Abstract: The parsing of speech can be viewed as the parsing of probabilistic input. The problem is formally stated and two algorithms are presented for efficient probabilistic parsing. A proof that these algorithms discover the most likely parse is given. The advantages of this algorithm over the conventional backtracking and ad hoc methods are also outlined.

On the Optimal Parsing of Speech

Richard J. Lipton and Lawrence Snyder

Research Report #37

October 1974

1. Introduction

Recently there has been considerable interest in the understanding of speech [1-5]. Although several approaches are under consideration, each approach has the follwing basic structure (see figure 1): First an acoustic process converts the stream of speech to a sequence of distributions $p_1, \ldots, p_n$. The distribution p_i assigns to each possible "word" the "likelihood" that this word occurs in the i^{th} position. Here "word" can be an English word as in Levinson's recognizer of discrete speech [4] or it can be a phoneme as in [5]. The "likelihood" is a real number that represents the confidence the acoustic process has in the belief that a given word occurs in the given position.

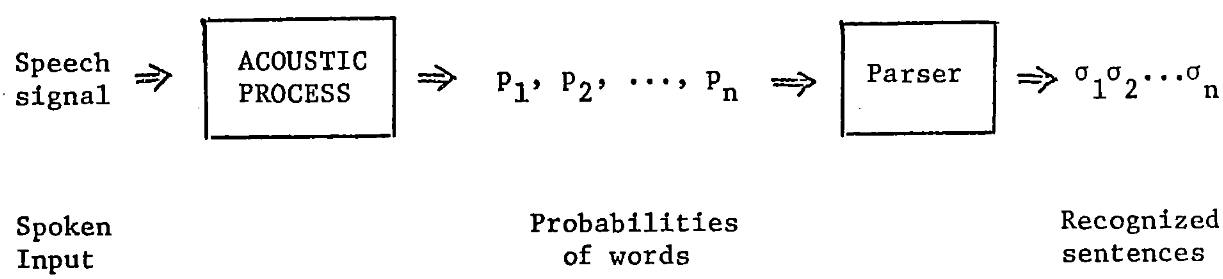

Figure 1. Speech Recognition Schema.

No assumption is made on whether or not the distributions $p_1, \ldots, p_n$ must actually be probability distributions, i.e.

$$\sum_{\text{all words } w} p_i(w)$$

need not equal 1. The second part of the current approaches to speech understanding attempt to "parse" the sequence of distributions $p_1, \ldots, p_n$. The speech that is being analyzed comes from some language L. For example, Hearsay uses the language of possible chess moves [3], BBN uses a query language [2], while Levinson uses a simple Fortran-like programming language [4]. Parsing the sequence of distributions means finding a sequence of words $\sigma_1\sigma_1\cdots\sigma_n$ such that

(1) $\sigma_1\sigma_2\cdots\sigma_n \in L$ and

(2) $p_1(\sigma_1) \ldots p_n(\sigma_n)$ is maximum.

Informally, parsing as it is used here means finding the most likely sentence in the language L. Thus the basic assumption is:

The actual sentence (input to the acoustic process) is the most likely sentence in L with respect to $p_1, \ldots, p_n$.

Of the two phases of the current speech understanding systems - creation of the distributions and parsing of those distributions - the latter is the subject of this report.

Precisely, the problem is:

Given A language $L \subseteq \Sigma^*$ and a sequence $p_1, \ldots, p_n$ of distributions where $p_i: \Sigma \to [0,1]$;†

---

† Σ^* is the set of all strings over the alphabet Σ; $[0,1]$ is the set of real numbers from 0 to 1.

Find A sequence $\sigma_1 \ldots \sigma_n \in L$ such that $p_1(\sigma_1) \ldots p_n(\sigma_n)$ is maximum.

Call this problem the Probabilistic Parsing Problem (PPP): essentially it is the classic parsing problem except that the input is a "fuzzy" input in the sense of [10]. To be accurate, of course, the parsing problem should produce a derivation for the $\sigma_1 \ldots \sigma_n$ with respect to a particular grammar for L. No confusion should arise, so we retain this usage which is consistent with the speech understanding literature.

