not have to be all distinct; they may even be the same as  $Z$ . Finally, let  $D_Z$  denote the degree of the non-terminal  $Z$ . We will express our estimates in terms of these symbols.

We now present techniques for computing  $P_{G,Z}^{\text{node}}(n)$ ,  $P_{G,Z}^{\text{edge}}(n)$  and  $P_{G,Z}^{\text{degree}}(n)$ , given the grammar  $G$ .

**6.2 Computing the Distribution of the Number of Nodes** During the process of hyperedge replacement, nodes in the subgraph are glued to nodes in the host-graph. As a result, it is convenient to define the probability mass function of the number of internal nodes:  $P_Z^{\text{int}}(n)$  is the probability that a graph derived from non-terminal  $Z$  has  $n$  internal nodes. It follows that

$$(6.1) \quad P_Z^{\text{node}}(n) = P_Z^{\text{int}}(n - D_Z)$$

Thus computing  $P_Z^{\text{node}}$  is equivalent to computing  $P_Z^{\text{int}}$ . We first show how to compute  $P_Z^{\text{int}}$  for the series-parallel grammar (Figure 1) and then generalize the technique to all grammars.

In Figure 1, the right hand sides of the three productions have, respectively 0, 1 and 0 internal node. Further, the right hand sides of the second and the third productions have two non-terminals each, both labeled  $S$ . Any graph  $g$  derived from  $S$  will have a derivation of the form  $S \Rightarrow \alpha \xrightarrow{*} g$ . Thus, the number of internal nodes in  $g$  will be given by the number of internal nodes in  $\alpha$  plus the number of internal nodes in the subgraphs generated by the non-terminals in  $\alpha$ . This leads to the following equation for the PMF of the number of internal nodes:

$$(6.2) \quad \begin{aligned} P_S^{\text{int}}(n) &= (1 - p)\delta(n, 0) \\ &+ p_1 \sum_{n_1, n_2 \geq 0: n_1 + n_2 + 1 = n} P_S^{\text{int}}(n_1) P_S^{\text{int}}(n_2) \\ &+ p_2 \sum_{n_1, n_2 \geq 0: n_1 + n_2 = n} P_S^{\text{int}}(n_1) P_S^{\text{int}}(n_2) \end{aligned}$$

where  $\delta(n, m) = 1$  if  $n = m$  and 0 otherwise. We now generalize the result to arbitrary grammars.

Using the notation defined in Section 6.1, the PMF of the number of internal nodes in a graph generated by non-terminal  $Z$  is given by:

$$(6.3) \quad P_Z^{\text{int}}(n) = \sum_{j=1}^{N_Z} p_{Z,j} P_{Z,j}^{\text{int}}(n)$$

where  $P_{Z,j}^{\text{int}}$  is the probability that  $\alpha_j$ , the right-hand side  $j^{\text{th}}$  production with  $Z$  on the left hand side, expands into a graph with  $n$  internal nodes.

Further,  $P_{Z,j}^{\text{int}}(n)$  can be computed as follows:

$$(6.4) \quad P_{Z,j}^{\text{int}}(n) = \begin{cases} \delta(n, v_{Z,j}^{\text{int}}), & \text{if } \alpha_j \text{ is a terminal graph,} \\ \sum_{n_1, n_2, \dots, n_{h_{Z,j}}} \prod_{k=1}^{h_{Z,j}} P_{Z_j,k}^{\text{int}}(n_k) & \text{otherwise.} \end{cases}$$

where the sum on the right hand side of Equation (6.4) runs over all non-negative integer solutions  $(n_1, n_2, \dots, n_{h_{Z,j}})$  of the equation:

$$n_1 + n_2 + \dots + n_{h_{Z,j}} + v_{Z,j}^{\text{int}} = n$$

To see how Equation (6.3) reduces to Equation (6.2) for the series-parallel grammar, we note that series parallel grammar has only one non-terminal,  $S$ . The terminal production has probability  $p_{S,1} = (1 - p)$  and has  $v_{S,1}^{\text{int}} = 0$  internal nodes on the right hand side. Hence, according to Equation (6.4), it contributes the term  $(1 - p)\delta(n, 0)$ . The series production has probability  $p_{S,2} = p_1$  and  $v_{S,2}^{\text{int}} = 1$  internal nodes on the right hand side. Hence, according to Equation (6.4), its contribution is  $p_1 \sum_{n_1, n_2 \geq 0: n_1 + n_2 + 1 = n} P_S^{\text{int}}(n_1) P_S^{\text{int}}(n_2)$ . For the parallel production,  $p_{S,3} = p_2$  and  $v_{S,3}^{\text{int}} = 0$  leading to a contribution of  $p_2 \sum_{n_1, n_2 \geq 0: n_1 + n_2 = n} P_S^{\text{int}}(n_1) P_S^{\text{int}}(n_2)$ . By adding these terms on the right hand side of Equation (6.3), we get exactly Equation (6.2).

In the next section, we present analogous techniques for computing the PMF of the number of edges.

