A Textual processor to handle ATIS queries

Douglas O'Shaughnessy

INRS-Telecommunications
University of Quebec
3 Place du Commerce
Nuns Island, Quebec, Canada H3E 1H6

ABSTRACT

This paper describes the initial development of a natural language text processor, as the first step in an INRS dialogue-by-voice system. The eventual system will accept natural, spontaneous speech from users and produce responses from the databases in the form of synthetic speech. This paper reports results in processing the textual version of ATIS (Air Travel Information System) queries. The current system (programmed in C) accepts as input the cleaned-up text (SNOR) version of the spoken queries, and produces the desired Official Airline Guide (OAG) information as output. It uses only the words in the input text, and not any punctuation marks, on the assumption that such marks are difficult to obtain directly from speech input. Based on the training text data, the system correctly interprets a large majority of the textual queries.

INTRODUCTION

Speech recognition systems have made significant progress in recent years toward the goal of correctly interpreting continuously-spoken utterances. However, substantial restrictions are usually imposed upon the speaker, to guarantee success. Typically, one must either pause after each word, restrict one's choice of words to a small vocabulary, and/or train the system to adapt to one's voice. In many systems, it is not feasible to insist on vocabulary restrictions, nor can the system always be trained ahead of time to a user's voice. Many applications over the telephone to serve the general public will be of this latter type. Furthermore, most users do not like altering their speaking style, and especially not speaking in isolated-word formant. Thus most practical applications of the future will have to be speaker-independent (i.e., trained ahead of time by other speakers), without major restrictions in vocabulary, and be able to accept normal, spontaneous speech.

In particular, one major application is allowing the general public to do transactions directly with computer databases (including over the telephone). As an example of this type of interaction, we are currently examining a system to permit direct access for a user to air travel information. A user can pose natural questions to the database and receive answers just as a travel agent does. The database is that of the Official Airline Guide (OAG). To simplify the task slightly, we use a subset of the flights in the OAG: only those for airports at nine major US cities (Atlanta, Boston, Baltimore, Denver, Dallas, Oakland, Philadelphia, Pittsburgh, and San Francisco). Otherwise, the entire OAG database is used.

In the future, we will investigate actual voice dialogues between a user and the database, but for now the subject of this study is limited to the analysis of individual queries by users. We wish to design an automatic system to correctly respond to the user with the desired OAG information. As a first step toward this goal, the current study is further limited to the analysis of textual versions of the user's utterances, rather than the speech itself. Thus we assume perfect operation of an initial speech recognizer, which would accept the spontaneous queries of a user and output the word sequence corresponding to the speech. Such word sequences can have grammatical mistakes and repeated words, as often occur in natural speech. Our textual processor must handle such deviations from normal written text that occur with spontaneous speech. In particular, this means that one cannot rely directly on standard English text processors, which presume grammatical input text.

TEXT PROCESSING FOR OAG QUERIES

The task of natural language processing (including deviations as found in spontaneous speech) is difficult (e.g., witness the difficulty of automatic machine translation of natural languages). However, we have simplified the task here by assuming that the user is querying the OAG database. Thus we have a good idea of the type of questions that will usually be asked, and of the typical subjects of those questions. We do not, however, know the format that any individual user may employ. Furthermore, each user is free to use one's own style of speaking and one's own choice of words. We start out with a vocabulary that includes all the words (including names) in the OAG database, and extend that vocabulary to include words discovered during training sessions with trial users. While one could theoretically access a dictionary of over 100,000 words (as might be found in a large English dictionary, augmented by the names found in the OAG database), such an approach is probably inefficient for this OAG application (especially if such a large vocabulary had to be searched in a full speech recognition task). Thus we have chosen to limit ourselves to a dictionary of about 700 words (with separate entries for parts of contractions, e.g., 're, 'll). We also employ a list of 47 common word suffixes (e.g., -s, -ed); unrecognized words with such endings have corresponding final letters removed before trying the dictionary again. For example, the word 'cities' is not in the dictionary; so the final -s is removed (and the 'ie' changed to 'y') to locate 'city' in the lexicon (the plural nature of the located noun is also

noted).

