In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*,
Portland, OR, USA, August 1996

Generalized Queries on Probabilistic Context-Free Grammars

David V. Pynadath and **Michael P. Wellman**
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109 USA
{pynadath,wellman}@umich.edu

Abstract

Probabilistic context-free grammars (PCFGs) provide a simple way to represent a particular class of distributions over sentences in a context-free language. Efficient parsing algorithms for answering particular queries about a PCFG (i.e., calculating the probability of a given sentence, or finding the most likely parse) have been applied to a variety of pattern-recognition problems. We extend the class of queries that can be answered in several ways: (1) allowing missing tokens in a sentence or sentence fragment, (2) supporting queries about intermediate structure, such as the presence of particular nonterminals, and (3) flexible conditioning on a variety of types of evidence. Our method works by constructing a Bayesian network to represent the distribution of parse trees induced by a given PCFG. The network structure mirrors that of the chart in a standard parser, and is generated using a similar dynamic-programming approach. We present an algorithm for constructing Bayesian networks from PCFGs, and show how queries or patterns of queries on the network correspond to interesting queries on PCFGs.

Introduction

Most pattern-recognition problems start from observations generated by some structured stochastic process. Probabilistic context-free grammars (PCFGs) (Gonzalez & Thomason 1978; Charniak 1993) have provided a useful method for modeling uncertainty in a wide range of structures, including programming languages (Wetherell 1980), images (Chou 1989), speech signals (Ney 1992), and RNA sequences (Sakakibara *et al.* 1995). Domains like plan recognition, where non-probabilistic grammars have provided useful models (Vilain 1990), may also benefit from an explicit stochastic model.

Once we have created a PCFG model of a process, we can apply existing PCFG parsing algorithms to answer a variety of queries. However, these techniques are limited in the types of evidence they can exploit and the types of queries they can answer. In particular, the standard techniques generally require specification of a complete observation sequence. In many contexts, we may have only a partial sequence available, or other kinds of contextual evidence. In addition, we may be interested in computing the probabilities of types of events that the extant techniques do not directly support. Finally, the PCFG model itself imposes restrictions on the probabilistic dependence structure, which we may wish to relax.

To extend the forms of evidence, queries, and distributions supported, we need a flexible and expressive representation for the distribution of structures generated by the grammar. We adopt Bayesian networks for this purpose, and define an algorithm to generate a network representing the distribution of possible parse trees corresponding to a given PCFG. We then present algorithms for extending the class of queries to include the conditional probability of a symbol appearing anywhere within any region of the parse tree, conditioned on any evidence about symbols appearing in the parse tree. The Bayesian network also provides a flexible structure for future extensions to context-sensitive probabilities, similar to the probabilistic parse tables of (Briscoe & Carroll 1993).

Probabilistic Context-Free Grammars

A probabilistic context-free grammar is a tuple $\langle H_T, H_N, E^1, P \rangle$, where H_T is the set of terminal symbols, H_N the set of nonterminal symbols, $E^1 \in H_N$ the start symbol, and P the set of productions. Productions take the form $E \rightarrow \xi\ (p)$, with $E \in H_N$, $\xi \in (H_T \cup H_N)^+$, and $p = \Pr(E \rightarrow \xi)$, the probability that E will be expanded into the string ξ. The probability of applying a particular production to an intermediate string is conditionally independent of what productions were previously applied to obtain the current string, or what productions will be applied to the other symbols in the current string, given the presence of the left-hand symbol. Therefore, the probability of a given derivation is simply the product of the probabilities of the individual productions involved. The probability of a string in the language is the sum taken over all possible derivations. In the grammar (from (Charniak 1993)) shown in Figure 1, the start symbol is S.

s	→	np vp	(0.8)	pp	→	p np	(1.0)
s	→	vp	(0.2)	p	→	like	(1.0)
np	→	n	(0.4)	v	→	swat	(0.2)
np	→	n pp	(0.4)	v	→	flies	(0.4)
np	→	n np	(0.2)	v	→	like	(0.4)
vp	→	v	(0.3)	n	→	swat	(0.05)
vp	→	v np	(0.3)	n	→	flies	(0.45)
vp	→	v pp	(0.2)	n	→	ants	(0.5)
vp	→	v np pp	(0.2)				

Figure 1: A probabilistic context-free grammar.