A popular method currently used to solve the PPP is based on backtracking [5]. Suppose that $p_1, \ldots, p_n$ is a sequence of distributions. Order all strings in Σ^n (i.e. all strings of exactly length n) as follows:

$$x_1 \ldots x_n \circledcirc{\leq} y_1 \ldots y_n$$

provided there is a k such that $x_1 \ldots x_k = y_1 \ldots y_k$ and $p_{k+1}(x_{k+1}) \leq p_{k+1}(y_{k+1})$. Ties are possible, of course, since the distribution is not required to make unique assignments to symbols. Whether ties are ordered by some rule or taken arbitrarily makes little difference and does not affect the argument given below. Thus $x_1 \ldots x_n \circledcirc{\leq} y_1 \ldots y_n$ provided the first place, say k, that two sentences differ (going left to right) has x_{k+1} less likely than y_{k+1} with respect to the distribution p_{k+1}. The ordering $\circledcirc{\leq}$ is easily seen to be a linear ordering; hence, the strings of Σ^n are linearly ordered. The backtracking method is to search this linear order $\alpha_1 (\leq) \cdots (\leq) \alpha_m$ starting from α_m and stopping at the first α_i which is in the given language L. Although there are many embellishments to this method, the fact remains that the search is the fundamental feature.

The disadvantages of this method are, basically, two. First, backtracking is a potentially slow algorithm, i.e. in worst case the backtracking algorithm can take exponential time on the order $|\Sigma|^n$ (exponential in n). While this fact is well known to other users of backtracking type algorithms [11], it is not often acknowledged in the speech recognition area. The problem is that a small increase in the complexity of the speech recognition task (i.e. increase length of sentences, increase input set's size and so on) may lead to a huge increase in the running time of the backtracking algorithm.

The second disadvantage is even more important. The backtracking algorithm does not solve the PPP. Not only is it possibly slow but it is incorrect. A simple example should suffice to demonstrate this assertion. Let Σ, the input alphabet, be $\{a,b\}$, let the language $L = \{ab,ba\}$, and let $p_1 p_2$ be

$$p_1(a) = 1, \quad p_1(b) = .3, \quad p_2(a) = 1, \quad p_2(b) = .01.$$

The ordering defined for the backtracking is

$$bb \ (\leq)\ ba \ (\leq)\ ab \ (\leq)\ aa$$

with the search proceeding from right to left. The backtracking algorithm would stop after finding that aa $\notin$ L and ab $\in$ L. The likelihood of ab is $p_1(a)p_2(b) = .01$. However, ba $\in$ L and the likelihood of ba is $p_1(b)p_2(a) = .3$. Thus the backtracking method has failed to find the most likely sentence. Since the basic assumption was that the most likely sentence was the one input to the speech system this failure is very damaging. Obviously, searching the list in reverse order won't improve performance, generally. It should also be noted that this difficulty is independent of how complex the language is that is being recognized. In an effort, presumably, to ameliorate these difficulties, heuristics have been added which in effect reorder the list to improve the likelihood of a correct solution. A disadvantage of these heuristics is they are ad hoc and they must be tailored carefully to language L. This makes it almost impossible to have a general speech understanding system since the heuristics of one language probably won't be applicable to the next. Also, since the heuristics are only "frequently" helpful (as opposed to "always" helpful) there is no influence on worst case execution time. Moreover, it is virtually impossible to predict average behavior on new data.

In contrast to the current attempts to solve the PPP we will present an algorithm that is efficient and correct, i.e. the algorithm will always find a most likely sentence. More exactly we will show that if L is a context free language [6] with q productions, then our algorithm finds a most likely sentence corresponding to the distributions $p_1, \ldots, p_n$ in $O(qn^3)$ time. This algorithm is a generalization of Younger's n^3 parser for context free languages. In addition we will show that if L is a regular language [6] with q productions, then there is an algorithm that solves the PPP with distributions $p_1, \ldots, p_n$ in $O(qn)$ time. The advantages of these algorithms are that they are efficient in worst case $O(qn^3)$ and $O(qn)$ vs. $O(|\Sigma|^n)$ and that they correctly solve the PPP while the backtracking techniques do not. Moreover our algorithms need no heuristics at all; therefore, one can imagine using them in a general (i.e. language independent) speech system.

Note that our algorithms work for languages that are context free or regular. This is not restrictive for two reasons. First, several of the current speech systems use languages that are regular, while in general the rest appear to be context free. For example, the chess language of Hearsay [3] can be written as a regular language with approximately 110 productions. Second, context free languages or even regular languages with many productions (i.e. with q large) can approximate, to a high degree, languages that are not context free [12]. Since the running times of our algorithms are all linear in q we can potentially handle languages with many productions, say $q \simeq 1000$. Thus, we can potentially handle languages which, although they are context free, have great power and diversity.

