

THE VOYAGER SPEECH UNDERSTANDING SYSTEM: A PROGRESS REPORT*

Victor Zue, James Glass, David Goodine, Hong Leung,
Michael Phillips, Joseph Polifroni, and Stephanie Seneff

Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

As part of the DARPA Spoken Language System program, we recently initiated an effort in spoken language understanding. A spoken language system addresses applications in which speech is used for interactive problem solving between a person and a computer. In these applications, not only must the system convert the speech signal into text, it must also understand the linguistic structure of a sentence in order to generate the correct response. This paper describes our early experience with the development of the MIT VOYAGER spoken language system.

INTRODUCTION

Recently, we have been directing our research effort towards spoken language understanding as part of the DARPA Spoken Language System program. The project is motivated by the belief that many of the applications suitable for human/machine interaction using speech typically involve interactive problem solving. That is, in addition to converting the speech signal to text, the computer must also *understand* the linguistic structure of a sentence in order to generate the correct response. We have focused our attention on three main issues. First, the system must integrate speech recognition with natural language in order to achieve speech understanding. Second, the system must have a realistic application domain, and be able to translate spoken input into appropriate actions. Finally, the system must begin to deal with spontaneous speech, since people do not always utter grammatically well-formed sentences during a spoken dialogue.

Over the past six months, we have constructed the skeleton of a spoken language system. The purpose of this paper is to describe the various components of this system. In related activities, we have collected a sizeable spontaneous speech database, and have used the data for analyses, system training and evaluation. The collection and analysis of the spontaneous speech database, and the preliminary evaluation of our spoken language system are described in two companion papers that appear elsewhere in these proceedings [1,2].

TASK DESCRIPTION

In order to explore issues related to a fully-interactive spoken language system, we have selected a task in which the system knows about the physical environment of a specific geographical area as well as certain objects inside this area, and can provide assistance on how to get from one location to another within this area. The system, which we call VOYAGER, currently focuses on the the city of Cambridge, Massachusetts, between MIT and Harvard University, as shown in Figure 1. It can answer a number of different types of questions about certain hotels, restaurants, hospitals, and other objects within this region. At the moment, VOYAGER has a vocabulary of 324 words. Within this limited domain of knowledge, it is our hope that

*This research was supported by DARPA under Contract N00014-89-J-1332, monitored through the Office of Naval Research.