When a word outside this vocabulary is employed, it does not directly affect the text analysis. (From context, we can usually determine the syntactic category of such a word, but not its semantic content.) We assume that such words are not critical for the OAG queries here (empirically this has been generally true; although future training data will likely discover some words which belong in the vocabulary because their presence affects the query response).

A standard parser for English (if one can be said to exist to handle all of natural English) was not used for this OAG application for two reasons: 1) a significant number (perhaps 10–15 %) of the sentences are not grammatical (which would cause ordinary parsers to have errors), and 2) the limited nature of the task does not require a full English parser. In particular, the query system needs only to extract certain critical information from the text queries, and can largely ignore other extraneous information in the text input. For example, the most common query in the training data appears to be asking about flights (e.g., the user specifies departure and destination cities, with optional timing constraints and/or factors dealing with meals and service class, and wishes to receive details about flights that meet these requirements). In such a scenario, the system must determine the identities of the departure and destination cities and properly extract other information relevant to selecting the desired flights from among the hundreds in the database.

Extraneous information in such sentences can be in many forms. Idioms, for example, are common in spontaneous speech, but contain little information relevant to help the system to give the correct answer (e.g., "hello," "all right," "excuse me," "thank you"). One could list all known idioms in the dictionary (to be recognized and ignored when found); however, our approach was to use a relatively constrained dictionary of 700 words and simply ignore words that were not found in the dictionary.

DISCOVERING THE NATURE OF THE DESIRED INFORMATION

One key aspect of the system's task is to identify what type of information the user wants. For example, does the user indicate a desire for a listing of flights, fares, available meals, stopover cities, explanations, etc.? Is the user's request in the form of a question or a statement? Does the user want a list of information, or a simple yes-or-no answer? We follow the convention that appears to be the case in the training examples, in that we give a list of information in virtually all cases. The assumption is that, even when asking a yes/no question, the user will be better informed if he receives more information than actually requested. For example, in response to "Are there any flights to Pittsburgh?," instead of simply responding "yes," we list the appropriate flights. When a person says "Can you show me...?," "Do you know...?," or "Don't you have a ...?," he really does not want only a yes-or-no answer. Thus our system identifies the desired information and lists it, rather than giving a yes/no answer. This also avoids the problem of necessarily determining whether the query is a question or a statement. Usually this latter fact can be discovered by the existence of a reversal of the initial words in the utterance (for a yes-no question), or the presence of a Wh-word (e.g., what, when, how) at the start. We cannot simply look at the sentence-final punctuation, because the query text has no punctuation marks (only words).

We employ heuristics to determine the subject of the request (i.e., the desired information). Keywords are used to discover the subject, and the first such keyword in each query is usually assumed to be the one asked about (ensuing keywords are then assumed to be used to qualify the request). The major keywords include: *flight, fare, time, airport, city, class, code, cost, capacity, distance, reservation, book, ground, date, day, restriction, define, describe, explain, abbreviation.* For example, keywords such as *mean(ing), explain, abbreviation, represent,* and *stand for* signal that an explanation is desired.

In sentences starting with "What is X...?" or "Show me X...," the choice of subject arrives early in the sentence and is obvious. In other sentences, the topic can arrive late (e.g., "All right now, can you please show me all the available *flights*...?"). Sentences starting with "Which X..." or "How (long, far, big, much,...)..." also lead to an obvious topic choice. Those beginning directly with a noun (e.g., "Cost of...") are interpreted as having an implied "Show me the" preceding. Sentences starting with a preposition, on the other hand, are more difficult to analyze as to topic (e.g., "On the flights to Atlanta are meals served?" requests meal information, not a general flight listing); in such cases, the noun in the prepositional phrase is treated as qualifying information.

