Estimating Maximum Likelihood Parameters for Stochastic Context-Free Graph Grammars

Tim Oates, Shailesh Doshi, and Fang Huang

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County, Baltimore, MD 21250
{oates,sdoshi1,fhuang2}@cs.umbc.edu

Abstract. Given a sample from an unknown probability distribution over strings, there exist algorithms for inferring the structure and parameters of stochastic grammatical representations of the unknown distribution, i.e. string grammars. Despite the fact that research on grammatical representations of sets of graphs has been conducted since the late 1960's, almost no work has considered the possibility of stochastic graph grammars and no algorithms exist for inferring stochastic graph grammars from data. This paper presents PEGG, an algorithm for estimating the parameters of stochastic context-free graph grammars given a sample from an unknown probability distribution over graphs. It is established formally that for a certain class of graph grammars PEGG finds parameter estimates in polynomial time that maximize the likelihood of the data, and preliminary empirical results demonstrate that the algorithm performs well in practice.

1 Introduction

Graphs are a natural representation for relational data. Nodes correspond to entities, edges correspond to relations, and symbolic or numeric labels on nodes and edges provide additional information about particular entities and relations. Graphs are routinely used to represent everything from social networks (Pattison, 1993) to chemical compounds (Cook et al., 2001) to visual scenes (Hong & Huang, 2002).

Suppose you have a set of graphs, each representing an observed instance of a known money laundering scheme (Office of Technology Assessment, 1995). It would be useful to learn a statistical model of these graphs that supports the following operations:

- *Compute graph probabilities:* If the model represents a probability distribution over graphs, then it is possible to determine the probability of a new graph given the model. In the context of money laundering schemes, this would amount to determining whether a newly observed set of business relationships and transactions (represented as a graph) is likely to be an instance of money laundering.

- *Identify recurring structures:* Money laundering schemes may contain common components (i.e. sub-graphs) that are arranged in a variety of ways. To better understand the domain, it is useful to explicitly identify such components and the common ways in which they are connected to one another.
- *Sample new graphs:* Given the model, one might want to sample new graphs (money laundering schemes) according to the probability distribution defined by the model. This might be useful in exploring the space of possible schemes, perhaps looking for new variants that law enforcement has not previously considered, or for generating training examples from which humans or programs can learn.

Stochastic grammatical representations of probability distributions over strings, such as stochastic context-free grammars (SCFGs), support these three operations. Given a SCFG, G, and a string, s, it is possible to efficiently compute $p(s|G)$. It is also straightforward to sample strings from the probability distribution defined by G. Finally, there exist a number of methods for learning both the structure (Stolcke, 1994) and parameters (Lari & Young, 1990) of string grammars from data. The most well-known algorithm for computing maximum likelihood estimates of the parameters of string grammars is the Inside-Outside algorithm (Lari & Young, 1990). In addition to estimating parameters, this algorithm can be used to learn structure. This is done by constructing a grammar containing, for example, all possible CNF productions that can be created from a given set of terminals and non-terminals. Inside-Outside can then prune away (i.e. set production probabilities to zero) those productions that are possible but that are not actually in the grammar that generated the training data. Also, Inside-Outside can be used as a component in a system that explicitly searches over the space of grammar structures, iteratively evaluating structures/parameters via, for example, the description length of the grammar and the data given the grammar.

We have embarked on a program of research aimed at creating algorithms for learning and reasoning with stochastic grammatical representations of probability distributions over graphs that provide functionality mirroring that available for string grammars. There exists a fairly extensive literature on deterministic graph grammars that define sets of graphs in the language of the grammar (see, for example, (Engelfriet & Rozenberg, 1997) and (Ehrig et al., 1999)), just as deterministic string grammars define sets of strings that are in the language of the grammar. However, the vast majority of existing work on graph grammars has completely ignored the possibility of stochastic graph grammars and there is no work whatsoever on learning either the structure or parameters of stochastic graph grammars.

This paper describes an algorithm for estimating the parameters of stochastic graph grammars that we call Parameter Estimation for Graph Grammars (PEGG), the first algorithm of its kind. PEGG is similar in many respects to the Inside-Outside algorithm. PEGG computes inside and outside probabilities in polynomial time, and can use these probabilities to efficiently compute $p(g|G)$, the probability of graph g given graph grammar G. In addition, PEGG com-

putes maximum likelihood estimates of grammar parameters for a given grammar structure and set of graphs, again in polynomial time. Though we have explored the use of Bayesian model merging techniques developed for learning the structure (i.e. productions) of string grammars (Stolcke, 1994) in the context of learning the structure of graph grammars (Doshi et al., 2002), the current focus is on parameter estimation.

The ability to learn grammar-based representations of probability distributions over graphs has the attractive property that non-terminals encode information about classes of functionally equivalent sub-graphs. For example, most money laundering schemes have a method for introducing illegal funds into the financial system and a method for moving the funds around to distance them from the source. If sub-graphs in the ground instances of money laundering schemes correspond to these methods, and there are different instantiations of each method, it is reasonable to expect that the learned grammar will contain a non-terminal that expands to ways of introducing funds into the financial system and another non-terminal that expands to ways of moving these funds around. Identifying these non-terminals in the learned grammar makes it possible to enumerate the sub-graphs they generate (i.e. all possible instantiations of a method) and to determine their probability of occurrence to, for example, focus law enforcement efforts.