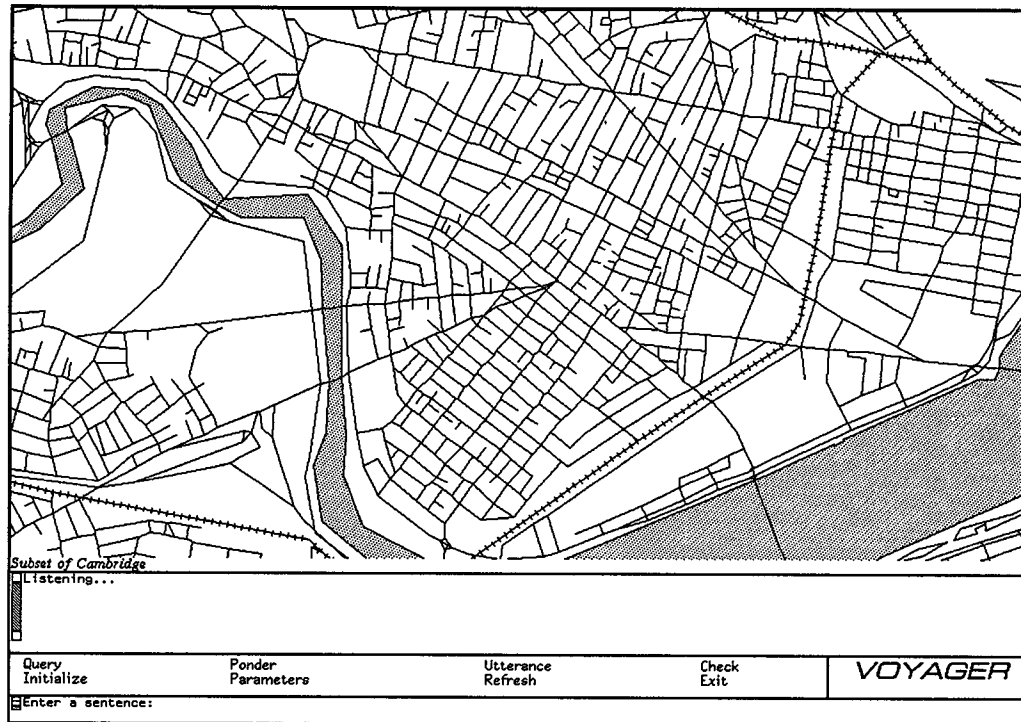


Figure 1: A display showing the geographical region known to the VOYAGER system.

VOYAGER will eventually be able to handle any reasonable query that a native speaker is likely to initiate. As time progresses, VOYAGER's knowledge base will undoubtedly grow.

SYSTEM DESCRIPTION

VOYAGER is made up of three components. The speech recognition component converts the speech signal into a set of word hypotheses. The natural language component then provides a linguistic interpretation of the set of words. The parse generated by the natural language component is subsequently transformed into a set of query functions, which are passed to the back-end for response generation. The back-end is an enhanced version of a direction assistance program developed by Jim Davis of MIT's Media Laboratory [3]. We will describe each component in sequence, paying particular attention to those parts that have not been previously reported.

SPEECH RECOGNITION COMPONENT

The first component of VOYAGER uses the SUMMIT speech recognition system developed in our group. SUMMIT places heavy emphasis on the extraction of phonetic information from the speech signal. It achieves speech recognition by explicitly detecting acoustic landmarks in the signal in order to facilitate acoustic-phonetic feature extraction. The system can be trained automatically, since it does not rely on extensive knowledge engineering. The design philosophy, implementation, and evaluation of the SUMMIT system have been described in detail previously [4]. As a result, we will only report in this paper modifications to the system since the last workshop. These include the development of a new module for lexical expansion via

phonological rules, and a new corrective training procedure.

Lexical Expansion

The original SUMMIT system used a phonological expansion capability provided to us by SRI [6]. Within the last year, however, we have decided to rewrite this part of the system in order to establish increased flexibility and speed. The new version, named MARBLE, offers several new properties. A canonic set of phonemes is represented by a set of default values for distinctive features. Specified allophonic information due to context dependencies can be represented in the particular instance of the phoneme generated in a word lattice. Thus, for instance, when a word-final /s/ and a word-initial /s/ merge, the resulting /s/ can be marked as [+geminate]. This information can then be incorporated into the scoring for the particular allophone. The allophonic slot can also be used to indicate place of articulation of adjacent consonants, for example, to facilitate the decoding of context-dependent models. The rule-writing process is also straightforward, and it is simple to keep track of the rule ordering. Finally, the time it takes to expand a lexicon has been reduced. We believe this new rule system will be a powerful tool for effectively representing context dependencies.

Corrective Training

The training of SUMMIT is performed iteratively, after being initialized on the TIMIT database [4,5]. For each iteration, the recognizer computes the *best* alignment between a path in the acoustic phonetic network and a path in the lexical network, i.e., the recognized output. The recognizer also computes a *forced* alignment using only the correct string of words. The system then trains the next iteration of phonetic models based on the matches between lexical arcs and phonetic segments in the forced alignments. The recognizer also adjusts lexical arc weights based on a comparison of the number of times the arc was used incorrectly (present in the best alignment but not in the forced alignment) to the number of times the arc was missed (present in the forced alignment but not in the best alignment). The goal of this corrective training procedure is to equalize the number of times an arc is missed and the number of times the arc is used in the wrong way. If sufficient training data is not available for a particular arc, then the weights are derived by collapsing it with other phonetically-similar arcs.

Presently, lexical decoding is accomplished by using the Viterbi algorithm to find the best path that matches the acoustic-phonetic network with the lexical network. Since the speech recognition and natural language components are not as yet fully integrated, we currently use a word-pair language model with a perplexity of 22 to constrain the search space.