**6.3 Computing the Distribution of the Number of Edges** In this section, we show how the probability

mass function of the number of edges can be calculated, given the underlying grammar. In the case of nodes, the PMF of the number of nodes was computed in terms of the PMF of the number of internal nodes. However, for edges, the PMF of the number of edges can be computed directly, since hyperedge replacement does not involve gluing of edges. We first present our technique in the context of the series parallel grammar (Figure 1) and then generalize the results to arbitrary grammars.

In Figure 1, the right hand sides of the three productions have, respectively 1, 0 and 0 edges. Further, the right hand sides of the second and the third productions have two non-terminals each, both labeled  $S$ . Any graph  $g$  derived from  $S$  will have a derivation of the form  $S \Rightarrow \alpha \xrightarrow{*} g$ . Thus, the number of edges in  $g$  will be given by the number of terminal edges in  $\alpha$  plus the number of edges in the subgraphs generated by the non-terminals in  $\alpha$ . This leads to the following equation for the PMF of the number of edges:

$$(6.5) \quad \begin{aligned} P_S^{\text{edge}}(n) &= (1-p)\delta(n,1) \\ &+ p_1 \sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{edge}}(n_1)P_S^{\text{edge}}(n_2) \\ &+ p_2 \sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{edge}}(n_1)P_S^{\text{edge}}(n_2) \end{aligned}$$

We now generalize the result to arbitrary grammars.

Using the notation defined in Section 6.1, the PMF of the number of edges in a graph generated by non-terminal  $Z$  is given by:

$$(6.6) \quad P_Z^{\text{edge}}(n) = \sum_{j=1}^{N_Z} p_{Z,j} P_{Z,j}^{\text{edge}}(n)$$

where  $P_{Z,j}^{\text{edge}}$  is the probability that  $\alpha_j$ , the right-hand side  $j^{\text{th}}$  production with  $Z$  on the left hand side, expands into a graph with  $n$  edges.

Further,  $P_{Z,j}^{\text{edge}}(n)$  can be computed as follows:

$$(6.7) \quad P_{Z,j}^{\text{int}}(n) = \begin{cases} \delta(n, a_{Z,j}), & \text{if } \alpha_j \text{ is a terminal graph,} \\ \sum_{n_1, n_2, \dots, n_{h_{Z,j}}} \prod_{k=1}^{h_{Z,j}} P_{Z,j,k}^{\text{edge}}(n_k) & \text{otherwise.} \end{cases}$$

where the sum on the right hand side of Equation (6.7) runs over all non-negative integer solutions  $(n_1, n_2, \dots, n_{h_{Z,j}})$  of the equation:

$$n_1 + n_2 + \dots + n_{h_{Z,j}} + a_{Z,j} = n$$

To see how Equation (6.6) reduces to Equation (6.5) for the series-parallel grammar, we note that series parallel grammar has only one non-terminal,  $S$ . The terminal production has probability  $p_{S,1} = (1-p)$  and has

$a_{S,1} = 1$  terminal edge on the right hand side. Hence, according to Equation (6.7), it contributes the term  $(1-p)\delta(n,1)$ . The series production has probability  $p_{S,2} = p_1$  and  $a_{S,2} = 0$  edges on the right hand side. Hence, according to Equation (6.7), its contribution is  $p_1 \sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{edge}}(n_1)P_S^{\text{edge}}(n_2)$ . For the parallel production,  $p_{S,3} = p_2$  and  $a_{S,3} = 0$  leading to a contribution of  $p_2 \sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{edge}}(n_1)P_S^{\text{edge}}(n_2)$ . By adding these terms on the right hand side of Equation (6.6), we get exactly Equation (6.5).

In the next section, we turn to the problem of computing the PMF of the degree of a node chosen uniformly at random.

#### 6.4 Computing the Distribution of Node Degree

In this section, we show how the probability mass function of the degree of a randomly selected node can be calculated, given the underlying grammar. We first present our technique in the context of the series parallel grammar (Figure 1) and then generalize the results to arbitrary grammars.

The PMFs computed for the number of nodes and the number of edges were exact; however, the PMF of the node degree is approximate, because we approximate the number of nodes in a graph by the expected number of nodes in that graph. For each non-terminal  $Z$  in the grammar, let  $n_Z$  be the expected number of nodes in a graph derived from  $Z$ . By definition,  $n_Z = \sum_{n=1}^{\infty} n P_Z^{\text{node}}(n)$ ; thus,  $n_Z$  can be computed using the techniques in Section 6.2.

First, consider the degree of a node chosen randomly from the right hand side of the first production in Figure 1. Since there are only two nodes in the graph, both with degree 1, the probability that the chosen node will have degree  $n$  is given by  $\frac{1}{2}(\delta(n,1) + \delta(n,1))$ .

Now consider a graph obtained by expanding the right hand side of the second production in Figure 1. The expected number of nodes in such a graph is  $n_S + n_S - 2 = 2n_S - 2$ . Two of these nodes are external nodes. For each of these external nodes, the probability that the degree is  $n$  is given by  $\sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{degree}}(n_1)P_S^{\text{degree}}(n_2)$ . Each of the non-terminals  $S$  generates a subgraph; the expected number of internal nodes in each subgraph is  $n_S - 2$ , and the probability that a randomly chosen node from this subgraph is degree  $n$  is, by definition,  $P_S^{\text{degree}}(n)$ . Thus, the probability that a randomly selected node in a graph obtained by expanding the right hand side of the second production is given by  $\frac{1}{2n_S-2}[2(n_S-2)P_S^{\text{degree}}(n) + 2 \sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{degree}}(n_1)P_S^{\text{degree}}(n_2)]$ .