From a more formal standpoint, graphs are logical structures, so individual graphs and sets of graphs can be described by logical formulas. It is possible to deduce properties of graphs and sets of graphs from these descriptions (Immerman, 1999). PEGG opens up the possibility of automatically synthesizing logical descriptions (i.e. graph grammars) of sets of graphs from data. For example, the expressive power of certain graph grammar formalisms is co-extensive with that of monadic second-order logic (Courcelle, 1997).

The remainder of this paper is organized as follows. Section 2 describes stochastic context-free graph grammars and discusses their relationship to stochastic context-free string grammars. Despite the fact that graph grammars have a rich history of application in a variety of domains, no algorithms exist for learning them from data. To introduce the fundamental concepts of grammar induction from data, section 3 reviews the Inside-Outside algorithm for estimating the parameters of stochastic context-free string grammars. Section 4 introduces the Parameter Estimation for Graph Grammars (PEGG) algorithm for learning maximum likelihood parameter estimates for graph grammars. Section 5 presents the results of a set of preliminary experiments with PEGG. Section 6 reviews related work, concludes, and discusses a number of directions in which we are taking this research.

2 Graph Grammars

This section provides an overview of graph grammars. For a thorough introduction to the formal foundations of graph grammars, see (Engelfriet & Rozenberg,

1997), and to learn more about the vast array of domains in which graph grammars have been applied, see (Ehrig et al., 1999).

The easiest way to build intuition about graph grammars is by way of comparison with string grammars, for which we will take stochastic context-free grammars to be the paradigmatic example. (For the remainder of this paper the term *string grammar* means stochastic context-free string grammar.) A string grammar G is a 4-tuple (S, N, Σ, P) where N is a set of non-terminal symbols, $S \in N$ is the start symbol, Σ is a set of terminal symbols disjoint from N, and P is a set of productions. Associated with each production is a probability such that the probabilities for productions with the same left-hand side sum to one. Sometimes it will be convenient to describe grammars as being composed of *structure* and *parameters*, where the parameters are the production probabilities and the structure is everything else.

In this paper we will be concerned exclusively with stochastic context-free graph grammars (Mosbah, 1994), and will use the term *graph grammar* to refer to grammars of this type. Despite the fact that our present concern is with stochastic graph grammars, it is important to note that prior work reported in the literature has focused almost exclusively on deterministic grammars.

Just as string grammars define probability distributions over strings, graph grammars define probability distributions over graphs. A graph grammar G is a 4-tuple (S, N, Σ, P) where N is a set of non-terminal symbols, $S \in N$ is the start symbol, Σ is a set of terminal symbols disjoint from N, and P is a set of productions. Associated with each production is a probability such that the probabilities for productions with the same left-hand side sum to one.

The primary difference between string grammars and graph grammars lies in the right-hand sides of productions. String grammar productions have strings of terminals and non-terminals on their right-hand sides. Graph grammar productions have graphs on their right-hand sides. At this point the reader may well be wondering where the terminals and non-terminals appear in the graphs generated by graph grammars. It turns out that they can be associated with nodes, yielding a class of grammars known as Node Controlled Embedding (NCE) graph grammars, or they can be associated with edges, yielding a class of grammars known as Hyperedge Replacement (HR) graph grammars. For reasons that will be discussed later, we focus exclusively on HR grammars.

Figure 1 shows the three productions in a simple HR grammar (Drewes et al., 1997) that has one non-terminal - S. Each left-hand side is a single non-terminal and each right-hand side is a graph. Some of the edges in the graphs are labeled with non-terminals in boxes. These *non-terminal edges* can be expanded, a process that involves removing the edge and replacing it with the graph on the right-hand side of a matching production. Each right-hand side has a pair of nodes labeled 1 and 2 that are used to orient the graph when it replaces a non-terminal edge. We will generally use the term *host graph* to refer to the graph containing the non-terminal edge and the term *sub-graph* to refer to the graph that replaces the non-terminal edge.

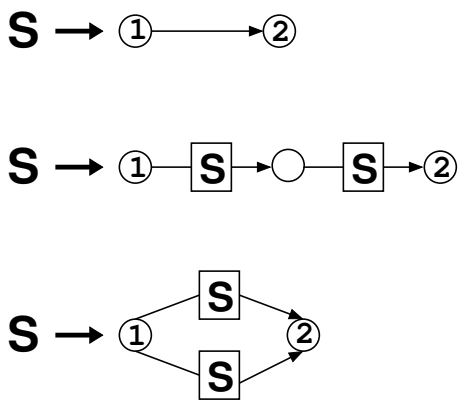

Fig. 1. Productions in a simple HR grammar.

Figure 2 shows a partial derivation using the productions in figure 1. The second graph in figure 2 is obtained from the first by removing the labeled edge and replacing it with the sub-graph on the right-hand side of the second production in figure 1. After removing the edge, all that remains is two disconnected nodes, one that used to be at the head of the edge and the other at the tail. The edge is replaced by *gluing* the node labeled 1 in the sub-graph to the node that was at the head of the removed edge. Likewise, the node labeled 2 in the sub-graph is glued to (i.e. made the same node as) the node that was at the tail of the removed edge.

The last graph in figure 2 is obtained from the penultimate graph by replacing a non-terminal edge with the right-hand side of the first production in figure 1. This results in an edge with no label – a terminal edge – which can therefore not be expanded. A terminal graph is one that contains only terminal edges. Terminal edges can be unlabeled, as in the current example, or productions can specify labels for them from the set of terminals Σ.