NATURAL LANGUAGE COMPONENT

In the context of a spoken language system, the natural language component should perform two critical functions: 1) to provide constraint for the recognizer component, and 2) to provide an interpretation of the meaning of the sentence to the back end. Our natural language system, TINA, was specifically designed to meet these two needs. The basic design of TINA has been described elsewhere [7], and therefore will only be briefly mentioned here. Instead, we would like to focus on the issue of how to incorporate semantics into the parses. We have found that an enrichment of the parse tree with semantically loaded categories at the lower levels leads to both improved word predictions and a relatively straightforward interface with the back end.

General Description

The grammar is entered as a set of simple context-free rules which are automatically converted to a

shared network structure. The *nodes* in the network are augmented with constraint filters (both syntactic and semantic) that operate only on locally available parameters. Typically, several independent hypotheses are simultaneously active, and parameter modifications include protection of shared information, such that a parallel implementation would be possible. Efficient memory use is achieved through a recycling mechanism for the node structures, such that they become available in a resource pool whenever their current assignment is completed. These issues will become important as we move towards a fully integrated system.

One of the key features of TINA is that all arcs in the network are associated with probabilities, acquired automatically from a set of example sentences. It is important to note that the probabilities are established *not* on the rule productions but rather on arcs connecting sibling pairs in a shared structure for a number of linked rules. For instance, all occurrences of SUBJECT are pooled together for probability assignments on their children, regardless of the structural positions of these occurrences within a clause. The effect of such pooling is essentially a hierarchical bigram model. We believe this mechanism offers the capability of generating probabilities in a reasonable way by sharing counts on syntactically/semantically identical units in differing structural environments.

Semantic Filtering

For VOYAGER, we were interested in designing a parser that could handle all reasonable ways a person might request information within the domain, but that would also reject any ill-formed sentences, on the grounds of both semantic and syntactic anomalies. Building such a tight grammar not only leads to a very low perplexity for the recognizer, but also virtually eliminates the problem of multiple parses. This is because all parses that are syntactically legitimate but semantically anomalous are weeded out. It has the added benefit of improving computation time, if the semantic constraints are integrated early in the parsing process, more or less at the first chance of resolution.

We also wanted to maintain our criterion that a node should only have access to information locally available to it by default. That is, it should not be allowed to hunt back through the parse tree looking for a resolution of, for example, number agreement. By default, all nodes pass along the parameters passed to them by near relatives. The hard part is to come up with a compact representation that contains all information necessary to carry out the constraints. In terms of syntax, there are patterns describing properties such as person, number, case, and determiner. Semantics are represented by patterns that include an automatic hierarchical inheritance of broader properties from more specific ones. Thus for example the semantic category *Restaurant* automatically acquires *Building* and *Place* as additional semantic features. In addition to the syntactic features, nodes also pass along semantic features that are automatically reset by designated nodes, such as terminal vocabulary entries.

The two slots, **Current-Focus** and **Float-Object** that are used for dealing with gaps, turned out to also be very useful for providing semantic constraint. In fact, we decided to take the approach of *only* using these two parameters for semantic filtering, to see whether in fact that would be adequate. Their use in the gap mechanism is described elsewhere [7], but for clarification we will briefly review it here. *Generators* are nodes that enter their subparse into the **Current-Focus** slot. *Activators* move the **Current-Focus** into the **Float-Object** position, for their children. *Absorbers* can accept the **Float-Object** as their subparse, in place of something from the input stream. The net result of this mechanism, aside from its intended use in gap resolution, is that it provides a second-order memory system for identifying the semantic categories of certain key content words in the history.