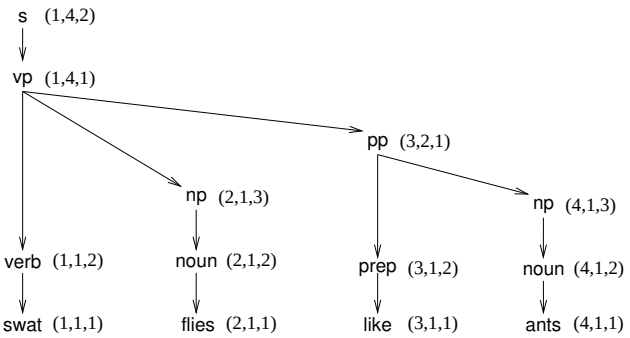

Figure 2: Parse tree for Swat flies like ants, with (i, j, k) indices labeled.

Indexing Parse Trees

Calculating the probability of a particular parse tree can sometimes be useful, but we may also wish to derive the probability of some more abstract feature of a parse tree. To pose such queries, we require a scheme to specify events as the appearance of symbols at designated points in the parse tree. We use three indices to identify a node in a parse in terms of the structure of the subtree rooted at that node. Two indices delimit the leaf nodes of the subtree, defining a substring of the entire terminal sequence. The index i refers to the position of the substring within the entire terminal string, with $i = 1$ indicating the start of the string. The index j refers to the length of the substring. For example, the pp node in the parse tree of Figure 2 is the root of the subtree whose leaf nodes are like and ants, so $i = 3$ and $j = 2$. These i and j indices are commonly used in PCFG algorithms.

However, we cannot always uniquely specify a node with these two indices alone. In the branch of the parse tree passing through np, n, and flies, all three nodes have $i = 2$ and $j = 1$. To differentiate them, we introduce the k index, defined recursively. If a node has no child with the same i and j indices, then it has $k = 1$. Otherwise, its k index is one more than the k index of its child. Thus, the flies node has $k = 1$, the n node above it has a $k = 2$, and its parent np has $k = 3$. We have labeled each node in the parse tree of Figure 2 with its (i, j, k) indices.

We can think of the k index of a node as its level of abstraction, with higher values indicating more abstract symbols. For instance, the flies symbol is a specialization of the n concept, which, in turn, is a specialization of the np concept. Each possible specialization corresponds to an *abstraction production* of the form $E \rightarrow E'$. In a parse tree involving such a production, the nodes for E and E' will have identical i and j values, but the k value for E will be one more than that of E'. We denote the set of abstraction productions as $P_A \subseteq P$.

All other productions are *decomposition productions*, in the set $P_D = P \setminus P_A$, and have two or more symbols on the right-hand side. If a node E is expanded by a decomposition production, the sum of the j values for its children will equal its own j value, since the length of the original substring derived from E must equal the total lengths of the substrings of its children. In addition, since each child must derive a string of nonzero length, no child has the same j index as E, which must then have a k value of 1. Therefore, abstraction productions connect nodes whose indices match in the i and j components, while decomposition productions connect nodes whose indices differ.

Dynamic Programming Algorithm

We can compute the probability of a string by summing probabilities over the set of its possible parse trees, which grows exponentially with the string's length. Fortunately, parse trees often share common subtrees, a fact exploited by the standard dynamic programming approach for both probabilistic and non-probabilistic CFGs (Jelinek, Lafferty, & Mercer 1992). The central structure is a table, or *chart*, storing previous results for each substring in the input sentence. Each entry in the chart corresponds to a substring $x_i \cdots x_{i+j-1}$ (ignoring abstraction level, k) of the observation string $x_1 \cdots x_L$. For each symbol E, an entry contains the probability that the corresponding substring is derived from that symbol, $\Pr(x_i \cdots x_{i+j-1}|E)$.