2. PPP for Regular Languages

In this section the PPP is studied for the case where the input language L is regular. As stated earlier many languages currently used in the speech understanding area are actually regular (e.g. Hearsay [3]); moreover, regular languages with many productions can closely approximate complex domains of discourse. Since the algorithm presented here requires $O(qn)$ time (where q = the number of productions in a grammar for L and n = the number of input distributions), our algorithm can handle large regular languages.

Let L be a regular language over the input alphabet Σ which is accepted by the nondeterministic finite state automaton $< Q, \Sigma, Q_0, s, \delta >$ where (see [6])

1. Q is the set of states,
2. Σ is the input alphabet,
3. Q_0 is the set of accepting states,
4. $s \in Q$ is the start state, and
5. $\delta \subseteq Q \times \Sigma \times Q$ is the set of transitions.

Also let q be the cardinality of the set δ. (Note, q corresponds to the number of productions in a grammar for L.)

Let $p_1, \ldots, p_n$ be the input distributions; hence, each p_i maps Σ to [0,1]. The algorithm we are about to define will operate on the data structures $\Phi_k(x)$ and $f_k(x)$ where $0 \leq k \leq n$ and $x \in Q$. The values of $\Phi_k(x)$ are real numbers, while the values of $f_k(x)$ are strings over Σ. Intuitively,

$\Phi_k(x)$ = likelihood that the acceptor is in state x after scanning the first k terms of the "probabilistic" input;

$f_k(x)$ = $\sigma_1 \ldots \sigma_k$ provided $\sigma_1 \ldots \sigma_k$ sends the acceptor into state x and $p_1(\sigma_1) \ldots p_k(\sigma_k) = \Phi_k(x)$.

The algorithm first initializes the data structures as follows:

$$\Phi_0(x) = \begin{cases} 1 & \text{if } x = s; \\ 0 & \text{otherwise.} \end{cases}$$

$$f_0(x) = \Lambda \quad \text{all } x$$

where Λ is the null string. Now for $x \in Q$ and $0 \leq k \leq n$, $\Phi_{k+1}(x)$ and $f_{k+1}(x)$ are calculated inductively for each state x as follows:

(*) Let $(y, \sigma, x) \in \delta$ be such that $\Phi_k(y)p_{k+1}(\sigma)$ is as large as possible.

If several σ's satisfy this condition we can, without loss of generality, make an arbitrary choice. Then

$$\Phi_{k+1}(x) = \Phi_k(y)p_{k+1}(\sigma)$$

and

$$f_{k+1}(x) = f_k(y)\sigma \quad .$$

The solution to the PPP is then determined as follows: Find a $\Phi_n(x)$ which is maximum where x is an accepting state. The answer to the PPP is then the string $f_n(x)$.

Next we establish in the following theorem that the above algorithm does indeed solve the PPP. Let the notation

$s \xrightarrow{w_1 w_2 \ldots w_k} x$ abbreviate "the symbol sequence $w_1, w_2, \ldots, w_k$ carries the nondeterministic finite automaton from state s to state x."

<u>Theorem</u> For all $k \geq 0$ and $x \in Q$,

$$\Phi_k(x) = \max_{s \xrightarrow{w_1 \ldots w_k} x} p_1(w_1) \ldots p_k(w_k) .$$

<u>Proof</u> We will prove by induction on k the stronger result:

(1a) $s \xrightarrow{\sigma_1 \ldots \sigma_k} x$ and $p_1(\sigma_1) \ldots p_k(\sigma_k) = \Phi_k(x)$

where $\sigma_1 \ldots \sigma_k = f_k(x)$

(1b) $$\Phi_k(x) = \max_{s \xrightarrow{w_1 \ldots w_k} x} p_1(w_1) \ldots p_k(w_k)$$

Clearly (1a) and (1b) are true for $k = 0$. [By convention an empty product is 1.] Now consider $k+1$ case. By the definition of the algorithm $f_{k+1}(x) = f_k(y)\sigma$ and $\Phi_{k+1}(x) = \Phi_k(y)p_{k+1}(\sigma)$ where

$$\Phi_k(y)p_{k+1}(\sigma)$$

is as large as possible such that $y \xrightarrow{\sigma} x$. By induction we know

(2a) $s \xrightarrow{\sigma_1 \ldots \sigma_k} y$ and $p_1(\sigma_1) \ldots p_k(\sigma_k) = \Phi_k(y)$

where $\sigma_1 \ldots \sigma_k = f_k(y)$

(2b) $$\Phi_k(y) = \max_{s \xrightarrow{w_1 \ldots w_k} y} p_1(w_1) \ldots p_k(w_k) .$$