Note that every production in figure 1 has exactly two distinguished nodes, labeled 1 and 2, that are used to orient the sub-graph in the host graph when an edge is replaced. When expanding a non-terminal in the derivation of a string there is no ambiguity about how to join the substrings to the left and right of the non-terminal with its expansion. Things are not so clear when expanding non-terminal edges to graphs. Given that the sub-graph to which the non-terminal is expanded will be attached by gluing, there are in general several possible attachments. Consider the second production in figure 1, whose right-hand side has three nodes. When it is used to replace a non-terminal edge, there are 6 possible ways of gluing the sub-graph to the host graph. Any of the three sub-graph nodes can be glued to the host graph node that was at the head of the non-terminal edge, and any of the remaining two sub-graph nodes can be glued to the host-graph node that was at the tail of the non-terminal edge. To remove

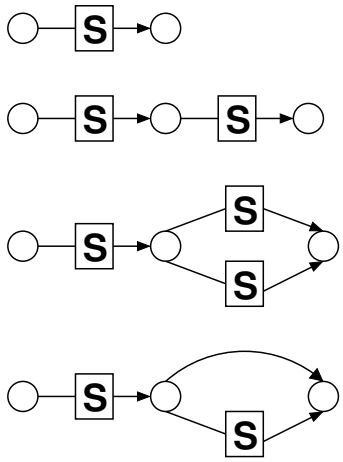

Fig. 2. A partial derivation using the productions in figure 1.

this ambiguity, each production specifies which nodes in the sub-graph are to be glued to which nodes in the host graph.

In general, non-terminal edges can be *hyperedges* that join more than two nodes. A hyperedge is said to be an n-edge if it joins n nodes. All of the hyperedges in the above example are 2-edges, or simple edges. If an n-edge labeled with non-terminal X is to be expanded, there must be a production with X as its left-hand side and a graph on its right-hand side that has n distinguished nodes (e.g. labeled 1 - n) that will be glued to the nodes that were attached to the hyperedge before it was removed.

3 Parameter Estimation for String Grammars

Our goal is to develop a set of algorithms for graph grammars that mirror those available for string grammars, with the starting point being an algorithm for estimating the parameters of graph grammars from data. This section reviews the most widely used algorithm for estimating the parameters of string grammars from data - the Inside-Outside (IO) algorithm (Lari & Young, 1990). This review will provide the necessary background for readers unfamiliar with IO and will make it possible to focus on issues specific to graph grammars in section 4 where we derive a version of IO for graph grammars (the PEGG algorithm).

Let $G = (\mathcal{S}, \theta)$ be a stochastic context-free string grammar with structure $\mathcal{S}$ and parameters θ. Let E be a set of training examples created by sampling from the probability distribution over strings defined by G. Given $\mathcal{S}$ and E, the goal of parameter estimation is to obtain a set of parameters, $\hat{\theta}$, such that $p(E|S, \hat{\theta})$ is maximized.

If G is unambiguous then maximum likelihood parameter estimation is easy. A grammar is *unambiguous* if every string in $L(G)$ has exactly one derivation (Hopcroft & Ullman, 1979). That is, given a string in $L(G)$ it is possible to determine which productions were used to derive the string. Let $c(X \to \gamma|s)$ be the number of times production $X \to \gamma$ is used in the derivation of string s. Let $c(X \to \gamma|E)$ be $\sum_{s \in E} c(X \to \gamma|s)$. Then the maximum likelihood estimate for $p(X \to \gamma)$ is:

$$\hat{p}(X \to \gamma) = \frac{c(X \to \gamma|E)}{\sum_\delta c(X \to \delta|E)}$$

The estimate is simply the number of times X was expanded to γ divided by the number of times X occurred.

If G is ambiguous then strings in $L(G)$ can have multiple derivations, and parameter estimation becomes more difficult. The problem is that only the strings in E are observable, not their derivations. Given a string $s \in E$, one of the possibly many derivations of s was actually used to generate the string when sampling from the probability distribution over strings defined by G. It is production counts from this derivation, and none of the other legal derivations, that are needed for parameter estimation.

This is an example of a hidden data problem. Given information about which derivation was used when sampling each $s \in E$, the estimation problem is easy, but we do not have this information. As is typical in such cases the solution is Expectation Maximization (EM) (Dempster et al., 1977). For string s with m possible derivations - $d_1, d_2, \ldots, d_m$ - we introduce indicator variables - $z_1, z_2, \ldots, z_m$, such that $z_i = 1$ if d_i is the derivation used when s was sampled. Otherwise, $z_i = 0$. The expected value of z_i can therefore be computed as follows:

$$\begin{aligned} E[z_i] &= 1 * p(z_i = 1) + 0 * p(z_i = 0) \\ &= p(z_i = 1) \\ &= p(d_i \text{ is the true derivation}) \\ &= \frac{p(d_i|G)}{\sum_j p(d_j|G)} \end{aligned}$$

In the expectation step, the indicator variables are used to compute expected counts:

$$\begin{aligned} \hat{c}(X \to \gamma|s) &= \sum_i E[z_i] c(X \to \gamma|d_i) \\ &= \frac{\sum_i p(d_i|G) c(X \to \gamma|d_i)}{\sum_j p(d_j|G)} \end{aligned} \tag{1}$$

In the maximization step, the expected counts are used to compute new maximum likelihood parameter estimates:

$$\hat{p}(X \to \gamma) = \frac{\hat{c}(X \to \gamma|E)}{\sum_\delta \hat{c}(X \to \delta|E)}$$

Iterating the E-step and the M-step is guaranteed to lead to a local maximum in the likelihood surface.