FILLING SLOTS IN THE QUALIFYING INFORMATION

Most tables in the OAG database contain columns of information organized by type (e.g., codes, flight numbers, company names, days, classes, etc.), and each row is an entry relating typically a code (number or letter sequence) to relevant information describing a flight, a fare, an aircraft, etc. Most requests are filled by listing information from lines in a table in the database. The subject of the request specifies which table to use. To select which lines to list, we must extract relevant qualifying information from the input text (e.g., if 'flights' is the subject, qualifying data may be the departure and/or destination cities and may concern time of travel). After the query subject is established early in each query sentence, ensuing words form phrases and clauses that fill slots in the qualifying information. These ensuing words may form prepositional phrases or relative clauses; no major distinction or classification as to phrase or clause function is needed here, only identification as to what information is contained in the phrases and clauses.

Flight table

It is a simple task to identify city names in a textual query, but more difficult to determine whether a city is the departure, stopover, or destination point. Many requests are straightforward, however (e.g., "...from X to Y..."). Keywords preceding a city or airport name usually identify a city's role: departure (*from, out of, leav(ing), depart(ing)*), destination (*to, land(ing), arriv(ing)*), or stopover (*connecting at, stop(ping) at, via, through*). Lacking such keywords (e.g., "the Atlanta Boston flight"), we assume that the first city is the departing one and the second is the landing one. If only one city is named without keywords, its role must be gleaned from other parts of the query text. (Repeated information is ignored; e.g., "..from Dallas to Baltimore leaving Dallas..") The keywords above set flags to look for a matching city (e.g., when "to" is encountered, the

destination flag is set; upon finding an ensuing city name, that flag is turned off and the destination city slot is filled). Intervening words such as "the airport at" do not affect the flag. If the sentence ends without a match and the query subject is a city or airport (e.g., "what cities does United fly to?"), the system will look under the corresponding column in the flight table.

When the user specifies an airline name, it is easily identified, with the possible exception of companies whose names are uttered as a sequence of letters (e.g. US, TWA). The latter case can cause confusion because user requests often contain letter sequences that refer to other tables or to items other than airline companies. For example, the user may be spelling out the code name for a column in a table, the code name of an aircraft or airport, or the code for a service class.

Aircraft table

As a second table example, the aircraft table is usually accessed via a query about an aircraft model number (e.g., "what is a 737," "describe a D8S"), but it may also be queried in terms of its column entries (e.g., "what airplane is the fastest," "which plane has the longest range"). Each column is labeled with a noun heading, to which relevant adjectives are associated (e.g., the *weight* column is associated in the system to the descriptors *heavy* and *light*), which allows comparative requests between aircraft. Where the OAG model number differs from the company's public model number (e.g., a DC10 is officially a 'D10'), the system notes this as a special case.

SEQUENCES OF LETTERS

A two-letter code followed immediately by a digit sequence is tested to be a possible airline name + flight number, by looking for that entry in the flight table. A three-letter code invokes a search of the airport table, for a possible airport code name; a four-letter sequence calls for a possible city code. A letter sequence containing a slash ('/') invokes a look at the restriction table.

Since the input text is all in capital letters, the distinction between the letter 'A' (as part of a code name) and the article 'a' can lead to ambiguities. For example, 'WHAT IS A D EIGHT S?' requests an entry in the aircraft table corresponding to the code 'D8S.' However, there conceivably could be a code elsewhere in the OAG database of the form 'AD8S.' The system looks first to match the longer letter (and digit) sequence; if no match is located, it strips off the initial 'A' and tries again.

Users may utter a code name of two or more letters as a single word. Such pronounceable code names are included in the system dictionary. Confusions can arise when such words also have other meanings. For example, in "WHAT DOES AS MEAN (IN THE AIRLINE TABLE)?," if the user does not specify the airline table (where "AS" means Alaska Airlines), the system might not understand that "AS" is a code (and not a conjunction). Given the frequency of ungrammatical queries in the training data, this is not unreasonable. However, the system looks for the subject of the query when it sees the word "mean(ing)" in the context "what does X mean" or "what is the meaning of (the) X."

