Calculating the Probability of a Partial Parse of a Sentence

Fred Kochman and Joseph Kupin

Center for Communications Research
Institute for Defense Analyses
Princeton, New Jersey 08540

ABSTRACT

A standard problem in parsing algorithms is the organization of branched searches to deal with ambiguous sentences. We discuss shift-reduce parsing of stochastic context-free grammars and show how to construct a probabilistic score for ranking competing parse hypotheses. The score we use is the likelihood that the collection of subtrees can be completed into a full parse tree by means of the steps the parser is constrained to follow.

INTRODUCTION

Stochastic context-free grammars have been suggested for a role in speech-recognition algorithms, e.g. [1, 4, 9]. In order to be fully effective as an adjunct to speech recognition, the power of the probability apparatus needs to be applied to the problem of controlling the branched search for parses of ambiguous input.

The method we suggest for doing this employs shift-reduce (LR) parsing of context-free grammars together with a probability based score for ranking competing parse hypotheses. Shift-reduce parsers can be made very efficient for unambiguous grammars (and unambiguous inputs) and Tomita [7] shows how much of this efficiency can be maintained in the face of ambiguity. This makes this class of parsers a good candidate for many speech problems. The structural simplicity of shift-reduce parsers makes the analysis of the interaction of the parser with the stochastic properties of the language particularly clean.

The score we calculate is the likelihood that the collection of subtrees constructed by the parser so far can be completed into a full parse tree by means of the steps that the parser is constrained to follow, taking into account all possibilities for the unscanned part of the input. This score is the same as that suggested by Wright [9], who also studied shift-reduce parsers. We provide an exact method for calculating the desired quantity, while Wright's calculation requires several approximations.

Why do we care about this particular quantity? As a first rough answer, note that when this quantity is zero, then the hypothesis should be abandoned; there is no possibility that the parse tree can be completed. Furthermore, the bigger this quantity is, the larger the mass of the probability space that can be explored by pursuing that particular hypothesis.

For a more detailed answer, consider a breadth first search of candidate hypotheses. For each one we would like to know which is the correct one, given the grammar and the text segment we have observed: $a_1, \ldots, a_t$. We would like to calculate $P(H|a_1, \ldots, a_t)$.

This quantity is equal to

$$P(H\&a_1, \ldots, a_t)/P(a_1, \ldots, a_t).$$

The denominator in the above expression $P(a_1, \ldots, a_t)$ is the grand probability of seeing the observations $a_1, \ldots, a_t$ given the grammar. This is some fixed quantity. We might not know what it is, but as long as we are only comparing hypotheses that all explain the same string $a_1, \ldots, a_t$, this quantity is a scaling factor that can safely be ignored. The numerator is the quantity we intend to calculate.

For a depth-first or best-first search, as employed by [1], the quantity $P(a_1, \ldots, a_t)$ cannot be ignored. This makes the depth-first approach significantly more complicated.

In the rest of this paper we will restrict our attention to grammars in Chomsky-normal form. A similar probability analysis can be made for arbitrary context-free grammars, but the notation becomes cumbersome and the formulae more complicated. We note that all the topics in this paper are treated in considerably more detail, including proofs, in [3].

SHIFT-REDUCE PARSING

A ***bottom-up*** parser is one which reconstructs parsing trees by first constructing parsing subtrees over short disjoint segments of the text, then linking these together into a smaller number of larger trees, and so on recursively until a single parse tree emerges, covering the entire text. In this section we study a particular class of bottom-up parsers, called ***shift-reduce*** parsers, which conform to the following rules, leading to the reconstruction of a right-most-first derivation of the sentence being parsed.

The parser receives symbols one at a time from left to right and at each stage of the process, the parser's memory contains a sequence of disjoint parsing subtrees which completely cover the current input. Roughly speaking, as each new symbol is accepted (or ***shifted-in***) the parser decides how to incorporate it into a subtree and perhaps how to link several existing subtrees together (i.e. ***reduce***). The sequence of subtrees in the parser's memory at a given instant is called a ***parse hypothesis***, or a ***parser stack***.