The only potential difficulty is that computing expected counts requires summing over all possible derivations of a string, of which there may be exponentially many. The Inside-Outside algorithm uses dynamic programming to compute these counts in polynomial time (Lari & Young, 1990). Our discussion of the algorithm will follow the presentation in (Charniak, 1993).

For string s and non-terminal X, let $s_{i,j}$ denote the sub-string of s ranging from the i^{th} to the j^{th} position, and let $X_{i,j}$ denote the fact that non-terminal X roots the subtree that derives $s_{i,j}$. We can now define the inside probability, $\beta_X(i,j)$, as the probability that X will derive $s_{i,j}$. More formally:

$$\beta_X(i,j) = p(s_{i,j}|X_{i,j})$$

The outside probability, $\alpha_X(i,j)$, is the probability of deriving the string $s_{1,i-1}Xs_{j+1,n}$ from the start symbol such that X spans $s_{i,j}$. More formally:

$$\alpha_X(i,j) = p(s_{1,i-1}, X_{i,j}, s_{j+1,n})$$

In the formula above, $n = |s|$. As figure 3 suggests, given that non-terminal X roots the sub-tree that derives $s_{i,j}$, the inside probability $\beta_X(i,j)$ is the probability of X deriving the part of s inside the sub-tree and the outside probability $\alpha_X(i,j)$ is the probability of the start symbol deriving the part of s outside the sub-tree.

How are α and β useful in parameter estimation? Rather than implementing equation 1 as a sum over derivations, we will soon see that knowing α and β makes it possible to compute expected counts by summing over all possible substrings of s that a given non-terminal can generate. For a string of length n there are $n(n-1)/2$ substrings, which is far fewer than the worst case exponential number of possible derivations.

For example, consider the somewhat simpler problem of computing the expected number of times X occurs in a derivation of string s. This non-terminal can potentially root sub-trees that generate any of the $n(n-1)/2$ substrings of s. The expected number of occurrences of X is thus given by the following sum:

$$\hat{c}(X) = \sum_{i,j} p(X_{i,j}|s)$$

This expression can be rewritten as follows in terms of inside and outside probabilities exclusively:

$$\begin{aligned}
\hat{c}(X) &= \sum_{i,j} p(X_{i,j}|s) \\
&= \frac{1}{p(s)} \sum_{i,j} p(X_{i,j}, s) \\
&= \frac{1}{p(s)} \sum_{i,j} p(s_{1,i-1}, s_{i,j}, s_{j+1,n}, X_{i,j})
\end{aligned}$$

$$= \frac{1}{p(s)} \sum_{i,j} p(s_{i,j}|X_{i,j}) p(s_{1,i-1}, X_{i,j}, s_{j+1,n})$$
$$= \frac{1}{p(s)} \sum_{i,j} \alpha_X(i,j) \beta_X(i,j)$$

The move from the first line to the second above is a simple application of the definition of conditional probability. We then expand s, apply the chain rule of probability, and finally substitute α and β.

Equation 1 requires $\hat{c}(X \to \gamma)$, not $\hat{c}(X)$. Suppose for the moment that our grammar is in Chomsky Normal Form. That is, all productions are of the form $X \to YZ$ or $X \to \sigma$ where X, Y, and Z are non-terminals and σ is a terminal. To compute $\hat{c}(X \to \gamma)$, rather than just summing over all possible substrings that X can generate, we sum over all possible substrings that X can generate and all possible ways that Y and Z can carve up the substring. Consider figure 3. If X generates $s_{i,j}$ and X expands to YZ, then concatenating the substring generated by Y with the substring generated by Z must yield $s_{i,j}$.

The expected counts for $X \to YZ$ are defined to be:

$$\hat{c}(X \to YZ) = \sum_{i,j,k} p(X_{i,j}, Y_{i,k}, Z_{k+1,j}|s)$$

It is easy to show that this is equivalent to:

$$\hat{c}(X \to YZ) = \frac{1}{p(s)} \sum_{i,j,k} \beta_Y(i,k) \beta_Z(k+1,j) \alpha_X(i,j) p(X \to YZ) \quad (2)$$

A complete derivation will be given in the next section when a formula for computing expected counts for graph grammars is presented.

Clearly, evaluating equation 2 requires $O(n^3)$ computation, in addition to that required to compute α and β. For string grammars, tables of α and β values are computed via dynamic programming in $O(m^2 n)$ time where m is the number of non-terminals in the grammar. Section 4 will show a complete derivation of the formulas for computing α and β in the context of graph grammars.

4 Parameter Estimation for Graph Grammars

In this section we define and derive analogs of inside and outside probabilities for graph grammars. Just as α and β can be computed efficiently, top down and bottom up respectively, for string grammars by combining sub-strings, they can be computed efficiently for graph grammars by combining sub-graphs. While there are only polynomially many sub-strings of any given string, there in general can be exponentially many sub-graphs of any given graph. It turns out there there is a natural class of graphs (Lautemann, 1990) for which the number of sub-graphs that one must consider when computing α and β is polynomial in

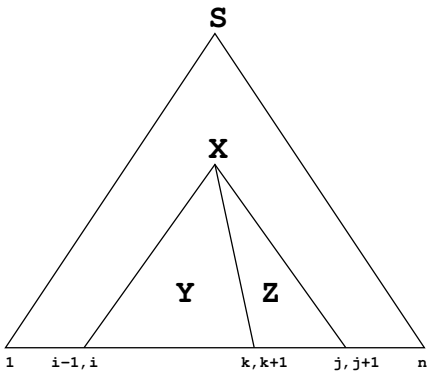

Fig. 3. Given that X derives $s_{i,j}$ and that X expands to YZ, there are only $j - i + 1$ ways that Y and Z can carve up $s_{i,j}$.

the size of the graph. For this type of grammar, maximum likelihood parameter estimates can be computed in polynomial time.