NUMBERS

Numbers in the input text can refer to dates, times, prices, groups of people, flight numbers, flight codes, fare codes, aircraft codes, etc. The system uses context to correctly interpret digit sequences as numbers. For example, ordinal numbers (except for 'first' – which is often associated with 'first class') are usually associated with dates (similarly for cardinal numbers adjacent to a month name); a number followed by 'a m' or 'p m' is also easy to interpret as a time. Numbers preceded by an article (e.g., 'a 737') are tested to see if they match a model number for an aircraft. More interesting are cases of numbers run together; e.g., "Is the departure time for two thirteen four twenty?" (flight 213 leaving at 4:20?).

The system assumes that numbers are spoken following certain syntactic rules. In particular, people say times as hour + minutes (e.g., 11:40 is 'eleven forty', and not 'one thousand one hundred forty' or any other possibility). Digits are converted into a full number form (e.g., 'sixteen eight twenty' =16820), including time of day (e.g., 'seven o'clock' = 700); thus someone using military time (e.g., 'eighteen hundred hours' = 1800) will be properly interpreted.

Faced with a number of several digits (e.g., a flight number or code), people usually pronounce it digit-by-digit. For 3- or 4-digit numbers, however, the pronunciation is often grouped into digit pairs (e.g., 'flight twenty three forty two' = 2342). Lastly, there is the question of interpeting times as AM or PM; when not explicitly specified, the system assumes flights at reasonable hours (i.e., no departures or landings between 11 pm and 6 am) (e.g., 'twelve o'clock' means 12:00 and not midnight).

If a digit sequence is preceded by the words 'flight (code)' or 'fare (code),' the identification of the sequence is obvious. Otherwise, a six-digit sequence starting with '1' is assumed to be a flight code, and a seven-digit one starting with '7' to be a fare code. The sequence 'nineteen ninety-X' after a word sequence containing a month is interpreted as a year.

The preferred times of flights can be specified as: 1) 'after X' and/or 'before Y,' 2) 'between X and Y,' or 3) 'around Z.' Alternatively, the user may specify vague times with terms such as *morning, evening,* and *night.*

A number between about 80 and 1000 is assumed to be a fare if followed by the word 'dollars,' adjacent to the word(s) 'fare (of),' or even followed by the word 'flight.'

SPECIAL REQUESTS

Occasionally, the user wishes to view the desired information in a specific fashion, e.g., flights ordered by departure time, or fares in order of increasing price. This is determined by the keywords 'sort(ed)' or '(in) order(ed)' plus 'by X' or 'de/increasing X' (where X is price, weight, etc.), or 'alphabetically.'

Mathematical operations are sometimes requested: "the difference between fares class Y and F," "the difference in time between Atlanta and Dallas." The keywords *difference, sum,* and *average* invoke the corresponding mathematical operations using values extracted from the tables for the coordinated items mentioned immediately after the keywords (e.g., "the average fare for classes Y and F").

COORDINATION

Coordination in general is a difficult computational linguistic problem. The system attempts to group words and phrases on as local as basis as possible. Thus, the word *and* (or *or*) will link adjacent words to form a single unit if the words are from the same syntactic class (e.g., "between Dallas and Baltimore", "fares for taxis and limousines"). If necessary, larger units are grouped next (e.g., "Delta 402 and United 567"); finally the conjunction is treated as separating clauses if followed by a verb (e.g., "... and list the..."). A local coordination is verified, if possible, through the appearance of a plural classifying word just before or after the coordinated units (e.g., "flights thirty four and ninety three," "the Y and F classes").

The coordination routine normally links at the most local level (e.g., "flights from Oakland or Dallas to Atlanta" will group the first two cities as departure sites). However, if an inconsistency arrives immediately afterward (e.g., an attempt to fill a slot already filled), the routine will attempt to link larger units (e.g., "flights from Boston to Pittsburgh and Dallas to Atlanta" would normally link Pittsburgh and Dallas as destination cities, but the "to Atlanta" words are inconsistent with that interpretation; so the coordination routine will group the first two cities together and the last two cities together, giving two listings as output.