By a similar argument, we can show that the probability that a randomly selected node in a graph obtained by expanding the right hand side of the third

production is given by  $\frac{1}{2n_S-1}[2(n_S-2)P_S^{\text{degree}}(n) + 2P_S^{\text{degree}}(n) + \sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{degree}}(n_1)P_S^{\text{degree}}(n_2)]$ .

Combining the above, the probability that a randomly chosen node from a graph derived from  $S$  is  $n$  is given by:

$$\begin{aligned}
 P_S^{\text{degree}}(n) &= (1-p)\frac{1}{2}(\delta(n,1) + \delta(n,1)) \\
 &+ p_1\frac{1}{2n_S-2}[2(n_S-2)P_S^{\text{degree}}(n) \\
 &+ 2\sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{degree}}(n_1)P_S^{\text{degree}}(n_2)] \\
 &+ \frac{1}{2n_S-1}[2(n_S-2)P_S^{\text{degree}}(n) + 2P_S^{\text{degree}}(n) \\
 (6.8) \quad &+ \sum_{n_1, n_2 \geq 0: n_1+n_2=n} P_S^{\text{degree}}(n_1)P_S^{\text{degree}}(n_2)]
 \end{aligned}$$

Now, using the notation defined in Section 6.1, we generalize the above result to arbitrary grammars. Given a grammar  $G$  and any non-terminal  $Z$  in it, the degree distribution of randomly selected nodes in graphs derived from  $Z$  is given by:

$$(6.9) \quad P_Z^{\text{degree}}(n) = \sum_{j=1}^{N_Z} p_j P_{Z,j}^{\text{degree}}(n)$$

where  $P_{Z,j}^{\text{degree}}$  is the degree distribution corresponding to graphs derived from  $\alpha_j$  the right hand side of the  $j^{\text{th}}$  production for  $Z$ .

Recall from Section 6.1 that  $\alpha_j$  has  $h_{Z,j}$  hyperedges labeled  $Z_{j,1}, Z_{j,2}, \dots, Z_{j,h_{Z,j}}$ . Then, it can be easily shown that the expected number of nodes in any graph derived from  $\alpha_j$  is given by:

$$(6.10) \quad n_{Z,j} = v_{Z,j} + \sum_{k=1}^{h_{Z,j}} (n_{Z_j,k} - D_{Z_j,k})$$

Let  $\alpha_j \xrightarrow{*} g$ . Then, a node sampled randomly from  $g$  could either have come from the nodes in  $\alpha_j$  itself, or from the subgraphs obtained by expanding the hyperedges in  $\alpha_j$ . We consider these two types of nodes in turn.

Let the nodes in  $\alpha_j$  be labeled  $\{v_{Z,j,l}\}_{l=1}^{v_{Z,j}}$ . Let  $\pi(v_{Z,j,l}, n)$  denote the probability that  $v_{Z,j,l}$  has degree  $n$ , and let  $a(v_{Z,j,l})$  denote the number of terminal edges incident on  $v_{Z,j,l}$ . Then, there are two possibilities:

1. There are no hyperedges incident on  $v_{Z,j,l}$ . In this case,

$$\pi(v_{Z,j,l}, n) = \delta(n, a(v_{Z,j,l}))$$

2. There are  $h_{Z,j,l} > 0$  hyperedges incident on  $v_{Z,j,l}$ , and they are labeled  $Z_{j,l,1}, Z_{j,l,2}, \dots, Z_{j,l,h_{Z,j,l}}$ . In this case, it is easy to see that

$$\pi(v_{Z,j,l}, n) = \sum_{\substack{n_1, \dots, n_{h_{Z,j,l}} \geq 0 \\ n_1 + \dots + n_{h_{Z,j,l}} + a(v_{Z,j,l}) = n}} \prod_{k=1}^{h_{Z,j,l}} P_{Z_{j,l,k}}^{\text{degree}}(n_k)$$

Now we turn to nodes that arise from subgraphs obtained by expanding the hyperedges in  $\alpha_j$ . Suppose the hyperedge  $Z_{j,k}$  ( $1 \leq k \leq h_{Z,j}$ ) expands into subgraph  $g_k$ ; then the expected number of *internal* nodes in  $g_k$  is given by  $n_{Z_{j,k}} - D_{Z_{j,k}}$ ; the external nodes are glued to nodes in  $\alpha_j$ , and hence have already been accounted for in the preceding paragraph. For each of these nodes, the probability that the degree is  $n$  is, simply,  $P_{Z_{j,k}}^{\text{degree}}(n)$ .