Now $f_{k+1}(x) = \sigma_1 \ldots \sigma_k \sigma$, $s \xrightarrow{\sigma_1 \ldots \sigma_k} y \xrightarrow{\sigma} x$, and

$\Phi_{k+1}(x) = \Phi_k(y) p_{k+1}(\sigma) = p_1(\sigma_1) \ldots p_k(\sigma_k) p_{k+1}(\sigma)$; hence, (1a) is true. Next

$$\Phi_{k+1}(x) = \max_{y \xrightarrow{\sigma} x} \Phi_k(y) p_{k+1}(\sigma) .$$

Therefore, by induction (2b), we have

$$\Phi_{k+1}(x) = \max_{y \xrightarrow{\sigma} x} p_{k+1}(\sigma) \cdot \max_{s \xrightarrow{w_1 \ldots w_k} y} p_1(w_1) \ldots p_k(w_k)$$

$$= \max_{y \xrightarrow{\sigma} x} \max_{s \xrightarrow{w_1 \ldots w_k} y} p_1(w_1) \ldots p_k(w_k) p_{k+1}(\sigma)$$

$$= \max_{s \xrightarrow{w_1 \ldots w_k \sigma} x} p_1(w_1) \ldots p_k(w_k) p_{k+1}(\sigma) .$$

Thus, (1b) is true for $k+1$. □

It only remains to show that the running time of this algorithm, for some reasonable machine model, is $O(qn)$. It should be clear that the running time is dominated by how fast one can compute (*). Let q_x be the number of transitions of the form (y, σ, x), for some y and some σ, i.e. the in-degree of x in the state graph. Clearly,

$$\sum_{x \in Q} q_x = q .$$

The cost of computing (*) for a fixed x and fixed k is $O(q_x)$. Therefore, the total cost of computing (*) is

$$\sum_{k=0}^{n-1} \sum_{x \in Q} O(q_x)$$

which sums to $O(qn)$. It thus follows that the running of this algorithm is $O(qn)$. Using well known programming techniques, the space requirement is clearly $O(mn)$, for m states.

3. PPP for Context Free Languages

The last section studied the PPP for input languages that are regular. Here an algorithm is presented that solves the PPP when the input language is context free [6]. This algorithm is a straightforward generalization of Younger's parser [7,6] to handle "probabilistic" input. The algorithm presented here operates in $O(qn^3)$ time where q = the number of productions in a Chomsky Normal Form grammar for the input language and n = the number of input distributions.

Let L be a context free language over the input alphabet Σ which is generated by the Chomsky Normal Form grammar $G = < V, T, S, P >$ (see [6]) where

1. V is the set of <u>nonterminals</u>,
2. T is the set of <u>terminals</u>,
3. S is the <u>start</u> symbol,
4. P is the set of <u>productions</u>.

Let q be the size of the set P.

Let $p_1, \ldots, p_n$ be the input distributions. The algorithm we are about to define will operate on the data structures $\Phi_{ij}(x)$ and $f_{ij}(x)$ where $0 \le i \le j \le n$ and $x \in V$. The values of $\Phi_{ij}(x)$ are real numbers, while the values of $f_{ij}(x)$ are strings. Intuitively, $\Phi_{ij}(x)$ and $f_{ij}(x)$ behave as follows:

$\Phi_{ij}(x)$ = likelihood that $x \overset{*}{\Longrightarrow}$ the substring of the input from position i to position j;

$f_{ij}(x) = \sigma_i \ldots \sigma_j$ provided $x \overset{*}{\Longrightarrow} \sigma_i \ldots \sigma_j$ and $p_i(\sigma_i) \ldots p_j(\sigma_j) = \Phi_{ij}(x)$.

The algorithm will first initialize the data structures as follows:

$$\Phi_{ii}(x) = \max\{p_i(\sigma) \mid x \to \sigma \text{ is a production in } P\}$$

$$f_{ii}(x) = \sigma \text{ provided } x \to \sigma \text{ is in P and } \Phi_{ii}(x) = p_i(\sigma).$$

Now for all x in V and $0 \le i \le j \le n$, $\Phi_{ij}(x)$ and $f_{ij}(x)$ are calculated inductively as follows:

(*) Let $x \to yz$ be a production such that $\Phi_{ik}(y)\Phi_{k+1j}(z)$ is as large as possible.