For non-terminal hyperedge X and graph g we define the inside probability $\beta_X(g)$ to be $p(g|X)$, the probability that X will derive g. Note that $\beta_S(g)$ is the probability of g in the distribution over graphs defined by the grammar. There are two cases to consider - either X derives g in one step, or X derives some other sub-graph in one step and g can be derived from that sub-graph in one or more steps:

$$\begin{aligned} \beta_X(g) &= p(g|X) \\ &= p(X \rightarrow g) + \sum_{X \rightarrow \gamma} p(X \rightarrow \gamma) p(\gamma \stackrel{*}{\rightarrow} g) \end{aligned} \tag{3}$$

In equation 3, we use $\rightarrow$ to denote derivation in one step via a production in the grammar and $\stackrel{*}{\rightarrow}$ to denote derivation in one or more steps.

The difficult part of evaluating equation 3 is computing $p(\gamma \stackrel{*}{\rightarrow} g)$. Because γ is the right-hand side of a production it can be an arbitrary hypergraph. Suppose γ has m hyperedges - $h_1, h_2, \ldots, h_m$. If γ can derive g, then there must be m graphs - $g_1, g_2, \ldots, g_m$ - such that h_i derives g_i for $1 \leq i \leq m$ and the graph that results from replacing each h_i with the corresponding g_i is isomorphic to g. Note that each g_i must be isomorphic to a sub-graph of g for this to occur. It is therefore theoretically possible to determine if γ (and thus X) can derive g by generating all possible sub-graphs of g, forming all ordered sets of these sub-graphs of size m, generating the graphs that result from substituting the sub-graphs in each ordered set for the hyperedges in γ, and testing to see if any of the resulting graphs are equal to g.

Because the sub-graphs are taken directly from g the equality test can be performed in polynomial time (i.e. a test for graph isomorphism is not required).

However, there may be exponentially many sub-graphs. As stated earlier, we will restrict our attention to a robust class of graphs for which the number of sub-graphs one must consider is small (polynomial). Let's finish our derivation of $\beta_X(g)$ before getting to the details of computing it efficiently.

Let $\Psi(\gamma, g)$ be the set that results from computing all ordered sub-sets of size m of the set of all sub-graphs of g. Recall that γ is a hypergraph with m hyperedges (i.e. non-terminals). Let $\Psi_i(\gamma, g)$ be the i^{th} element of this set. Each element of Ψ represents a mapping of hyperedges in γ to structure in g. If any of these mappings yield g, then it is the case that γ can derive g.

To compute $p(\gamma \stackrel{*}{\rightarrow} g)$ we simply need to iterate over each element of $\Psi(\gamma, g)$ and compute the probability of the joint event that each of the h_i derive each of the g_i and sum this probability for each element that produces a graph equal to g. That is, for each $\Psi_i(\gamma, X)$ we need to compute:

$$p(h_1^i \stackrel{*}{\rightarrow} g_1^i, h_2^i \stackrel{*}{\rightarrow} g_2^i, \ldots, h_m^i \stackrel{*}{\rightarrow} g_m^i)$$

Because HR graph grammars are context free, derivations that start from different hyperedges are completely independent of one another (Courcelle, 1987). Therefore, the probability of the joint event is equivalent to the product of the probabilities of the individual events. That is:

$$p(h_1^i \stackrel{*}{\rightarrow} g_1^i, h_2^i \stackrel{*}{\rightarrow} g_2^i, \ldots, h_m^i \stackrel{*}{\rightarrow} g_m^i) = \prod_{j=1}^{m} p(h_j^i \stackrel{*}{\rightarrow} g_j^i)$$

Combining the above with equation 3 yields an expression for $\beta_X(g)$ in terms of other inside probabilities:

$$\begin{aligned}
\beta_X(g) &= p(X \rightarrow g) + \sum_{X \rightarrow \gamma} p(X \rightarrow \gamma) p(\gamma \stackrel{*}{\rightarrow} g) \\
&= p(X \rightarrow g) + \sum_{X \rightarrow \gamma} p(X \rightarrow \gamma) \sum_i p(\Psi_i(\gamma, g)) \\
&= p(X \rightarrow g) + \sum_{X \rightarrow \gamma} p(X \rightarrow \gamma) \sum_i p(h_1^i \stackrel{*}{\rightarrow} g_1^i, h_2^i \stackrel{*}{\rightarrow} g_2^i, \ldots, h_m^i \stackrel{*}{\rightarrow} g_m^i) \\
&= p(X \rightarrow g) + \sum_{X \rightarrow \gamma} p(X \rightarrow \gamma) \sum_i \prod_{j=1}^{m} p(h_j^i \stackrel{*}{\rightarrow} g_j^i) \\
&= p(X \rightarrow g) + \sum_{X \rightarrow \gamma} p(X \rightarrow \gamma) \sum_i \prod_{j=1}^{m} p(g_j^i | h_j^i) \\
&= p(X \rightarrow g) + \sum_{X \rightarrow \gamma} p(X \rightarrow \gamma) \sum_i \prod_{j=1}^{m} \beta_{h_j^i}(g_j^i) \qquad (4)
\end{aligned}$$

Equation 4 makes it possible to compute inside probabilities in terms of other inside probabilities. Note that this computation can proceed bottom up because the sub-graphs considered in the inner sum, i.e. the recursive computation of $\beta_{h_j^i}(g_j^i)$, must be smaller than g because they are composed via γ to

yield g. That is, one can compute β for sub-graphs containing one node, then sub-graphs containing two nodes, and so on. The number of levels in this bottom up computation is bounded by the size of g. The outer summation is linear in the number of productions in the grammar, and the product is linear in the maximum number of hyperedges in any right-hand side, which we assume to be bounded by a small constant. However, the inner sum iterates over all elements of Ψ, of which there can be exponentially many.