We are finally in a position to give an expression for  $P_{Z,j}^{\text{degree}}$  which occurs in the right hand side of Equation (6.9):

$$\begin{aligned}
 P_{Z,j}^{\text{degree}} &= \frac{1}{n_{Z,j}} \left[ \sum_{k=1}^{h_{Z,j}} (n_{Z_{j,k}} - D_{Z_{j,k}}) P_{Z_{j,k}}^{\text{degree}} \right. \\
 (6.11) \quad &+ \left. \sum_{l=1}^{v_{Z,j}} \pi(v_{Z,j,l}) \right]
 \end{aligned}$$

Equations (6.6) and (6.7) suggest that the computation of  $P_Z^{\text{edge}}$  is recursive; the same is true for  $P_Z^{\text{int}}$  and  $P_Z^{\text{degree}}$ . In the following section, we demonstrate how these quantities may be computed efficiently.

## 7 Implementation Issues

In this section, we address issues related to the implementation of the techniques presented in the previous section. For simplicity, we use the notation  $P_Z(n)$  to denote any one of the three probabilities  $P_Z^{\text{node}}(n)$ ,  $P_Z^{\text{edge}}(n)$  and  $P_Z^{\text{degree}}(n)$ , because the same remarks apply to all three cases. The two issues that one faces in implementing the equations in Section 6 are as follows.

**7.1 Recursive Computation** Consider any grammar  $G$ , and let  $N[G]$  denote the set of non-terminals in  $G$ . It can be seen from Equations (6.3) and (6.4), (6.6) and (6.7), and (6.9) and (6.11) that for any non-terminal  $Z$  and  $n \geq 0$ ,  $P_Z(n)$  is defined in terms of the probabilities of the productions, and in terms of  $\{P_{Z'}(m) : Z' \in N[G], m \leq n\}$ . However,  $P_Z(n)$  will never require computation of  $P_{Z'}(m)$  for any  $Z'$  and  $m > n$ .

For some grammars, it is guaranteed that  $m < n$  for all the recursive calls described above. For

nodes, this happens when the right hand side of every production has an internal node. For edges, this happens when the right hand side of every production has a terminal edge. For degree, this happens when every node in the right hand side of every production has a terminal edge incident on it. In these cases, a simple dynamic programming approach suffices, where we compute  $\{P_z(n) : Z \in N[G]\}$  only after we have computed  $\{P_Z(m) : Z \in N[G], 0 \leq m < n\}$ .

However, there are grammars that do not satisfy the constraints mentioned in the previous paragraph; for such grammars, recursive calls with  $m = n$  can occur. We address this problem by combining dynamic programming with fixed-point iterations. For every  $n \geq 0$  the probabilities  $\{P_z(n) : Z \in N[G]\}$  are computed iteratively, as the limit of a sequence of increasingly accurate estimates  $\{P_z^t(n) : Z \in N[G]\}$ , where  $t = 1, 2, 3, \dots$  denotes the number of iterations. For every  $t$ , the values  $\{P_z^t(n) : Z \in N[G]\}$  are computed from the corresponding equations in Section 6 (e.g., (6.6) and (6.7) for edges) using the values of  $\{P_z^{t-1}(n) : Z \in N[G]\}$  and  $\{P_z(m) : Z \in N[G], m < n\}$ , both of which are guaranteed to have been already computed. Fixed point iterations continue until  $|P_S^t(n) - P_S^{t-1}(n)|$  is sufficiently small, or until the maximum permissible number of fixed point iterations have been performed. In Section 9, we show that such fixed point iterations lead to reasonably accurate estimates. We now turn to the issue of detecting termination of our algorithm.

**7.2 Termination Criterion** In this section, we discuss the problem of deciding when our algorithm terminates, i.e., we wish to determine the largest  $n$  for which  $P_Z(n)$  should be computed. Since  $\lim_{n \rightarrow \infty} P_Z(n) = 0$  for all  $Z \in N[G]$ , a simple strategy is to keep computing until  $P_Z(n) < \epsilon$  for some threshold  $\epsilon > 0$ . However, this strategy does not work in general. For example, consider the  $a^n b^n c^n$  grammar in Figure 3. Any graph generated by this grammar can have only  $n = 3k$  edges, where  $k \in \mathbb{N}$ . Thus,

$$n \not\equiv 0 \pmod{3} \Rightarrow P_S^{\text{edge}}(n) = 0$$

Comparing floating point numbers for equality is not recommended, as these numbers are represented approximately; hence, the strategy of just ignoring these zero probabilities is not feasible, either.

Two strategies that *do* work are as follows:

1. Keep computing  $\{P_z(n) : Z \in N[G]\}$  until  $1 - \sum_{k=0}^n P_S(k) < \epsilon$ , for some threshold  $\epsilon$ . Since  $\sum_{k=0}^{\infty} P_S(k) = 1$  and probabilities are non-negative, this guarantees that for all  $m > n$ ,  $P_S(m) < \epsilon$ .

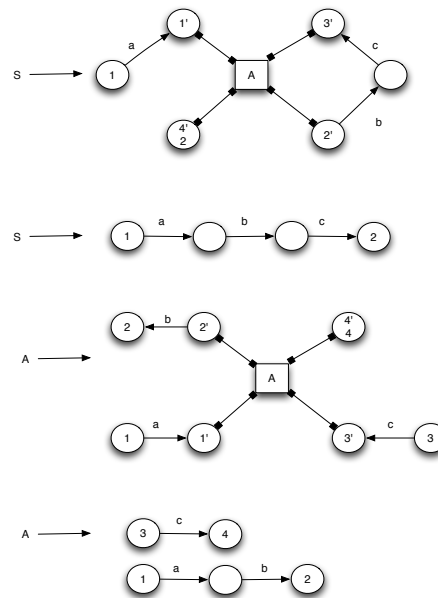


Figure 3: The grammar  $a^n b^n c^n$ . Note that this grammar contains directed labeled terminal edges.