At the bottom of the table are the results for substrings of length one, and the top entry holds the result for the entire string, $\Pr(x_1 \cdots x_L|E^1)$, which is exactly the probability of the observed string. We can compute these probabilities bottom-up, since we know that $\Pr(x_i|E) = 1$ if E is the observed symbol x_i, and 0 otherwise. We can define all other probabilities recursively as the sum, over all productions $E \rightarrow \xi\ (p)$, of the product of p and the probability $\Pr(x_i \cdots x_{i+j-1}|\xi)$. Here, we can make use of the PCFG independence assumptions, and compute this probability as the product of the probabilities of the individual symbols, where we have to consider all possible substring lengths for these symbols. A slight alteration to this procedure also allows us to obtain the most probable parse tree for the observed string.

To compute the probability of the sentence Swat flies like ants, we would use the algorithm to generate the table shown in Figure 3, after eliminating any intermediate entries

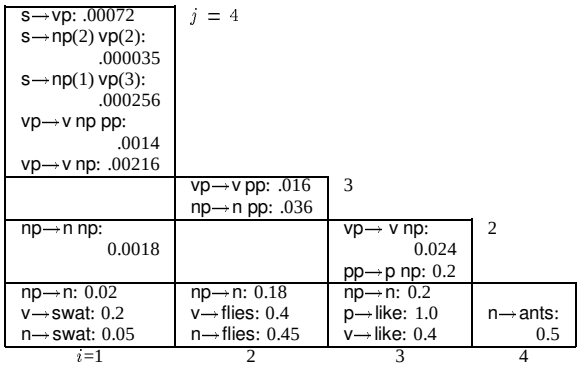

Figure 3: Chart for Swat flies like ants.

that were not referenced by higher-level entries. There are also separate entries for each production, though this is not necessary if we are only interested in the final sentence probability. In the top entry, there are two listings for the production s→np vp, with different substring lengths for the right-hand side symbols. The sum of all probabilities for productions with s on the right-hand side in this entry yields the total sentence probability of 0.001011.

This algorithm is capable of computing any "inside" probability, the probability of a particular terminal string appearing inside the subtree rooted by a particular nonterminal. We can work top-down in an analogous manner to compute any "outside" probability (Charniak 1993), the probability of a subtree rooted by a particular nonterminal appearing amid a particular terminal string. Given these probabilities we can compute the probability of any particular nonterminal symbol appearing in the parse tree as the root of a subtree covering some substring. For example, in the sentence Swat flies like ants, we can compute the probability that like ants is a prepositional phrase, using a combination of inside and outside probabilities. The Left-to-Right Inside (LRI) algorithm (Jelinek, Lafferty, & Mercer 1992) specifies how we can manipulate certain probability matrices and combine the results with the inside probabilities to obtain the probability of a given initial substring, such as the probability of a sentence (of any length) beginning with the words Swat flies. Furthermore, we can use such initial substring probabilities to compute the conditional probability of the next observation given all previous observations.

However, there are still many types of queries not covered by existing algorithms. For example, given observations of arbitrary partial observation strings, it is unclear how to exploit the standard chart directly. Similarly, we are unaware of methods to handle observation of nonterminals only (e.g., the last two words form a prepositional phrase). We seek, therefore, a mechanism that would admit observational evidence of any form as part of a query about a PCFG, without requiring us to enumerate all consistent parse trees.

Bayesian Networks for PCFGs

Bayesian networks (Pearl 1987) provide an expressive and efficient representation for probability distributions. They are expressive in that they can represent any joint distribution over a finite set of discrete-valued random variables. They are efficient in that they exploit an important class of conditional independence relationships among the random variables. Moreover, Bayesian networks are convenient computational devices, supporting the calculation of arbitrary conditional probability expressions involving their random variables. Therefore, if we can create a Bayesian network representing the distribution of parse trees for a given probabilistic grammar, then we can incorporate partial observations of a sentence as well as other forms of evidence, and determine the resulting probabilities of various features of the parse trees.

We base our Bayesian-network encoding of PCFGs on the parse tree indexing scheme presented in the previous section. The random variable N_{ijk} denotes the symbol in the parse tree at the position indicated by the (i, j, k) indices. Index combinations not appearing in the tree correspond to N variables taking on the null value nil. To simplify the dependency structure, we also introduce random variables P_{ijk} to represent the productions that expand the corresponding symbols N_{ijk}. However, the identity of the production is not quite sufficient to render the corresponding children in the parse tree conditionally independent, so we dictate that the P variable take on different values for each breakdown of the right-hand symbols' substring lengths. This increases the state space of the variables, but simplifies the dependency structure.