There may be several values of y, z and k satisfying this requirement, but an arbitrary choice can be made without loss of generality. Then

$$\Phi_{ij}(x) = \Phi_{ik}(y)\Phi_{k+1j}(z)$$

and

$$f_{ij}(x) = f_{ik}(y)f_{k+1j}(z) .$$

The solution to the PPP is then obtained as follows: the string $f_{1n}(S)$ is the solution.

The proof that this algorithm indeed does solve the PPP is similar to the proof for the regular case and is omitted.

The running time of this algorithm is easily seen to be dominated by the step (*). Let q_x be the number of productions with x on the left hand side. Clearly,

$$\sum_{x \in V} q_x = q .$$

For a fixed i and j and a fixed $x \in V$ the cost of step (*) is at most $O(q_x n)$. The running time of the algorithm is therefore bounded above by

$$\sum_{0 \le i \le j \le n} \sum_{x \in V} O(q_x n)$$

which sums to at most $O(qn^3)$. Thus the running time of this algorithm is at most $O(qn^3)$.

4. Conclusions

The PPP has been solved efficiently for both regular and context free languages. While we have been mainly motivated by applications to the parsing of speech, the results presented here should have application in other areas of pattern recognition. For example, the recognition of signals such as EKG ([13]) should be expressable as a PPP. Note that the usual parsing task is a special case of the PPP, i.e. if $\sigma_1\sigma_2\ldots\sigma_n$ is the input then $p_i(\sigma_i) = 1$ and $p_i(\sigma_j) = 0$ for $i \neq j$. Thus any better solution to the PPP for context free grammars will provide an improved context free parsing algorithm.

It is also interesting to observe that the problem of "error correction in parsing" ([8]) is a special case of what we call the PPP. For suppose that $\sigma_1\ldots\sigma_n$ is an input that we wish to parse for some language L. Define distributions $p_1, \ldots, p_n$ by

$$p_i(x) = \begin{cases} 1, & \text{if } x = \sigma_i; \\ 1/2, & \text{otherwise.} \end{cases}$$

Observe that $p_1(x_1)\ldots p_n(x_n) = \frac{1}{2^k}$ if and only if the Hamming distance from $\sigma_1\ldots\sigma_n$ to $x_1\ldots x_n$ is k. Therefore, solving the PPP is equivalent to finding a sentence $x_1 \ldots x_n$ in the language that is as close to $\sigma_1 \ldots \sigma_n$ as possible, with respect to the Hamming distance. The problems of insertion and deletion do not seem to present any problem and are currently being studied.

Acknowledgement

The authors wish to thank Jerry Hobbs for a number of helpful comments.

References

[1] A. Newell et al.
Speech understanding systems: final report of a study group.
Carnegie-Mellon University, Pittsburgh, May 1971.

[2] W. A. Woods et al.
The lunar sciences natural language information system: final report.
BBN Report 2378, Bolt, Beranek and Newman, Cambridge, June 1972.

[3] D. R. Reddy et al.
The Hearsay speech understanding system: an example of the recognition process.
3rd International Joint Conference on Artificial Intelligence, Stanford Research Institute, August 1973.

[4] S. E. Levinson.
An Artificial Intelligence Approach to Automatic Speech Recognition.
PhD thesis, University of Rhode Island, 1974.

[5] D. E. Walker.
Speech understanding through syntactic and semantic analysis.
3rd International Joint Conference on Artificial Intelligence, Stanford Research Institute, August 1973.

[6] J. E. Hopcroft and J. D. Ullman.
Formal Languages and Their Relation to Automata.
Addison-Wesley, 1969.

[7] D. H. Younger.
Recognition and parsing of context-free languages in time n^3.
Information and Control 10(2), 1967.

[8] G. Lyon.
Syntax-directed least-errors analysis for context-free languages: a practical approach.
CACM 17(1), 1974.

[9] R. E. Bellman.
Dynamic Programming.
Princeton University Press, 1957.

[10] L. A. Zadeh.
Fuzzy sets.
Information and Control 8, 1965.

[11] M. Held and R. M. Karp.
A dynamic programming approach to sequencing problems.
J. SIAM 10, 1962.

[12] J. R. Hobbs
A Metalanguage for Expressing Grammatical Restrictions in Nodal Spans Parsing of Natural Language.
PhD thesis, New York University, 1974.

[13] G. C. Stockman, L. N. Kanal, and M. C. Kyle.
Design of a waveform parsing system.
University of Maryland Technical Report TR-266, October 1973.