2. Keep computing  $\{P_z(n) : Z \in N[G]\}$  for  $n = 0, 1, \dots, N_{\text{MAX}}$  where  $N_{\text{MAX}}$  is the maximum number of steps permitted before termination. In a typical application scenario, constraints on available resources, response time, etc. would govern the value of  $N_{\text{MAX}}$ .

In our experiments, we have combined the above two solutions. We let  $\epsilon = 0.001$  be the error threshold, and terminate computing after calculating  $P_S(n)$  if either  $1 - \sum_{k=1}^n P_S(k) < \epsilon$ , or  $n = N_{\text{MAX}}$ , where we arbitrarily set  $N_{\text{MAX}} = 500$ . In the next section, we discuss the time complexity of the proposed algorithms.

## 8 Time Complexity

As in the Section 7, we use  $P_Z(n)$  to denote any one of the three probabilities  $P_Z^{\text{node}}(n)$ ,  $P_Z^{\text{edge}}(n)$  or  $P_Z^{\text{degree}}(n)$ , because the same remarks apply to all three cases. For any grammar  $G$  and any non-terminal  $Z \in N[G]$ , the time taken to compute  $P_Z(n)$  depends both on the number of fixed point iterations for that  $n$ , and on the time taken to complete one fixed point iteration. In our implementation, we provide an upper-bound  $F_{\text{MAX}}$  for the number of fixed point iterations for each  $n$ . In one such iteration, the bottleneck lies in computing sums of

the form

$$\sum_{\substack{n_1, n_2, \dots, n_{h_{Z,j}} \geq 0: \\ n_1 + n_2 + \dots + n_{h_{Z,j}} + n_0 = n}} \prod_{k=1}^{n_{h_{Z,j}}} P_{Z,j}(n_k)$$

where  $n_0$  is some constant. Since the  $P_{Z,j}(n_k)$  terms have already been computed, each product term requires  $h_{Z,j}$  multiplications. The number of product terms added in the summation equals  $\binom{n-n_0+h_{Z,j}}{h_{Z,j}}$  which is in  $O(\binom{n+h_{Z,j}}{h_{Z,j}})$ . This leads to the following theorem on the time complexity of computing  $P_Z(n)$ , for a given  $n$ .

**THEOREM 8.1.** *Let  $G$  be a grammar,  $N[G]$  denote the set of non-terminals in  $G$ , and  $h_{\text{MAX}}$  denote the maximum number of non-terminals in the right hand side of any production in  $G$ . Further, let  $F_{\text{MAX}}$  denote the maximum number of fixed point iterations permitted for any  $n \geq 0$ . Then, given the values  $\{P_Z(m) : Z \in N[G], m < n\}$ , the time  $T(n)$  taken to compute  $\{P_Z(n) : Z \in N[G]\}$  is given by:*

$$T(n) = O(F_{\text{MAX}}|N[G]|h_{\text{MAX}} \binom{n+h_{\text{MAX}}}{h_{\text{MAX}}})$$

However, the total running time of our algorithm would be the sum of the time taken to compute  $\{P_Z(n) : Z \in N[G]\}$ , for  $n = 0, 1, \dots, n_{\text{MAX}}$ , where convergence is detected at  $n = n_{\text{MAX}}$  using the criteria mentioned in Section 7.2. This overall time complexity is given by the following theorem.

**THEOREM 8.2.** *Let  $G, N[G], F_{\text{MAX}}$  and  $h_{\text{MAX}}$  be defined as in Theorem 8.1. For any  $n_{\text{MAX}} > 0$ , the time taken to compute  $\{P_Z(n) : Z \in N[G], 0 \leq n \leq n_{\text{MAX}}\}$  is given by:*

$$\sum_{n=0}^{n_{\text{MAX}}} T(n) = O(F_{\text{MAX}}|N[G]|h_{\text{MAX}} \binom{n_{\text{MAX}}+h_{\text{MAX}}+1}{h_{\text{MAX}}+1})$$

*Proof.* Follows immediately from Theorem 8.1 by applying the combinatorial identity:

$$\sum_{n=0}^{n_{\text{MAX}}} \binom{n+h_{\text{MAX}}}{h_{\text{MAX}}} = \binom{n_{\text{MAX}}+h_{\text{MAX}}+1}{h_{\text{MAX}}+1}$$

In the next section, we present experimental results that demonstrate the accuracy of our approach.

## 9 Experimental Results

In this section, we investigate the accuracy of the PMF estimates presented in Section 6, using both standard artificial test grammars, and a grammar learned from real data by [10]. For each grammar, we use the PMF estimates learned for the number of nodes, the number of edges, and the node degree, to compute the corresponding means and variances:

$$\mu = \sum_{n=0}^{n_{\text{MAX}}} n P_S(n)$$

$$\sigma^2 = \sum_{n=0}^{n_{\text{MAX}}} (n - \mu)^2 P_S(n)$$

We then generate a sample of 10,000 graphs from the corresponding grammar, and compare the sample means and sample variances with the estimated means and variances defined above. The experiments are repeated for various settings of the production probabilities, and the results are presented below.