As an example, consider the sentence, “(What street)_i is the Hyatt on (t_i)? . The *Q-Subject* places “what street” into the **Current-Focus** slot, but this unit is activated to **Float-Object** status by the following *Be-Question*. The *Subject* node refills the now empty **Current-Focus** with “the Hyatt.” The node *A-Street*, an absorber, can accept the **Float-Object** as a solution, but only if there is tight agreement in semantics, i.e., it requires the identifier *Street*. Thus a sentence such as “(What restaurant)_i is the Hyatt on (t_i)” would fail on semantic grounds. Furthermore, the node *On-Street* imposes semantic restrictions on the **Current-Focus**. Thus the sentence “(What street)_i is Cambridge on (t_i)?” would fail because *On-Street* does not permit

Region as the semantic category for the **Current-Focus**, "Cambridge."

The **Current-Focus** always contains the subject whenever a verb is proposed, and therefore it is easy to specify filtering constraints for subject-verb relationships. Thus for example, the verb "intersect" demands that its subject be *Street* and the verb "eat" demands *person*. We have not yet incorporated probabilities into the semantic predictions, mainly because our domain is simple enough that they don't seem necessary. However, in principle probabilities could be added. Furthermore, these probabilities could be acquired automatically by parsing a collection of sentences and counting semantic co-occurrence patterns.

An indicator of how well our semantic restrictions are doing can be obtained by running the sentence generator with the semantic filters in place. Table 1 gives a list of five consecutively generated sentences from the *Voyager* domain. For the most part, generated sentences are now well-formed both semantically and syntactically.

- | |
|--|
| <ol style="list-style-type: none">1. Do you know the most direct route to Broadway Avenue from here?2. Can I get Chinese cuisine at Legal's?3. I would like to walk to the subway stop from any hospital.4. Locate a T-stop in Inman Square.5. What kind of restaurant is located around Mount Auburn in Kendall Square of East Cambridge? |
|--|

Table 1: Sample sentences generated consecutively by the *VOYAGER* version of *TINA*.

APPLICATION BACK-END

After an utterance has been processed by *TINA*, it is passed to an interface component which constructs a command function from the natural language parse. This function is subsequently passed to the back-end where a response is generated. In this section, we will describe *VOYAGER*'s current command framework, the interface between *TINA* and the back-end, and some of the discourse capabilities of the back-end.

Command Framework

We will illustrate the current command framework of *VOYAGER* by way of the simple example shown below:

Query: Where is the nearest bank to MIT?
Function: (LOCATE (NEAREST (BANK nil) (SCHOOL "MIT")))

LOCATE is an example of a major function that determines the primary action to be performed by the command. It shows the physical location of an object or set of objects on the map. Table 2 lists some of the major functions currently implemented in *VOYAGER*.

Functions such as **BANK** and **SCHOOL** in the above example access the database to return an object or a set of objects. Such functions search for all entries that match the string pattern provided. When null arguments are provided, all possible candidates are returned from the database. Thus, for example, (**SCHOOL "MIT"**) and (**BANK nil**) will return the objects MIT and all known banks, respectively. Table 3 contains some examples of current data functions.

Finally, there are a number of functions in *VOYAGER* that act as filters, whereby the subset that fulfills some requirements are returned. The function (**NEAREST X y**), for example, returns the object in the set X that is closest to the object y. Table 4 contains several examples of filter functions.

Note that these functions can be nested, so that they can quite easily construct a complicated object. For example, "the Chinese restaurant on Main Street nearest to the hotel in Harvard Square that is closest to City Hall" would be represented by,

```

(NEAREST
  (ON-STREET
    (SERVE (RESTAURANT nil) "Chinese")
    (STREET "Main" "Street"))
  (NEAREST
    (IN-REGION (HOTEL nil) (SQUARE "Harvard"))
    (PUBLIC-BUILDING "City Hall")))

```

LOCATE	locate a set of objects
DESCRIPTION	describe a set of objects
PROPERTY	identify a property of a set of objects
DISTANCE	compute distance between two objects
DIRECTIONS	compute directions between two objects

Table 2: Examples of some major functions in the VOYAGER back-end.

STREET	return a set of streets
ADDRESS	return a set of addresses
INTERSECTION	return a set of intersections of two streets
SQUARE	return a set of squares

Table 3: Some examples of data functions in the VOYAGER back-end.