Dynamic Programming Phase

To complete the specification of the network, we identify the symbols and productions making up the domains of our random variables, as well as the conditional probability tables representing their dependencies. The PCFG specifies the relative probabilities of different productions for each nonterminal, but to specify the probabilities of alternate parse trees in terms of the N_{ijk} variables we need the probabilities of the length breakdowns. We can calculate these with a modified version of the standard dynamic programming algorithm sketched in the previous section.

This modified algorithm constructs a chart based on the set of all possible terminal strings, up to a bounded length n. Our resulting chart defines a function $\beta(E, j, k)$ (analogous to the inside probability in the standard parsing algorithm), specifying the probability that symbol E is the root node of a subtree, at abstraction level k, with a terminal substring of length j. Because this probability is not relative to a particular observation string, we can ignore the i index.

As in the previous dynamic programming algorithms, we can define this function recursively, initializing the entries to

k	E	$\beta(E,4,k)$
2	s	0.02016
1	s	0.0832
	np	0.0672
	vp	0.1008
	pp	0.176

k	E	$\beta(E,3,k)$
2	s	0.0208
1	s	0.0576
	np	0.176
	vp	0.104
	pp	0.08

k	E	$\beta(E,2,k)$
2	s	0.024
1	s	0.096
	np	0.08
	vp	0.12
	pp	0.4

k	E	$\beta(E,1,k)$
4	s	0.06
3	np	0.4
	vp	0.3
2	p	1.0
	v	1.0
	n	1.0
1	like	1.0
	swat	1.0
	flies	1.0
	ants	1.0

Figure 4: Final table for sample grammar.

0. Again, we start at $j = 1$ and work upward to $j = n$. For each terminal symbol x, $\beta(x, 1, 1) = 1$. For $k > 1$, only abstraction productions are possible, because, as discussed before, decomposition productions are applicable only when $k = 1$. For each abstraction production $E \rightarrow E'\ (p)$, we increment $\beta(E, j, k)$ by $p \cdot \beta(E', j, k-1)$. If $k = 1$, only decomposition productions are applicable, so for each decomposition production $E \rightarrow E_1 E_2 \cdots E_m\ (p)$, each substring length breakdown $j_1, \ldots, j_m$ (such that the $\sum_t j_t = j$), and each abstraction level k_t legal for each j_t, we increment $\beta(E, j, k)$ by $p \cdot \prod_{t=1}^{m} \beta(E_t, j_t, k_t)$. The table of Figure 4 lists the nonzero β values for our grammar over strings of maximum length 4.

For analysis of the complexity of this algorithm, it is useful to define d as the maximum abstraction level, and m as the maximum number of symbols on a production's right-hand side. For a maximum string length of n, the table requires space $O(n^2 d|H_N|)$, exploiting the fact that β for terminal symbols is one in the bottom row and zero elsewhere. For a specific value of j, there are $O(d)$ possible k values greater than 1, each requiring time $O(|P_A|)$. For $k = 1$, the algorithm requires time $O(|P_D| j^{m-1} d^m)$, for the evaluation of all decomposition productions, as well as all possible combinations of substring lengths and levels of abstractions for each symbol on the right-hand side. Therefore, the whole algorithm would take time $O(n[d|P_A| + |P_D| n^{m-1} d^m]) = O(|P| n^m d^m)$.

As an alternative, we can modify the standard chart parsing algorithm (Younger 1967; Earley 1970) to compute the required values for β by recording the k values and probabilities associated with each edge. We would also ignore any distinctions among terminal symbols, since we are computing values over all possible terminal strings. Therefore, the time required for computing β is equivalent to that required for parsing a terminal string of length n, which is $O(n^3)$ ignoring the parameters of the grammar.