### 9.1 Experiments with Artificial Test Grammars

We use three standard test grammars that are commonly used in the graph grammar literature: the series-parallel grammar in Figure 1 (see [15]), the  $a^n b^n c^n$  grammar in Figure 3 (see [15]), and the triangles-squares grammar in Figure 4. For each grammar, the production probabilities are assigned as follows: for each non-terminal  $Z$ , the terminal production with  $Z$  on the left hand side has probability  $p_{\text{term}}$ . The remaining  $N_z - 1$  productions with  $Z$  on the left hand side have a probability of  $(1 - p_{\text{term}})/(N_z - 1)$  each. We repeat experiments for  $p_{\text{term}} = 0.65 - 0.95$  in steps of 0.05, and for each experiment, we compare the estimated mean and variance against those for a sample of 10,000 graphs, sampled according to the assigned probabilities. Also, whenever  $P_Z^{\text{node}}(n)$ ,  $P_Z^{\text{edge}}(n)$  or  $P_Z^{\text{degree}}(n)$  are computed for any  $n > 0$ , we allow a maximum of  $F_{\text{MAX}} = 1000$  fixed point iterations; of course, fixed point iterations may converge much sooner.

The results are tabulated in Table 1 for the series-parallel grammar, Table 2 for the  $a^n b^n c^n$  grammar, and Table 3 for the triangles-squares grammar. In these tables, columns marked ‘‘Est.’’ refer to estimates made using our techniques, while columns marked ‘‘Obs.’’ refer to observed values from a sample of size 10,000.  $n_{\text{MAX}}$  refers to the highest value of  $n$  at which the PMFs are computed, before terminating due to convergence.

We now describe our experiments with biology data.

### 9.2 Experiments with Biology Data

In [10], Kukluk et al. have reported an edge replacement grammar for the G tetrad structure, which is used in HIV

Table 1: series-parallel grammar.

Terminating Probability	Mean		Variance		$n_{MAX}$
	Est.	Obs.	Est.	Obs.	
# Nodes					
0.6	2.9434	3.0468	6.5869	9.087	500
0.65	2.569	2.6145	2.2031	2.7087	28
0.7	2.3633	2.3884	0.8989	1.0369	11
0.75	2.2438	2.2465	0.4619	0.5127	8
0.8	2.1627	2.1622	0.2527	0.2641	6
0.85	2.102	2.1078	0.1353	0.1496	4
0.9	2.0581	2.0567	0.0704	0.0649	3
0.95	2.0273	2.0266	0.0297	0.0297	3
# Edges					
0.6	2.9222	2.9953	23.9811	27.5569	62
0.65	2.132	2.1295	7.1507	7.3089	32
0.7	1.7275	1.7405	2.8271	3.2298	19
0.75	1.4861	1.4957	1.323	1.4858	13
0.8	1.3264	1.3382	0.6792	0.7522	10
0.85	1.2078	1.2152	0.3362	0.3949	7
0.9	1.123	1.1233	0.1673	0.1811	6
0.95	1.0531	1.0559	0.06	0.0636	4
Degree					
0.6	1.3087	1.554	0.6238	0.8099	10
0.65	1.301	1.4315	0.5905	0.6185	9
0.7	1.2681	1.3243	0.5001	0.4087	9
0.75	1.2147	1.2531	0.351	0.3174	7
0.8	1.1635	1.1786	0.2389	0.2052	6
0.85	1.1188	1.1269	0.1602	0.1312	6
0.9	1.0744	1.0791	0.0899	0.0755	5
0.95	1.0347	1.0371	0.0378	0.0322	4

Table 2:  $a^n b^n c^n$  grammar.

Terminating Probability	Mean		Variance		$n_{MAX}$
	Est.	Obs.	Est.	Obs.	
# Nodes					
0.6	5.9803	6.0202	9.6392	10.2126	24
0.65	5.5983	5.5603	7.1871	6.9606	21
0.7	5.2687	5.3032	5.2901	5.5413	18
0.75	4.9805	4.9885	3.8004	3.9036	15
0.8	4.7437	4.7638	2.7468	2.8528	15
0.85	4.521	4.5199	1.805	1.8186	12
0.9	4.3317	4.3534	1.0985	1.1567	12
0.95	4.1562	4.1566	0.4906	0.5047	9
# Edges					
0.6	4.981	5.0571	9.632	10.306	25
0.65	4.5989	4.6629	7.1805	7.7009	22
0.7	4.2695	4.2897	5.2831	5.6078	19
0.75	3.9814	4.0158	3.7916	4.091	16
0.8	3.744	3.7593	2.7441	2.9434	16
0.85	3.5215	3.5463	1.8009	1.9165	13
0.9	3.3318	3.3303	1.0977	1.069	13
0.95	3.1564	3.1596	0.4897	0.5199	10
Degree					
0.6	1.603	1.6044	0.3775	0.0174	6
0.65	1.5818	1.5888	0.346	0.0152	5
0.7	1.5676	1.5726	0.3267	0.0132	5
0.75	1.5549	1.5592	0.3097	0.0112	5
0.8	1.5432	1.5465	0.2947	0.009	5
0.85	1.529	1.5355	0.2785	0.007	4
0.9	1.52	1.5216	0.2683	0.0044	4
0.95	1.5104	1.5109	0.2589	0.0023	4