AT	the subset of objects that are at a location
ON-STREET	the subset of objects that are on a street
SERVE	the subset of objects that serve a particular kind of food
NEAREST	the single object of a set that is nearest to a location

Table 4: Examples of filter functions in the VOYAGER back-end.

Interface between TINA and Back-End

Our natural language component does not produce a logical form that is a separate entity from the parse itself. Instead, structural roles such as *Subject* and *Dir-Object* are an integral part of the parse tree. Furthermore, prepositional phrases are given case-frame-like identities such as *From-Loc* and *In-Region* [8]. Because of the availability of such semantic labels within the parse tree, the nested command sequence required by the back-end can be generated by a recursive procedure operating directly on the parse tree.

The parses are transformed to a set of commands in a two-stage process. The first stage establishes the major function of the sentence, and the second stage fills in any arguments required by the major function. Each stage uses a list of entries that contain a parse pattern, a back-end function specification, and one or more argument specifications. In the first stage, each parse pattern corresponds to a sequence of one or more nodes in the parse tree and can specify a hierarchical constraint between certain nodes. Each argument specification corresponds to one or more entries in the second-stage list. In the second stage, a parse pattern

can only be a single node, and each argument specification may be either one or more entries in the second-stage list, a terminal node, or a null value. Terminal nodes return a string from the parse tree, such as "MIT". In most cases, each function specification corresponds to a single back-end function or a null value. When present, a function will be called with its associated arguments, such as (SCHOOL "MIT"). A function specification can also designate that the arguments are wrapped around each other. This mechanism is useful for generating nested filtering operations.

When a parse is passed to the interface component, the patterns in the first-stage list are compared to the parse tree. Whenever a match is found, any argument specifications are passed one at a time to the second stage for resolution. If the argument specification is a single node, only the portion of the parse tree found below this node is processed, thus restricting the domain of the second stage analysis. If there are multiple entries in an argument specification, the first one found is processed in the same way as a single node.

In our previous example, the presence of the word "where" in a trace would result in the specification of LOCATE as the major function. In this case, the first node found in the associated argument specification is a *Subject*. The portion of the parse tree found below *Subject* is thus passed to the second stage. In the second stage analysis the *Subject* node would find an *A-Place* node in the subparse. Evaluation of this node would subsequently generate two arguments (BANK nil) and (NEAREST (SCHOOL "MIT")) which are wrapped to produce the desired result.

Discourse Capabilities

The discourse capabilities of the current VOYAGER system are simplistic but nonetheless effective in handling the majority of the interactions within the designated task. Currently, anaphora resolution is dealt with in the back-end as opposed to the natural language component. We will describe briefly here how a discourse history is maintained, and how the system keeps track of incomplete requests, querying the user for more information as needed to fill in ambiguous material.

Two slots are reserved for discourse history. The first slot refers to the location of the user, which can be set or referred to. The second slot refers to the most recently referenced set of objects. This slot can be a single object, a set of objects, or two separate objects in the case where the previous command involved a calculation involving both a source and a destination. Because of the location slots, user queries can include pronominal reference, such as "What is their address?" or "How far is it from here?"

VOYAGER can also handle ambiguous queries, in which a function argument has either no value or multiple values, when a single value is required. Examples of ambiguous queries would be "How far is a bank?" since there are several banks, or "How far is MIT?" when no default location has been specified. VOYAGER points out such ambiguity to the user, by asking for specific clarification. The ambiguous command is also pushed onto a stack of incompletely specified commands. When the user provides additional information that is evaluated successfully, the top command in the stack is popped for reevaluation. If the additional information is not sufficient to resolve the original command, the command is again pushed onto the stack, with the new information incorporated. In the case where the clarification is also ambiguous, it is pushed onto the stack itself, until it can be clarified. A protection mechanism automatically clears the history stack whenever the user decides to abandon that line of discussion before all ambiguous queries are clarified. An example dialogue illustrating clarification capabilities is given in Table 5.

DISCUSSION

This paper summarizes the present status of our spoken language system development effort. In particular, we focused on those aspects of VOYAGER that have not been reported previously. The system is clearly in its infancy, since it has been in existence for only a few months. In the coming months, however, we hope to improve its performance and expand its capabilities.