To be more precise, here is how a shift-reduce parser updates the current hypothesis into a new one. Consider a parse hypothesis consisting of n subtrees: $\tau_1 \ldots \tau_n$, having root symbols $B_1 \ldots B_n$, respectively.

The three possible "moves" for reacting to the next input symbol 'x' are listed below.

1. 'x' can be shifted in and declared to be τ_{n+1}.
2. If there is a rule $A \rightarrow B_n$ in the grammar, then τ_n can be replaced by a new τ_n having A as a root and old τ_n as the left child of A. (Note that 'x' has not yet been shifted in.)
3. If there is a rule $A \rightarrow B_{n-1}B_n$ in the grammar, then τ_{n-1} and τ_n can be removed from the hypothesis and a new subtree τ_{n-1} is added, having A as a root and old τ_{n-1} as the left child of A and old τ_n as the right child of A. Again note that x remains to be shifted in.

The "input cycle" of a shift-reduce parser is typically to shift in a new symbol via move 1, use move 2 to give that symbol a nonterminal root, and then to perform some number of moves of type three. Choosing which (if any) of the allowable type-two and type-three rules should be used next in the parse can be quite difficult, but doing so cleverly makes the difference between efficient and inefficient parsing algorithms. When faced with a choice among possible moves some parsers make a breadth-first search among the possibilities. Others use a depth-first scheme, or even something intermediate between these two extremes. We will not be concerned with such schemes here. We concern ourselves only with a probabilistic score for the plausibility of available choices. (The best use of that score is a study in its own right.)

The important fact about shift-reduce parsers from our point of view is that they are quite limited in the kind of superstructure they can build above a given set of subtrees. Since new parent nodes can only be generated over the final few subtrees in the hypothesis, one can not "go back" and make non-final pairs of subtrees into siblings. (A precise result is proved in [3]). Figure 1 shows the necessary superstructure for an n-subtree hypothesis.

In this figure, the ellipses represent sequences of zero or more nodes in which each node is the left child of its parent. The diagram is also meant to admit the possibility that A_i is the same node as C_{i-1}. The right children of the nodes labeled C (labeled X) as well as those in the ellipses are all to be found in the remaining input.

THE LEFT-EDGE PROCESS

In order to calculate the sum of the probabilities of all complete parse trees that could result from the parser's further processing of a given hypothesis, we must sum across all the possibilities for the As and the Cs in figure 1 (which is a finite set) as well as summing over the potentially infinite set of sequences of nodes that could be lurking behind the ellipses. This sounds prohibitive, but we are saved by the fact that the sequences of nodes along the left-edge of a tree can be analyzed as the output of a Markov process. This fact is implicit in the work on trees and regular sets by [6], and was discovered independently by [2].

Happily, this observation leads to a closed form solution to the problem of calculating all the necessary probabilities for the infinite set of sequences. To begin, construct the matrix M which is the transition matrix of a Markov chain in which nonterminal symbols are transient states and terminal symbols are absorbing states.

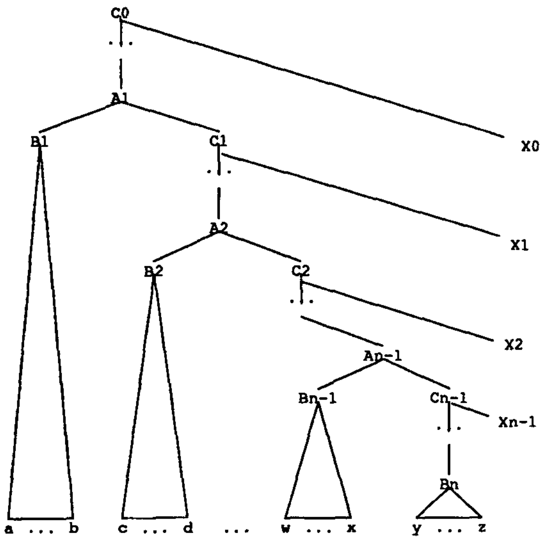

Figure 1: A parse hypothesis, with implied superstructure

$$\begin{aligned} M(A,B) &= \sum_{C} P(A \rightarrow BC) \quad \text{for nonterminals } B \\ M(A,b) &= P(A \rightarrow b) \qquad \text{for terminals } b \end{aligned}$$

Rows of M indexed by terminal symbols are identically zero.