If the number of sub-graphs considered in the inner sum in equation 4 were polynomial, then all inside probabilities could be computed in polynomial time. Lautemann (Lautemann, 1990) defines a class of HR grammars for which this is the case, i.e. grammars with *logarithmic k-separability*.

The k-separability of graph g (see definition 3.2.3 in (Lautemann, 1990)) is the maximum number of connected components that can be produced by removing k or fewer nodes from g. This definition becomes useful for our current purposes when considered in conjunction with lemma 3.2.1 from (Lautemann, 1990). To build intuition before stating the lemma, consider how you might try to determine if a hypergraph, γ, with a single hyperedge, h, can generate a given graph, g. Note that all of the nodes and edges in the hypergraph must appear in the final graph. You might therefore try all possible mappings of nodes and edges in the hypergraph to nodes and edges in the graph, and see if the hyperedge can generate the unmapped remainder of the graph.

The lemma says, essentially, that if replacing hyperedge h in hypergraph γ with graph g' yields graph g, then every connected component in g' minus the nodes in h is a connected component in g minus the nodes in h. That is, if you map the nodes in γ to nodes in g and then remove those mapped nodes from g and find the connected components in the resulting graph, you will have enumerated (at least) all of the connected components in the sub-graph with which h should be replaced to derive g.

Therefore, rather than enumerating all possible sub-graphs of g to determine if $\gamma \stackrel{*}{\rightarrow} g$, we can form all possible mappings of nodes and edges in γ onto g, compute the connected components that result when the mapped nodes are removed from g, and consider only those sub-graphs that are combinations of these connected components. Because γ is the right-hand side of a production and we assume that its size is bounded by a small constant, the number of possible mappings of γ onto g is polynomial in the size of g. If we further assume that the k-separability of the graph is logarithmic, then the number of connected components formed for each mapping of γ onto g is $O(\log |g|)$ and there are only polynomially many possible combinations of connected components. In polynomial time we can compute all of the subgraphs that need to be considered in the inner sum of equation 4, of which there are polynomially many. All of the inside probabilities can therefore be computed in polynomial time.

Intuitively, bounded k-separability requires that graphs have bounded degree and be connected. Consider the language containing all star graphs, i.e. graphs containing n nodes where nodes $2 - n$ have a single edge to node 1. If node 1 is removed, $n - 1$ connected components are created. At the other extreme,

consider a graph of n nodes and no edges. Removing any one node results in a graph with $n-1$ connected components. In both cases, the k-separability of the graph is linear in the size of the graph. For k-separability to have a lower bound, there must be a bound on node degree and the graph must be (mostly) connected.

We now turn to the derivation of the outside probability. Recall that the inside probability $\beta_X(g)$ is the probability that a non-terminal hyperedge labeled X will generate graph g. In practice, given a graph G, β values are computed for sub-graphs of G. That is, $\beta_X(g)$ is computed for values of g corresponding to different sub-graphs of some fixed graph G. The outside probability $\alpha_X(g)$ is the probability that the start symbol will generate the graph formed by replacing sub-graph g in graph G with a non-terminal hyperedge labeled X. It is called the outside probability because α is the probability of generating the graph structure in G outside the sub-graph generated by X. Note that the quantity $\alpha_X(g)\beta_X(g)$ is the probability of generating G in such a way that nonterminal X generates sub-graph g.

How might non-terminal X become responsible for generating sub-graph g? Suppose Y is a non-terminal, $Y \to \gamma$ is a production in the grammar, and γ contains a hyperedge labeled X. Further, let g' be a subgraph of G that contains g. If Y is responsible for generating g', then it could be the case that X generates g and the remainder of γ generates the remainder of g'. That is, we can compute outside probabilities from outside probabilities of larger subgraphs.

Let X_g denote the fact that X generates subgraph g, and h(γ) denote the set of hyperedges contained in hypergraph γ. The outside probability can be derived as follows:

$$
\begin{aligned}
\alpha_X(g) &= p(X_g, G-g) \\
&= \sum_{Y\to\gamma, X\in h(\gamma)} \sum_{g', g\subseteq g'} p(G-g', g'-g, X_g, Y_{g'}, (\gamma - X)_{g'-g}) \\
&= \sum_{Y\to\gamma, X\in h(\gamma)} \sum_{g', g\subseteq g'} p(Y_{g'}, G-g') p(X_g, (\gamma-X)_{g'-g} | Y_{g'}, G-g') \\
&\qquad\qquad p(g'-g | Y_{g'}, G-g', X_g, (\gamma-X)_{g'-g}) \\
&= \sum_{Y\to\gamma, X\in h(\gamma)} \sum_{g', g\subseteq g'} p(Y_g, G-g') p(X_g, (\gamma-X)_{g'-g} | Y_{g'}) \\
&\qquad\qquad p(g'-g | (\gamma-X)_{g'-g}) \\
&= \sum_{Y\to\gamma, X\in h(\gamma)} \sum_{g', g\subseteq g'} \alpha_Y(g') p(Y\to\gamma) p((\gamma-X)_{g'-g} \stackrel{*}{\to} g'-g) \\
&= \sum_{Y\to\gamma, X\in h(\gamma)} \sum_{g', g\subseteq g'} \alpha_Y(g') p(Y\to\gamma) \sum_i \prod_{j\in h(\gamma)-X} \beta_{h_j^i}(g_j^i) \qquad (5)
\end{aligned}
$$