The conjunctions *and* and *but* invoke a logical 'and' (intersection) when linking separate qualifying information (e.g., "leaving Boston and landing at Atlanta"), whereas *or* invokes a logical 'or' (union) (e.g., "arriving at or before five o'clock"). On the other hand, when the words immediately following an *and* relate to a subject topic (e.g., "show the flights and fares..."), then the topic is augmented to deal with both items (e.g., list both flight and fare information). Similarly, when the words after an *and* attempt to fill qualifying information slots already filled, the system produces an output using the information up to the *and*, and then continues further using the new qualifying information (e.g., in the 4-city example above, flights would be listed first for the first two cities, then for the next two).

COMPARISON

Some of the queries request a comparison of numbers (e.g., "list flights under three hundred dollars," "which airline has the most flights"). When a comparison keyword is located (e.g., *under, more, less, last, earliest, next*), the direction of the comparison (more vs. less) is noted, and the ensuing noun describes the item being measured (e.g., cost, number of flights, time of flight, etc.). In the case of *more* or *less*, the noun after the ensuing *than* is used for comparison to the subject of the query.

WORDS TO IGNORE

Many words are effectively ignored during the processing. For example, *some* and *all* (as in "show me some/all ...") have no relevance, since the system shows all possibilities in any case. Similarly, expressions such as "please," "OK," and "I'm sorry" (while useful in a polite, user-friendly interface) are ignored here. Also, some users have a habit of saying letters followed by "as in X" (e.g., "class Q as in queen"). This brings words (e.g., 'queen') into the dialogue that are invariably outside the system vocabulary. When the system sees LETTER + "as in", it ignores the "as in X"-phrase.

Since the system does not attempt to make a complete parse for the text input, it can handle word repetitions by speakers (as found with interruptions and hesitation pauses). While immediate word repetitions are ignored (on the assumption of possible hesitations), when the repetition is a digit, the full resulting number is first tried in the table look-up. For example, in "twelve twelve ninety," the number is assumed to be 121290; if no match is found in the tables, then the number 1290 is assumed.

DISCUSSION OF RESULTS

The system was officially tested on February 6, 1991, with the results that 54 queries were correctly answered and 94 were not. At the time, the system was not set up to give a "no answer" for cases that it did not feel confident. The relatively poor performance can be largely explained by the fact that the system was prematurely tested, without having been properly debugged. As of March 6, most of the bugs had been removed and the system was tested again on the same sentence queries, with the results that 110 were answered correctly, with 19 false responses and two "no answers." This second testing was done with the benefit of having examined the test data, to correct the program, both from the point of view of system bugs and inadequate coverage. The majority of the improvement was simply due to eliminating system bugs. A large majority of the remaining incorrect performance can also be readily eliminated with a little more effort. Thus, the approach described in this paper is certainly capable of handling queries typical of the ATIS data in the range exceeding 90%.

We examine now where the revised (debugged) system does well and not so well, and point out where the recent improvement is due to rule modification (to better cover more types of queries, as revealed by certain queries in the February 1991 test set) as opposed to simple system debugging. Since the system ignores words that it does not recognize, it continues to make a mistake on sentence cj0011sx, where "from Dallas Love Field" is interpreted as simply "from Dallas." Sentences ci00h1sx and ci00c1sx (noting December 14th as "121490") are now covered, due to a rule addition permitting dates in pure digit form (this type of date representation was new in the test data). Sentence cp00p1sx ("..earliest wide-body flight...") is now easily handled, with the addition of a rule testing for the first flight of the day (if "earliest" or "first" appears just before the word "flight"), as well as the last one (keywords = "last," "latest").

Looking at sentences which caused the most problems in the official February 1991 results (e.g., those for which at most two of the eight sites who submitted results were correct), we can see examples of where our system can correctly handle difficult queries. It is our system's ability to ignore irrelevant words and not require a full parse that allows it to accept syntactic and semantic structures that have not been seen in the training data. In sentence ci00k1sx ("Can you please tell me what time zone Dallas would be on thank you"), both the initial five words and final five words are irrelevant to the message and are ignored by our system, which seizes upon the initial, subject keywords *time zone* and ensuing location name *Dallas* to produce the correct answer. The preposition *on* at the end of the sentence can cause problems for parsers that insist on accounting for every word and which involve a semantic module (for which a city should not be "on" a time zone). Similar comments hold for the final preposition

to in sentence cl00w1sx ("Please show all cities that Delta airlines flies to"), whereas our system sees *cities* as the subject keyword and *Delta* as the only other relevant information.