Our strategy for system development was to bring up a complete working system as soon as possible, so

User:	Can you tell me how to get to the Chinese restaurant?
Voyager:	Where are you?
User:	I am near a bank.
Voyager:	I know of many banks. Can you specify which one you mean?
User:	The one on Main Street.
Voyager:	The only bank on Main Street that I know of is the Baybank at 226 Main Street in Cambridge. The Chinese restaurants that I know of are Hong Kong and Royal East. Can you specify which one you mean?
User:	The one nearest me.
Voyager:	The nearest Chinese restaurant to the Baybank at 226 Main Street that I know of is the Royal East. I will try to find the best way to go from the Baybank at 226 Main Street to the Royal East. (- Directions -)

Table 5: Example dialogue illustrating clarification capabilities.

that we may study component behavior from a system's perspective. As a result, we devoted most of our resources to constructing missing components, as opposed to refining existing ones. Nevertheless, we have begun to improve phonetic recognition accuracy of SUMMIT by incorporating context-dependent phoneme models. Preliminary evaluation indicates that phonetic recognition accuracy is improved by about 5% with these models on test utterances drawn from TIMIT. However, we have not incorporated this refinement into VOYAGER. We view the improvement of phonetic recognition accuracy as one of the most critical areas to pursue in the near future.

In order to provide an early indication of baseline performance, we have recently started to evaluate VOYAGER using the spontaneous speech database that we have collected. Performance evaluation for spoken language systems is an important issue with which the entire research community is beginning to grapple. In a companion paper, we report our own experience with spoken language system evaluation.

Finally, we should again point out that currently the speech recognition and natural language components are connected in a serial manner, in which the recognizer proposes a string of words and passes it along for linguistic analysis. Such a loose coupling is again a reflection of our desire to bring up a working system at the sacrifice of flexibility and overall performance. In the long run, we firmly believe that speech recognition and natural language must be fully integrated. Experiments in a fully integrated system have been under way, and encouraging results have been obtained. We hope to be able to report on this and other improvements in the near future.

Acknowledgments

We gratefully acknowledge Jim Davis of MIT's Media Laboratory for providing us with the map drawing, path finding, and response generation parts of the VOYAGER back end.

References

- [1] Zue, V., Daly, N., Glass, J., Goodine, D., Leung, H., Phillips, M., Polifroni, J., Seneff, S., and Soclof, M., "The Collection and Preliminary Analysis of a Spontaneous Speech Database," These Proceedings.
- [2] Zue, V., Glass, J., Goodine, D., Leung, H., Phillips, M., Polifroni, J., and Seneff, S., "Preliminary Evaluation of the VOYAGER Spoken Language System," These Proceedings.

- [3] Davis, J.R. and Trobaugh, T.F., "Directional Assistance," Technical Report 1, MIT Media Laboratory Speech Group, December 1987.
- [4] Zue, V., Glass, J., Phillips, M., and Seneff, S., "The MIT SUMMIT Speech Recognition System: A Progress Report," *Proceedings of the First DARPA Speech and Natural Language Workshop*, pp. 178-189, February, 1989.
- [5] Fisher, W., Doddington, G., and Goudie-Marshall, K., "The DARPA Speech Recognition Research Database: Specification and Status," *Proceedings of the DARPA Speech Recognition Workshop*, Report No. SAIC-86/1546, February, 1986.
- [6] Weintraub, M., and J. Bernstein, "RULE: A System for Constructing Recognition Lexicons," *Proc. DARPA Speech Recognition Workshop*, Report No. SAIC-87/1644, pp. 44-48, February 1987.
- [7] Seneff, S., "TINA: A Probabilistic Syntactic Parser for Speech Understanding Systems," *Proceedings of the First DARPA Speech and Natural Language Workshop*, pp. 168-178, February, 1989.
- [8] Fillmore, C., The Case for Case. In *Universals in Linguistic Theory*, E. Bach and R.T. Harms, Eds., Holt, Rinehart, and Winston, Chicago, pp. 1-90, 1968.