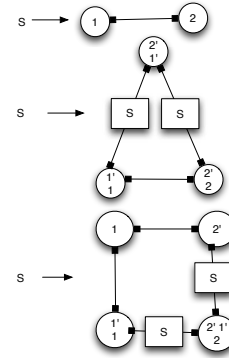


Figure 4: The triangles-squares grammar.

Table 3: triangles-squares grammar.

Terminating Probability	Mean		Variance		$n_{MAX}$
	Est.	Obs.	Est.	Obs.	
# Nodes					
0.6	4.8843	5.1385	54.5441	75.0241	92
0.65	3.6952	3.8012	16.2669	19.0285	46
0.7	3.0903	3.132	6.5241	7.2904	27
0.75	2.728	2.7285	3.0854	3.2254	18
0.8	2.4868	2.4828	1.5823	1.6539	13
0.85	2.3105	2.2942	0.8026	0.8616	9
0.9	2.1823	2.1887	0.3974	0.4231	7
0.95	2.0809	2.0813	0.152	0.1619	5
# Edges					
0.6	5.8071	6.0163	150.3415	194.355	153
0.65	3.8237	3.8697	44.432	48.6951	76
0.7	2.8224	2.9601	17.9168	22.9165	46
0.75	2.2143	2.2508	8.32	9.1933	30
0.8	1.8091	1.8731	4.1777	5.1128	21
0.85	1.5227	1.5679	2.1685	2.759	16
0.9	1.3014	1.3212	1.0154	1.1594	11
0.95	1.1341	1.135	0.3875	0.389	8
Degree					
0.6	1.2576	1.5856	0.3462	0.603	7
0.65	1.2804	1.4956	0.3849	0.502	7
0.7	1.271	1.3969	0.3692	0.4113	7
0.75	1.2384	1.3217	0.3093	0.3253	6
0.8	1.1987	1.2387	0.2495	0.2419	6
0.85	1.1498	1.1726	0.1762	0.1761	5
0.9	1.1013	1.1112	0.1141	0.1136	5
0.95	1.05	1.0562	0.0527	0.0574	4

research. We have chosen this grammar for our experiments. The grammar used in this paper is shown in Figure 5, which is identical to the grammar in [10], except that the R-groups in [10] have been replaced by  $CH_3$  for simplicity. Production probabilities are set in exactly the same way as with the artificial grammars.

Table 4 summarizes the results of experiments conducted with this grammar. The columns have the same meanings as their counterparts for the artificial grammars. We now interpret these results.

**9.3 Remarks** Tables 1, 2, 3 and 4 all show that the expectations computed using our techniques are reasonably close to the sample means; the variances serve as error estimates, and are usually close to the sample vari-

Table 4: g tetrad grammar.

Terminating Probability	Mean		Variance		$n_{MAX}$
	Est.	Obs.	Est.	Obs.	
# Nodes					
0.6	63.431	63.5791	1,465.2951	1,519.3237	297
0.65	58.7185	58.7025	1,092.3857	1,107.1702	260
0.7	54.6553	54.6843	803.7773	837.84	223
0.75	51.1025	51.0546	576.9475	582.447	186
0.8	48.1754	48.3499	417.463	452.9237	186
0.85	45.4314	45.2863	274.0285	274.6679	149
0.9	43.092	43.2365	166.9905	170.0158	149
0.95	40.9284	40.924	74.5045	74.3312	112
# Edges					
0.6	74.7149	74.556	2,167.1959	2,196.7479	361
0.65	68.9836	69.2775	1,615.6174	1,700.6755	316
0.7	64.042	64.4625	1,188.7013	1,257.2086	271
0.75	59.7217	60.2505	853.1208	948.1997	226
0.8	56.16	56.1285	617.4144	621.559	226
0.85	52.8232	53.3205	405.2049	440.0568	181
0.9	49.977	49.9005	246.9845	248.3476	181
0.95	47.3456	47.2455	110.1762	104.5102	136
Degree					
0.6	2.3318	2.336	1.8144	0.0013	6
0.65	2.3289	2.332	1.8128	0.0012	6
0.7	2.3258	2.3284	1.8115	0.001	6
0.75	2.3229	2.3246	1.8099	0.0009	6
0.8	2.3198	2.3207	1.8085	0.0007	6
0.85	2.3167	2.3172	1.8071	0.0005	6
0.9	2.3094	2.3141	1.804	0.0004	5
0.95	2.3085	2.3109	1.8034	0.0002	5

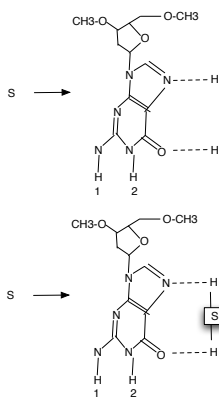


Figure 5: The g tetrad grammar.

ances as well. As the probability of terminal productions,  $p_{term}$  increases, the variances decrease appreciably, because longer derivations become increasingly improbable. The number of steps to convergence,  $n_{MAX}$ , also falls sharply with increase in  $p_{term}$ .