To illustrate the use of Markov chain theory for the left-edges of trees, we compute the probability of the event that the left edge of a randomly generated subtree terminates in a specified terminal symbol a, given that the root is a specified nonterminal symbol A. This event is the disjoint union of the events that a is the n^{th} symbol in the left-edge sequence, for all $n \geq 1$. Correspondingly, we want the sum

$$\sum_{n \geq 1} \text{P(the } n^{th} \text{ left-edge symbol is a} \mid \text{the root is } A)$$

which is the sum of the $(A,a)^{\text{th}}$ entries in the sequence M, M^2, M^3, etc., which is in turn the $(A,a)^{\text{th}}$ entry in $M+M^2+M^3+\cdots$. As it turns out, this matrix sum converges. The sum is equal to $M(I-M)^{-1}$. Thus the number we seek is the $(A,a)^{\text{th}}$ entry of $M(I-M)^{-1}$. Note our convention that the root is the 0^{th} symbol along the left edge of the tree.

As another illustration we compute the probability that the left edge of a subtree T terminates in some specific subtree τ, again given that the root of T is A. More precisely, we compute the conditional probability that the subtree τ appears as a subtree of T, with its root B somewhere in the left-edge of T, given that the root of T is A. This is the disjoint union of the events, as n varies, that B is the n^{th} symbol in the left edge of T and that τ then appears rooted at this B.

If (just for a moment) we exclude the possibility that τ is identical to T, then n must be at least 1. For each $n \geq 1$, the conditional probability that τ appears rooted at the n^{th} symbol B, is $P(\tau|B)$ multiplied by the $(A,B)^{th}$ entry of the n^{th} power of M. In this case we can find, much as in the preceding illustration, that the sum from 1 to infinity of these pgobabilities is

$$P(\tau|B) \times \text{ the } (A,B)^{th} \text{ entry of } M(I-M)^{-1}.$$

To include the possibility that τ is identical to T, then we must add the term:

$$P(\tau|B) \times P(A=B).$$

Since the second factor is one or zero depending on whether $B = A$, the sum of probabilities for all $n \geq 0$ is

$$P(\tau|B) \times \text{ the } (A,B)^{th} \text{ entry of } \left[I + M(I-M)^{-1}\right]$$

which simplifies to:

$$P(\tau|B) \times \text{ the } (A,B)^{th} \text{ entry of } (I-M)^{-1}. \qquad (1)$$

In order to calculate the probability of the set of parse trees which might complete a given parse hypothesis we will need formulas like these, but with the proviso that we need to specify the rule that is used to generate the root of τ from its parent.

So to calculate all the probabilities that could ever arise due to the ellipses, we have work of inverting a rather large, but rather sparse, matrix. This work is done when the rule probabilities are decided upon, and before any sentences are parsed. The size of the matrix depends on the number of symbols (terminals and nonterminals) in the grammar.

THE PROBABILITY CALCULATION

The probability calculation must be divided into two cases. In one case we are in the midst of processing input and do not know how many input symbols (if any) remain to be processed. The second situation is that we know that all input symbols have been processed. This second case is special because it implies that the only unknown events are which rules are to be used to link up the subtrees to the root. In this case, the summation down the left edges of subtrees is no longer needed.

When in the Midst of the Input

When there may be more input to be processed, the calculation of the probability of a parser hypothesis with only one subtree is exactly the equation (1) in which the start symbol of the grammar, S, takes the place of the symbol A in the formula.