Network Generation Phase

Upon completion of the dynamic programming phase, we can use the table entries to compute the domains of random variables N_{ijk} and P_{ijk} and the required conditional probabilities. We begin at the top of the abstraction hierarchy for strings of the maximum length n starting at position 1. The corresponding symbol variable can be either E^1 or the special null symbol nil^*, indicating that the parse tree begins at some other point below. The prior probability of the start symbol is proportional to $\beta(E^1, n, k)$, while that of nil^* is proportional to the sum of all other β values for the start symbol E^1. The exact probabilities are normalized so that the sum equals one.

We start with this node and pass through all of the nodes in order of decreasing j and k. With each N node, we insert the possible productions into the domain of its corresponding P node. For a production rule r that maps E to $E_1 \cdots E_m$ with probability p and a breakdown of substring lengths and abstraction levels $(j_1, k_1), \ldots, (j_m, k_m)$, the conditional probability $\Pr(P_{ijk} = r\ \langle (j_1, k_1), \ldots, (j_m, k_m) \rangle\ |N_{ijk} = E) \propto p \cdot \prod \beta(E_t, j_t, k_t)$. The exact probability is normalized so that the sum over all rules r and breakdowns $\langle (j_t, k_t) \rangle$ for a particular left-hand symbol E is 1. For any symbol $E' \neq E$ in the domain of N_{ijk}, we can set the conditional probability $\Pr(P_{ijk} = r\ \langle (j_1, k_1), \ldots, (j_m, k_m) \rangle\ |N_{ijk} = E') = 0$.

A symbol variable which takes on the value nil has no children, so its production variable will also take on a null value (i.e., $\Pr(P_{ijk} = \mathsf{nil} | N_{ijk} = \mathsf{nil}) = 1$). For the special symbol nil^*, there are two possibilities, either the parse tree starts at the next level below, or it starts further down the tree. In the first case, the production is $\mathsf{nil}^* \rightarrow E^1$, and has a conditional probability proportional to the β value of E^1 at the j and k value immediately below the current position, given that $N_{ijk} = \mathsf{nil}^*$. In the second case, the production is $\mathsf{nil}^* \rightarrow \mathsf{nil}^*$, and has a conditional probability proportional to the sum of the β values of E^1 at all j and k more than one level below, given that $N_{ijk} = \mathsf{nil}^*$.

When all possible values for a production variable are added, we add a link from the corresponding node to the variables corresponding to each symbol on the right-hand side and insert these symbols into the domains of these child variables. A child nodes takes on the appropriate right-hand side symbol with probability 1 if the parent node has taken on the value of the given production. A child node takes on the value nil with probability 1 if none of its parent nodes assign it a symbol value.

Figure 5 illustrates the network structure resulting from applying this algorithm to the table of Figure 4 with a length bound of 4. In general, the resulting network has $O(n^2 d)$ nodes. The n N_{i11} variables have $O(|H_T|)$ states each, while the $O(n^2 d)$ other N variables have $O(|H_N|)$ possible states. The P_{ijk} variables for $k > 1$ (of which there are $O(n^2 d)$) have a domain of $O(|P_A|)$ states. For P_{ij1} variables, there are states for each possible decomposition production, for each possible combination of substring lengths, and for each possible level of abstraction of the symbols on the right-hand side. Therefore, the P_{ij1} variables (of which

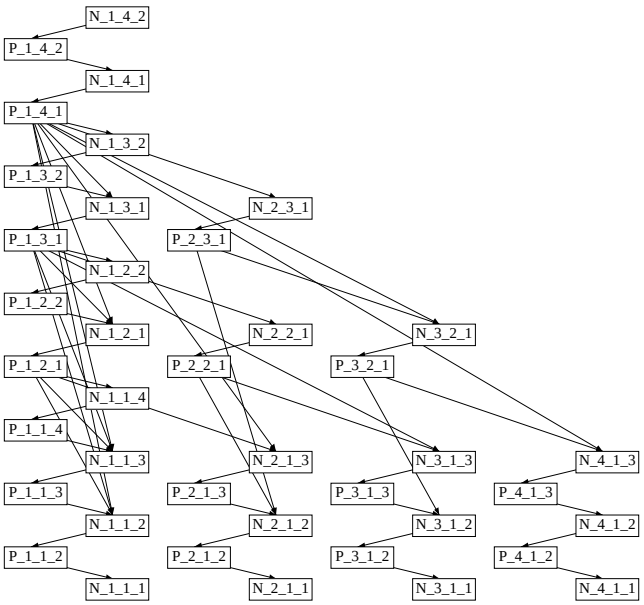

Figure 5: Network from example grammar.

there are $O(n^2)$) have a domain of $O(|P_D|j^{m-1}d^m)$ states.