In sentence ce00p1sx ("How long does it take to drive from the airport to downtown Atlanta"), the subject keyword is *how long, does it take to* is ignored, *drive* specifies ground transportation, and the remaining words fill in the *to - from* slots. By ignoring distracting words such as *menu* and *seat* (in sentence ch00k1sx - "...menu of departures..."; in sentence cl00d1sx - "how many persons does a 757 seat"), our system avoids mistaking the subject of some queries. Similar comments hold for the word *major* in sentence cj00i1sx ("closest major airport to San Francisco").

Our system correctly handles even most cases of mentioning of locations outside of the 11 cities of the database (although the case of "Love Field" above shows its limitations). In sentence cj0081sx ("...with a stop in Las Vegas"), the system looks for a stopover location after the keywords *stop in*; finding words there which are not in the dictionary, the system assumes a location outside the database.

In sentence cp0021sx ("Does American flight 1010 leaving at 1303 have any stops enroute"), the initial keyword *does* cues a yes-no question; *flight* is not taken as the subject because *number* and digits ensue immediately; instead, *stops* is taken as crucial information and the potentially confusing word *enroute* (which may not have appeared in earlier training data) is ignored. In the stilted sentence cp00f1sx ("How many engines does a D 10 equipment have"), the subject is identified by the first three words, the letter/number sequence D10 is noted as a model number, and *equipment* is treated as superfluous data.

There are cases of ATIS queries in which a full parser can be useful, but there are many other cases such as these above where not performing a full parse and ignoring superfluous words can accomplish the task as well with less effort.

DIFFERENCES BETWEEN TEXT AND SPEECH PARSERS

In recent years, there has been considerable work on parsing of general text in the context of natural language analysis [1]. A parser in a speech recognition context, however, encounters problems that a text parser does not have [2]. For example, in determining syntactic structure, a text parser has access to punctuation (e.g., quotation marks, parentheses), capitalization, and other phrase-offsetting devices (e.g., italics, underlining). Major punctuation marks (periods, exclamation and question marks) denote the ends of sentences, and others (colons and semicolons) mark the ends of major clauses; a text parser can thus easily determine major syntax boundaries and need only operate on sets of words between such markers to parse each set into a logical clause or sentence. For a speech parser, on the other hand, gross segmentation cues may take the (unreliable) form of pauses (e.g., speakers often pause at major syntactic boundaries - but not consistently - and hesitation pauses can cause significant difficulties). Swings in vocal fundamental frequency which are often correlated with syntactic boundaries [3] also furnish (at best) unreliable indicators for a parser to use. In text, the appearance of capital letters (except, of course, at the start of a sentence) indicates a proper name (and thus usually a noun); such a fact can help a text parser distinguish such words which may alternatively (without capitalization) be used as other parts-of-speech. A speech parser has no access to such information found readily in texts.

In the context of data entry via voice, one could envision requiring a user to pronounce aloud markers such as capitalization and punctuation. However, this forces a departure from natural speaking style (and slows the rate of data entry) and so is not preferred in most applications. In the application explored in this paper - an isolated-word system - only the actual words were pronounced (as in speaking naturally, except of course for the brief pause required after each word). In isolated-word systems, no durational information (i.e., from pauses or from word lengths), however unreliable, can be exploited to determine the syntactic function of individual words.

ACKNOWLEDGMENTS

This work was supported by the Networks of Centers of Excellence program of the Canadian government.

REFERENCES

[1] Dowty, Kattunen, & Zwicky, eds. *Natural Language Parsing* (Cambridge University Press, Cambridge, UK), 1985.

[2] Bates, M. "The Use of Syntax in a Speech Understanding System," *IEEE Transactions ASSP*, vol. ASSP-23:112-117, 1975.

[3] O'Shaughnessy, D. "Linguistic Features in Fundamental Frequency Patterns," *Journal of Phonetics*, vol. 7:119-145, 1979.