For hypotheses with $n > 1$ subtrees we need to take the A and C nodes from figure 1 into account. To calculate the probability of a parser hypothesis with n subtrees $\tau_1 \ldots \tau_n$ with root nodes $B_1 \ldots B_n$, we keep track of what rule is used to generate each B_i. This defines the necessary relationships among the various A_i B_i and C_i in figure 1. To perform our calculation we need the following matrices:

$$\begin{aligned} Q(A,r) &= p && \text{if rule } r \text{ is } A \xrightarrow{p} BC \text{ for some } B,C \\ &= zero && \text{otherwise} \\ Z(r,C) &= 1 && \text{if rule } r \text{ is } A \xrightarrow{p} BC \text{ for some } A,B \\ &= 0 && \text{otherwise} \\ M_0 &= (I-M)^{-1} && \text{for } M \text{ as defined above} \end{aligned}$$

The probability calculation requires the following four steps:

1. Compute $V_1^* =$ the S^{th} row of the product M_0Q. Zero out all entries except those corresponding to rules which have B_1 as a left child and call the result V_1.
2. For $i = 2,\ldots,n$ compute the product $V_i^* = V_{i-1}ZM_0Q$. Zero out all entries except those corresponding to rules which have B_i as a left child and call the result V_i.
3. Construct a final vector V_{fin} by zeroing out all entries of V_{n-1} except those corresponding to rules which have B_n as a right child.
4. The desired probability is the sum of the entries in V_n and V_{fin} multiplied by the conditional probability of the subtrees already constructed:

$$\prod_{i=1}^{n} P(\tau_i|B_i)$$

When at the End of the Input

If there is no more input and the hypothesis has only one subtree, then either the root of the subtree is the start symbol of the grammar, and hence the hypothesis has yielded a well-formed sentence with probability $P(\tau|S)$ or the hypothesis must be abandoned since it has not yielded a sentence and no further changes to it are possible.

Things are more interesting if the hypothesis contains more than one subtree. Consider a parser hypothesis H, consisting of $n > 1$ subtrees τ_1 through τ_n with root symbols B_1 through B_n respectively, with all of B_1 through B_n being nonterminal symbols. Suppose that the leaves of these subtrees exhaust the input, so no further shift operations are possible for the parser.

For each nonterminal B let M_B be the $\{\text{symbols}\} \times \{\text{symbols}\}$ matrix whose AC^{th} entry is the probability $P(A \to BC)$, if $A \to BC$ is a rule of the grammar while otherwise the AC^{th} entry is zero. Also, for each pair of nonterminals BC, let F_{BC} be the column vector indexed by nonterminals whose A^{th} entry is $P(A \to BC)$ if $A \to BC$ is a rule of the grammar; otherwise the A^{th} entry is zero. Let V_S be a row vector indexed by nonterminals with a 1 in the entry for S and zeros elsewhere.

Then, for $n > 1$, the probability of the hypothesis is equal to

$$V_S M_{B_1} M_{B_2} \ldots M_{B_{n-2}} F_{B_{n-1}B_n} \times \prod_{i=1}^{n} P(\tau_i|B_i)$$

Programming Considerations

There are several problems in making a practical parser based on the probabilities calculated above. First we must invert the rather large matrix $I - M$ and then for each parse hypothesis we must perform two or three matrix operations for each subtree of the hypothesis. This is not actually as bad as it seems.

First note that we can absorb two matrix operations for each subtree into one operation by precomputing M_0Q. If we use this as our "in-core" matrix, we can reproduce M_0 when needed (for $n = 1$ computations) by summing across the relevant rules.

Next we note that the vector by which we are premultiplying is very sparse. This is true since the preceding step was to zero-out all entries in the vector that have the "wrong" left child. This means that there are only a few rows of the big M_0Q matrix that concern us.