Comparing Table 2 to Table 1, we notice that variances for the node degree in the  $a^n b^n c^n$  grammar are much smaller than those for the node degree in the series-parallel grammar. This is because, in the  $a^n b^n c^n$  grammar, the node degree can have only two values, 1 or 2, whereas in the series-parallel grammar, the node degree is unbounded. Similarly, for the G tetrad grammar (Table 4), the node degree is 1, 2, 3, or 4 depending on whether the node is labeled H, O, N, or

C. Also, as derivations get longer in this grammar, the ratio of these four types of nodes remains approximately constant. This explains why the mean node degree does not change appreciably as  $p_{term}$  increases, and why the corresponding variances are so small.

To summarize, the experimental results show that the estimated means and variances are reasonably accurate, and hence suggest that the estimated PMFs used in computing them are reasonably accurate, too. In the next section, we conclude this paper and point out future directions.

## 10 Conclusion

Given any distribution of graphs, there are several probability distributions that are interesting and useful to researchers, such as the distributions of the number of nodes, the number of edges, and the degree of a randomly selected node. In this paper, we have presented techniques to efficiently compute the probability mass functions (PMFs) of these distributions using the underlying graph grammar. The computed PMFs can be used, for example, to calculate the corresponding means and variances. Our experimental results show that these estimates are accurate. Stochastic graph grammars have been used in the past mainly for inference, i.e., to estimate the probability that a given graph will be generated by a grammar. However, our work shows that stochastic graph grammars can indeed provide a much deeper understanding of the underlying graph distributions.

Future directions include characterizing the convergence rate of our algorithm in terms of the production probabilities and the permitted error, and applying our techniques to other real life scenarios, such as social networks, to extract useful information (such as degree distributions) from such domains.

## References

- [1] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, NJ, USA, 1990.
- [2] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, Mar. 1998.
- [3] M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL*, pages 263–270, 2002.
- [4] C. M. Cook, A. Rosenfeld, and A. R. Aronson. Grammatical inference by hill climbing. *infosci*, 10:59–80, 1976.
- [5] S. Doshi, F. Huang, and T. Oates. Inferring the structure of graph grammar from data. In *Proceedings of the International Conference on Knowledge Based Computer Systems (KBCS)*, 2002.
- [6] T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schölkopf and M. K. Warmuth, editors, *COLT*, volume 2777 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2003.
- [7] J. Huan, D. Bandyopadhyay, W. Wang, J. Snoeyink, J. Prins, and A. Tropsha. Comparing graph representations of protein structure for mining family-specific residue-based packing motifs. *Journal of Computational Biology*, 12(6):657–671, 2005.
- [8] E. Jeltsch and H.-J. Kreowski. Grammatical inference based on hyperedge replacement. In S. Lucas, editor, *Grammatical Inference: Theory, Applications and Alternatives; 1st International Colloquium*, pages 7/1–7/6. The Institution of Electrical Engineers, 1993.
- [9] I. Jonyer, L. B. Holder, and D. J. Cook. MDL-based context-free graph grammar induction.
- [10] J. P. Kukluk, L. B. Holder, and D. J. Cook. Inference of edge replacement graph grammars. *International Journal on Artificial Intelligence Tools*, (3):539–554, 2008.
- [11] K. Lari and S. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–36, 1990.
- [12] S. Mukherjee and T. Oates. Estimating graph parameters using graph grammars. In A. Clark, F. Coste, and L. Miclet, editors, *IGCI*, volume 5278 of *Lecture Notes in Computer Science*, pages 292–294. Springer, 2008.
- [13] T. Oates, S. Doshi, and F. Huang. Estimating maximum likelihood parameters for stochastic context-free graph grammars. In T. Horváth and A. Yamamoto, editors, *Proceedings of the 13th International Conference on Inductive Logic Programming*, volume 2835 of *Lecture Notes in Artificial Intelligence*, pages 281–298. Springer-Verlag, 2003.
- [14] D. Ron, Y. Singer, and N. Tishby. The power of amnesia. In J. D. Cowan, G. Tesauro, and J. Alspecter, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 176–183. Morgan Kaufmann Publishers, Inc., 1994.
- [15] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.
- [16] A. Stolcke and S. Omohundro. Inducing probabilistic grammars by bayesian model merging. In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second International ICGI Colloquium on Grammatical Inference and Applications*, volume 862 of *LNAI*, pages 106–118, Berlin, Sept. 1994. Springer Verlag.
- [17] A. Stolcke and J. Segal. Precise N-gram probabilities from stochastic context-free grammars. In *ACL*, pages 74–79, 1994.
- [18] S. Vishveshwara, K. Brinda, and N. Kannan. Protein structure: Insights from graph theory. In *Journal of Computational Chemistry*, volume 1, pages 187 – 211, 2002.
- [19] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724. IEEE Computer Society, 2002.