Unfortunately, even though each particular P variable has only the corresponding N variable as its parent, a given N_{ijk} variable could have potentially $O(i \cdot (n - i - j))$ P variables as parents, and the size of a node's conditional probability table is exponential in the number of its parents. If we define T to be the maximum number of entries of any conditional probability table in the network, then the total time complexity of the algorithm is then $O(n^2 d|P_A|T + n^2|P_D|n^{m-1}d^m T^m + ndT + n^2 dT) = O(|P|n^{m+1}d^m T^m)$, which dwarfs the complexity of the dynamic programming algorithm for the β function. However, this network is created only once for a particular grammar and length bound.

Inference

The Bayesian network can answer any of the queries addressed by the usual parsing algorithm. To find the probability of a particular terminal string $x_1 \cdots x_L$, we can instantiate the variables N_{i11} to be x_i, for $i \leq L$, and nil, for $i > L$. Then, we can use any of the standard Bayesian network propagation algorithms to compute the probability of this evidence. The result is the conditional probability of the sentence, given that the string is bounded in length by n. We can easily acquire the unconditional probability, since the probability of a string having length no more than n is the sum of the β values for E^1 over all lengths of n and under. To find the most probable parse tree, we would use the standard network algorithms for finding the most probable configuration of the network.

The network represents a distribution over strings of bounded length, so we cannot obtain the same probability of an initial substring $x_1 x_2 \cdots x_L$ as (Jelinek, Lafferty, & Mercer 1992), which considered all completion lengths. However, we can find initial substring probabilities over completions of length bounded by $n - L$. The algorithm is identical to that for the probability of the entire sentence, except that we do not instantiate the N_{i11} variables beyond $i = L$ to be nil.

The procedure for finding the probability that a particular symbol derives a particular substring is complicated by the fact that there are multiple levels of abstraction possible for a particular substring. Therefore, after we instantiate the evidence, we must query all of the N variables for the particular i and j values of interest. We can start with the a particular k value and find the posterior probability of N_{ijk} being the symbol of interest. Having "counted" this event, we set the likelihood of N_{ijk} being that symbol to be zero, and proceed to a different k. We maintain a running total as we proceed, with the final probability being the result when all of the nodes have been counted.

In general, we can answer any query about an event that can be expressed in terms of the basic N and P random variables. Obviously, if we are interested in whether a symbol appeared at a particular i, j, k location in the parse tree, we only need to examine the marginal probability distribution of the corresponding N variable. Alternatively, we can find the probability of a particular symbol deriving *any part* of a particular substring (specified by i and j indices) by performing a similar procedure to that for an exact substring described above. However, in this case, we would continue computing posterior probabilities over *all* i and j variables within the bounds.

As another example, consider the case of possible four-word sentences beginning with the phrase Swat flies. In the network of Figure 5, we instantiate N_{111} to be swat and N_{211} to be flies and then propagate this evidence. We then need only to examine the joint distributions of N_{311} and N_{411} to find that like flies is the most likely completion. This is similar to the Left-to-Right Inside algorithm of (Jelinek, Lafferty, & Mercer 1992), except that we can find the most probable joint configuration over multiple time steps, instead of over only the one immediately subsequent.

A greater advantage is in the utilization of evidence. Any of the queries mentioned previously can be conditioned on any event that can be expressed in terms of N and P variables. If we only have a partial observation of the string, we simply instantiate the N_{i11} variables corresponding to the positions of whatever observations we have, and then propagate to find whatever posterior probability we require. In addition, we can exploit additional evidence about nonterminals within the parse tree. For instance, we may want to find the probability of the sentence Swat flies like ants with the additional stipulation that like ants is a prepositional phrase. In this case, we instantiate the N_{i11} variables as usual, but we also instantiate N_{321} to be pp.