The second line is derived from the first by summing over all productions, $Y \to \gamma$, for which X is a hyperedge in γ, and all subgraphs, g', such that γ generates g', g is a subgraph of g', and X is a hyperedge in γ that generates g. Then we apply the chain rule of probability, and substitute α and β. The move from the

fifth line to the sixth line involves the same steps used in the derivation of the formula for inside probabilities that expand $p(\gamma \stackrel{*}{\rightarrow} g)$ into an expression that can be computed in terms of other inside probabilities.

Outside probabilities can be computed from the top down, with the base case being $\alpha_S(G) = 1$. That is, with probability 1 the start symbol is responsible for generating any graph in the language of the grammar. As with β, all of the sums and products are polynomial in the size of the graph except the one that iterations over the elements of $\Psi(\gamma, G)$. However, as with β, if the graph has logarithmic k-separability, there are only polynomially many elements of $\Psi(\gamma, G)$ to consider. Therefore, all outside probabilities can be computed in polynomial time.

Once we know the inside and outside probabilities, we can compute the expected counts of each production in the grammar. For productions in the form of $X \rightarrow \gamma$, these counts are computed as follows:

$$\begin{aligned}
\hat{c}(X \rightarrow \gamma) &= \sum_g p(X_g, \gamma | G) \\
&= \frac{1}{p(G)} \sum_g p(X_g, \gamma, G) \\
&= \frac{1}{p(G)} \sum_g p(X_g, \gamma, G - g, g) \\
&= \frac{1}{p(G)} \sum_g p(G - g, X_g) p(\gamma | X_g) p(g | \gamma) \\
&= \frac{1}{p(G)} \sum_g \alpha_X(g) p(X \rightarrow \gamma) p(\gamma \stackrel{*}{\rightarrow} g) \\
&= \frac{1}{p(G)} \sum_g \alpha_X(g) p(X \rightarrow \gamma) \sum_i \prod_{j=1}^{m} \beta_{h_j^i}(g_j^i)
\end{aligned}$$

For production in the form of $X \rightarrow g$, i.e. the right-hand side is a terminal graph, the counts are computed as follows:

$$\begin{aligned}
\hat{c}(X \rightarrow g) &= \sum_{g'} p(X'_g, g' = g | G) \\
&= \frac{1}{p(G)} \sum_{g'} p(X'_g, g' = g, G) \\
&= \frac{1}{p(G)} \sum_{g'} p(X_g, g' = g, G - g', g') \\
&= \frac{1}{p(G)} \sum_{g'} p(G - g', X'_g) p(g', g' = g | X'_g) \\
&= \frac{1}{p(G)} \sum_{g'} \alpha_X(g') p(X \rightarrow g, g' = g)
\end{aligned}$$

In the above formulas, $g' = g$ denotes that g' is isomorphic to g.

5 Preliminary Empirical Results

This section reports the results of some simple, preliminary experiments with an implementation of PEGG. Let $G = (\mathcal{S}, \theta)$ be a stochastic context-free HR graph grammar with structure $\mathcal{S}$ and parameters θ. Let E be a set of training examples created by sampling from the probability distribution over graphs defined by G. Given $\mathcal{S}$ and E, the goal of PEGG is to obtain a set of parameters, $\hat{\theta}$, such that $p(E|S, \hat{\theta})$ is maximized. We used the grammar shown in figure 1 for $\mathcal{S}$, and the true parameters were $\theta = (0.6, 0.2, 0.2)$. That is, the probability of expanding a hyperedge labeled S with the first production is 0.6, with the second production is 0.2, and with the third production is 0.2.

In the first experiment we sampled 1, 5, and 10 graphs from the grammar and ran PEGG on these samples. The learned parameters are shown in table 1. In all cases the parameters appear to be "reasonable", but they do deviate from the desired parameters. This might be due to the fact that the samples are small and are therefore not representative of the true distribution over graphs defined by the grammar. To test this hypothesis, we took another sample of 10 graphs for which the estimated parameters deviated significantly from the true parameters. Given the derivations of the 10 graphs, it was a simple matter to count the number of times the various productions were applied and manually compute maximum likelihood parameters. Both the estimated parameters and the ML parameters are shown in table 2. Note that the sample of 10 graphs was clearly not representative of the distribution over graphs defined by the true parameters. The Kullback-Leibler divergence between $\theta = (0.6, 0.2, 0.2)$ and $\theta_{ML} = (0.75, 0.15, 0.10)$ is 0.8985. However, PEGG did a good job of estimation. The KL divergence between θ_{PEGG} and θ_{ML} is 0.007, two orders of magnitude less than the divergence with the true parameters.

Table 1. Parameters estimated by PEGG for samples of size 1, 5, and 10.

$\|E\|$	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$
1	0.5714	0.2857	0.1429
5	0.6206	0.2168	0.1626
10	0.6486	0.2973	0.0541

Finally, to determine if PEGG was finding parameters that actually maximize the likelihood of the data, we computed the log-likelihood of a sample of 5 graphs given the true parameters and the parameters estimated for that sample. The log-likelihood of the data given the true parameters was -9.38092, and it was -9.40647 given the estimated parameters, a difference of less than three-tenths of one percent.