Also note that immediately after we calculate the vector result, we will again zero out entries with the "wrong" left child. This means that we really only need calculate those few entries in the result vector that have the desired left child. This reduces the matrix operation to much lower order, say 5×5. The size of the calculation is determined by how many rules have a given nonterminal as left child. A grammar will be easy to parse with this method if each nonterminal only appears as the left child in a few rules.

Finally, we note that each parse step can only create one new subtree, and that at the end of the hypothesis. So, if we remember the vector associated with each subtree as we make it, we only need to do one of these order 5×5 calculations to get the probability of the new hypothesis.

BUILDING A PARSER

One might consider implementing the above probability calculation in conjunction with some conventional shift-reduce parser. In this case one would let the LR0 parser suggest possibilities for updating a given parse hypothesis and use the above scheme to compute probabilities and discard unpromising hypotheses.

It is worth pointing out that all the information needed for LR0 parsing can in fact be reproduced from the probability vectors we calculate. Hence we do not really need to construct such a parser at all! The point is that starting from a particular hypothesis, a given proposal for a next move leads to a nonzero probability if and only if there is some completion of the input that would not "crash" for the conventional parser. The vectors V_n and V_{fin} contain all the information we could desire about the next step for the parser.

Finally, let us remark that our matrix calculations can be adapted to yield a shift-reduce parser even when no probabilities are initially present. We simply replace the transition matrix M with a suitably scaled incidence matrix M', in which $M'(A, B) = \varepsilon$ if B is the left child of A via *some* rule. Otherwise $M'(A, B) = 0$. A similar replacement is made for the matrix Q. The specific values of the "probabilities" then arising from our calculations do not matter, only whether or not they are zero. Thus, the off-line construction of parser tables could be accomplished via a matrix inversion, rather than the conventional recursive calculations.

CONCLUSIONS

With the addition of this score, there are now a number of different methods for controlling the parsing of sentences from a stochastic grammar, each with its own kind of parser and expected form of the grammar. The four we know of are: [1, 5, 8, 9]. It is possible to find "expensive" grammars for each of these scores. For our score, a "cheap" grammar is one in which each symbol is the left child in relatively few rules.

The goal, then, must be to find a parser, score and grammar that meet the needs of a particular application. We take at least some small comfort from the fact that our score has a Bayesian "maximum likelihood" interpretation, even though the superiority of that approach depends on the shaky assumption that the input being parsed really is the randomly-generated output of the stochastic grammar under consideration.

REFERENCES

[1] Chitrao, M. V. and R. Grishman., "Statistical Parsing of Messages," *Proc. DARPA Speech and Natural Language Workshop*, pp. 263–266, June 1990.

[2] Jelinek, F., "Computation of the Probability of Initial Substring Generation by Stochastic Context Free Grammars", *Internal Report*, Continuous Speech Recognition Group, IBM Research, T.J. Watson Research Center, Yorktown Heights, NY 10598, 10 pages.

[3] Kochman, F. and J. Kupin, "Sequential Processing of Input Using Stochastic Grammars" to appear.

[4] Lari, K and S. J. Young., "The Estimation of Stochastic Context-free Grammars Using the Inside-Outside Algorithm," *Computer Speech and Language* vol. 4, pp. 35-56, 1990

[5] Lee, H. C. and K. S. Fu., "A Stochastic Syntax Analysis Procedure and its Application to Pattern Classification," *IEEE Trans.* Vol. C-21, pp. 660–666, July 1972.

[6] Thatcher, J. W., "Characterizing Derivation Trees of Context Free Grammars through a Generalization of Finite Automata Theory" *Journal of Computer and System Sciences*, Vol1.4 Dec. 1967.

[7] Tomita, M., *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, Boston, 1986.

[8] Velasco, F. R. and C. R. Souza, "Sequential Syntactic Decoding," *Int. J. Comput. Inform. Sci.* Vol. 3.4, pp. 273–287, 1974.

[9] Wright, J. H., "LR parsing of Probabilistic Grammars with Input Uncertainty for Speech Recognition." *Computer Speech and Language* Vol. 4, pp. 297–323, 1990.