Conclusion

The algorithms presented here automatically generate a Bayesian network representing the distribution over all parses of strings (bounded in length by some parameter) in the language of a PCFG. The first stage uses a dynamic programming approach similar to that of standard parsing algorithms, while the second stage generates the network, using the results of the first stage to specify the probabilities. This network is generated only once for a particular PCFG and length bound. Once created, we can use this network to answer a variety of queries about possible strings and parse trees. In general, we can use the standard inference algorithms to compute the conditional probability or most probable configuration of any collection of our basic random variables, given any other event which can be expressed in terms of these variables.

These algorithms have been implemented and tested on a number of grammars, with the results verified against those of existing dynamic programming algorithms when applicable, and against enumeration algorithms when given nonstandard queries. When answering standard queries, the time requirements for network inference were comparable to those for the dynamic programming techniques. Our network inference methods achieved similar response times for some other types of queries, providing a vast improvement over the much slower brute force algorithms.

The network representation of the probability distribution also allows possible relaxations of the independence assumptions of the PCFG framework. We could extend the context-sensitivity of these probabilities within our network formalism by adjusting the probability tables associated with our production nodes. For instance, we may make the conditional probabilities a function of the (i, j, k) index values. Alternatively, we may introduce additional dependencies on other nodes in the network, or perhaps on features beyond the parse tree itself. The context-sensitivity of (Charniak & Carroll 1994), which conditions the production probabilities on the parent of the left-hand side symbol, would require only an additional link from N nodes to their potential children P nodes. Other external influences could include explicit context representation in natural language problems or influences of the current world state in planning, as required by many plan recognition problems (Pynadath & Wellman 1995).

Therefore, even though the evidence propagation is exponential in the worst case, our method incurs this cost in the service of greatly increased generality. Our hope is that the enhanced scope will make PCFGs a useful model for plan recognition and other domains that require more flexibility in query forms and in probabilistic structure. In addition, these algorithms may extend the usefulness of PCFGs in natural language processing and other pattern recognition domains where they have already been successful.

Acknowledgments We are grateful to the anonymous reviewers for careful reading and helpful suggestions. This work was supported in part by Grant F49620-94-1-0027 from the Air Force Office of Scientific Research.

References

Briscoe, T., and Carroll, J. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics* 19(1):25–59.

Charniak, E., and Carroll, G. 1994. Context-sensitive statistics for improved grammatical language models. In *Proceedings of the National Conference on AI*, 728–733.

Charniak, E. 1993. *Statistical Language Learning*. Cambridge, MA: MIT Press.

Chou, P. 1989. Recognition of equations using a two-dimensional stochastic context-free grammar. In *Proceedings SPIE, Visual Communications and Image Processing IV*, 852–863.

Earley, J. 1970. An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery* 13(2):94–102.

Gonzalez, R. C., and Thomason, M. S. 1978. *Syntactic pattern recognition: An introduction*. Reading, MA: Addison-Wesley Publishing Company. 177–215.

Jelinek, F.; Lafferty, J. D.; and Mercer, R. 1992. Basic methods of probabilistic context free grammars. In Laface, P., and DeMori, R., eds., *Speech Recognition and Understanding*. Berlin: Springer. 345–360.

Ney, H. 1992. Stochastic grammars and pattern recognition. In Laface, P., and DeMori, R., eds., *Speech Recognition and Understanding*. Berlin: Springer. 319–344.

Pearl, J. 1987. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

Pynadath, D. V., and Wellman, M. P. 1995. Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Conference on Uncertainty in AI*, 472–481.

Sakakibara, Y.; Brown, M.; Underwood, R. C.; Mian, I. S.; and Haussler, D. 1995. Stochastic context-free grammars for modeling RNA. In *Proceedings of the 27th Hawaii International Conference on System Sciences*, 284–293.

Vilain, M. 1990. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proceedings of the National Conference on AI*, 190–197.

Wetherell, C. S. 1980. Probabilistic languages: a review and some open questions. *Comp. Surveys* 12(4):361–379.

Younger, D. 1967. Recognition and parsing of context-free languages in time n^3. *Info. and Control* 10(2):189–208.