Table 2. Estimated parameters and manually computed ML parameters for a sample of size 10.

	PEGG	ML
$\hat{\theta}_1$	0.7368	0.75
$\hat{\theta}_2$	0.1579	0.15
$\hat{\theta}_3$	0.1053	0.10

6 Conclusion

This paper introduced the Parameter Estimation for Graph Grammars (PEGG) algorithm, the first algorithm for estimating the parameters of stochastic context-free hyperedge replacement graph grammars. PEGG computes inside and outside probabilities in polynomial time for graphs with logarithmic k-separability. In addition, PEGG uses these probabilities to compute maximum likelihood parameters for a fixed grammar structure and a sample of graphs drawn from some probability distribution over graphs.

Despite the fact that graph grammars have been an active area of research since the late 1960's, almost no work has dealt with stochastic graph grammars. One notable exception is (Mosbah, 1994), which explores the properties of graphs sampled from stochastic graph grammars.

There are only a handful of papers that directly address the problem of learning graph grammars, and none other than the current paper that leverage the vast body of work on inferring string grammars from data. (Bartsch-Sprol, 1983) describes an enumerative (i.e. computationally infeasible) method for inferring a restricted class of context-sensitive graph grammars. (Jeltsch & Kreowski, 1991) describes an algorithm for extracting common hyperedge replacement substructures from a set of graphs via merging techniques. This work is similar to that reported in (Jonyer et al., 2002) in which merging techniques were used to extract node replacement sub-structures from a set of graphs. Fletcher (Fletcher, 2001) developed a connectionist method for learning regular graph grammars. To the best of our knowledge, our paper is the first to present a formally sound algorithm for computing maximum likelihood parameter estimates for a large class of HR graph grammars.

Future work will involve developing an approach to inferring the structure of HR graph grammars based on Bayesian model merging techniques, similar to those we developed for node replacement grammars (Doshi et al., 2002). In combination with the PEGG algorithm described in this paper the result will be a powerful tool for inferring HR graph grammars from data. In addition, we are considering applications of this tool in the domain of bioinformatics. Finally, we are attempting to understand the relationship between stochastic context-free graph grammars and stochastic definite clause grammars as used in stochastic logic programming (Muggleton, 1996).

Bibliography

Bartsch-Sprol, B. (1983). Grammatical inference of graph grammars for syntactic pattern recognition. In H. Ehrig, M. Nagl and G. Rozenberg (Eds.), *Proceedings of the second international workshop on graph grammars and their applications to computer science*. Springer Verlag.

Charniak, E. (1993). *Statistical language learning*. MIT Press.

Cook, D., Holder, L. B., Su, S., Maglothin, R., & Jonyer, I. (2001). Structural mining of molecular biology data. *IEEE Engineering in Medicine and Biology*, *20*, 231–255.

Courcelle, B. (1987). An axiomatic definition of context-free rewriting and its application to NLC graph grammars. *Theoretical Computer Science*, *55*, 141–181.

Courcelle, B. (1997). The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg (Ed.), *Handbook of graph grammars and computing by graph transformation: Foundations*. World Scientific Publishing Company.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B*, *39*, 1–38.

Doshi, S., Huang, F., & Oates, T. (2002). Inferring the structure of graph grammars from data. . Proceedings of the International Conference on Knowledge-Based Computer Systems.

Drewes, F., Kreowski, H. J., & Habel, A. (1997). Hyperedge replacement graph grammars. In G. Rozenberg (Ed.), *Handbook of graph grammars and computing by graph transformation: Foundations*. World Scientific Publishing Company.

Ehrig, H., Engels, G., Kreowski, H.-J., & Rozenberg, G. (Eds.). (1999). *Handbook of graph grammars and computing by graph transformation: Applications, languages and tools*. World Scientific Publishing Company.

Engelfriet, J., & Rozenberg, G. (1997). Node replacement graph grammars. In G. Rozenberg (Ed.), *Handbook of graph grammars and computing by graph transformation: Foundations*. World Scientific Publishing Company.

Fletcher, P. (2001). Connectionist learning of regular graph grammars. *Connection Science*, *13*, 127–188.

Hong, P., & Huang, T. S. (2002). Spatial pattern discovery by learning a probabilistic parametric model from multiple attributed relational graphs. *Journal of Discrete Applied Mathematics*.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages and computation*. Addison-Wesley Publishing Company.

Immerman, N. (1999). *Descriptive complexity*. Springer.

Jeltsch, E., & Kreowski, H. J. (1991). Grammatical inference based on hyperedge replacement. *Lecture Notes in Computer Science*, *532*, 461–474.

Jonyer, I., Holder, L. B., & Cook, D. J. (2002). Concept formation using graph grammars. *Working Notes of the KDD Workshop on Multi-Relational Data Mining.*

Lari, K., & Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, *4*, 35–56.

Lautemann, C. (1990). The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, *27*, 399–421.

Mosbah, M. (1994). Properties of random graphs generated by probabilistic graph grammars. *Proceedings of the The Fifth International Workshop on Graph Grammars and their Application to Computer Science.*

Muggleton, S. (1996). Stochastic logic programs. In L. De Raedt (Ed.), *Advances in inductive logic programming*, 254–264.

Office of Technology Assessment, U. C. (1995). Information technologies for control of money laundering. OTA-ITC-360.

Pattison, P. E. (1993). *Algebraic models for social networks.* Cambridge University Press.

Stolcke, A. (1994). *Bayesian learning of probabilistic language models.* Doctoral dissertation, University of California, Berkeley.