

Query-by-critique: Spoken Language Access to Large Lists

Dan R. Olsen Jr, Jon R. Peachey

Brigham Young University, Computer Science Department

Provo, UT 84602 USA

{olsen, peachey}@cs.byu.edu

ABSTRACT

Spoken language interfaces provide highly mobile, small form-factor, hands-free, eyes-free interaction with information. Uniform access to large lists of information using spoken interfaces is highly desirable, but problematic due to inherent limitations of speech. A speech widget for lists of attributed objects is described that provides for approximate queries to retrieve desired items. User tests demonstrate that this is an effective technique for accessing information using speech.

KEYWORDS: Spoken language interfaces, search, tables

INTRODUCTION

The ICE (Interactive Computing Everywhere) project is focused on building infrastructure and interactive techniques that allow people to interact with information and services from a variety of physical situations. We see speech as a major component of such efforts. The major benefits of speech are that it is independent of physical posture, can be hands/eyes free, requires only a small physical size, and has very low power requirements.

Our approach is to create a set of “speech widgets” that can be readily composed to create a variety of applications, rather than natural language dialogs. Such widgets would have a standard “hear and say,” much like the “look and feel” standards in graphical user interfaces [10]. With a standardized widget set, developers can spend a significant amount of time working out the usability of the widgets. This effort can then be easily leveraged across all uses of the widget.

Our approach to such speech widgets is to organize our interactions around information structures. We see the manipulation of information as our guiding interactive paradigm. Widgets that handle atomic values such as numbers, dates, times, menus, and selection from small, enumerated lists are quite common. It is also common to collect such components together into a tree that provides

standard navigation mechanisms. Such structures create essentially fixed, finite sets of information. What are missing are standard mechanisms for working with large amounts of information using spoken language interfaces.

The simplest data structure that can handle an arbitrary amount of information is the list. However, lists are very problematic in spoken language interfaces. Listening to list items is much slower than visually scanning that same information. The transient nature of speech, coupled with limits on short-term memory makes it difficult for users to maintain a sense of context. Despite these difficulties, there is still a very strong need to access large amounts of information through speech.

Our model for lists is a table structure like that shown in Figure 1. There is a list of objects and a fixed set of attributes for those objects. However, in the problems we are interested in, there are tens to thousands of items and possibly tens of attributes.

Name	Age	Height
Fred	22	72
Joan	17	68
Gerald	40	73
Jackie	25	66
...

Figure 1: A Tabular List

It is infeasible to scroll through such a list because speech is such a slow feedback channel. The only possible approach is to search. However, short-term memory makes formulating a query difficult. Speaking a query in a restricted natural language is possible, but that leads to long complicated utterances. The need for feedback to handle recognition errors means that the query must exist as a separate object that the user must remember and manipulate. The spoken dialog becomes focused on the query as much as on the information being sought.

Queries are problematic for users even in graphical situations. Users are frequently vague about the things that they are searching for. As users peruse the data their sense of what they want will drift. While searching for a car of a particular make and year, desires will shift as the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
UIST'02, October 27-30, 2002, Paris, FRANCE.
Copyright 2002 ACM 1-58113-488-6/02/0010...\$5.00.

sticker price becomes apparent. A formal query is not conducive to these needs.

Our approach to manipulating lists is to present an information object to the user. The user then critiques the object by commenting on desired values for the attributes. After each critique utterance we rank objects based on the recent history of user comments and present the most likely object. The user's attention is always focused on the information object at hand rather than on an auxiliary query object.

QUERY BY CRITIQUE

A table representation of information like that shown in Figure 1 is adaptable to a wide variety of applications. There are three general interactive tasks that relate to such lists.

- Editing and modifying the list or its elements
- Getting an overall sense of the content of the list
- Locating desired items in the list.

In large lists, the ordering is rarely important other than sorting in various ways. Therefore insertion of new items is simply a case of specifying the attributes of the new object. This is an instance of filling a fixed set of fields, for which there are already good speech dialog forms. Deleting items involves selecting the items and then specifying deletion. The hard part is selecting the items to begin with. Modifying items is similarly dominated by the selection task. All the really hard parts involve locating desired items. We will focus on the selection task.

Getting an overall sense of the contents of a list is an important task. What is required is a summary of a list of items. Though this is of value, we have not studied this task.

This work is focused on designing spoken language dialogs that help users find desired items. There are two fundamental approaches to this problem. The first is to allow the user to specify some predicate and then report back on the items that satisfy that predicate. This is the traditional form-query approach. The second strategy is to define a ranking on the items and report them in highest to lowest order.

Predicate-based queries

Consider a user trying to locate an apartment for rent. Using a predicate approach, we might have the following dialog:

- U1: "rent less than \$500"
C1: "there are 35 items"
U2: "city is Spanish Fork"
C2: "there are 0 items"

At this point the user can become a little confused. There is no information about whether there might be apartments for \$550 or possibly in a town near Spanish

Fork. The rigidity of the predicate is not helping the user solve their problem. To break the impasse the user might say,

- U3: "rent less than \$525."
C3: "there are 0 items"

We now have a query algorithm problem. By implicitly intersecting all of the user's utterances, this new utterance will not expand the selected set. At this point the user is now confused about what the query really is. We can have the system start to speak the query, but if more than 3 attributes become involved, the time spent listening to the current query starts to dominate everything else.

A second problem arises when the user wishes to modify the query. We must add dialog that removes or modifies components of the predicate. We soon start to have more dialogs about queries than we do about the actual information the user is working with. We can eliminate the notion of a query object by having users speak queries as full sentences, such as "Apartment with rent less than \$500, bedrooms greater than 1, air-conditioning, parking is covered, and city is Spanish Fork." Not only does such a sentence pose recognition problems, it is also very hard for a user to create and correctly understand.

One approach to such list queries is to simplify the query model. Instead of conjunctions and disjunctions of simple predicates, queries can be modeled as upper and lower bounds on each attribute. Equality is a special case where upper and lower are the same. This simplifies what the user must understand and what must be spoken. For the most part the query can now be changed by new statements. Utterance U3 would move the upper bound rather than add a new conjunction. This simplified query model does not correctly handle attributes without a sorted order.

A key difficulty that remains is the fact that the system is still reporting either too many items to use effectively or no items at all. The user then must guess at possible query modifications that will yield some information. This is not acceptable.

Search by Ranking

An alternative approach is to provide a ranking on the data items and let the user work down the list from highest to lowest. In this formulation the user is always working in terms of concrete data. For example, they are continually hearing about possible apartments. Our dialog may proceed as follows.

- U4: "rent less than \$500"
C4: "apartment rent is \$500, city is Salt Lake, bedrooms is 1"
U5: "city is Spanish Fork"
C5: "apartment city is Spanish Fork, rent is \$600"

In this dialog the system is always attempting to satisfy the requirements that the user is expressing but is violating some of them in cases where they cannot all be satisfied. In C5 the system has suggested an apartment that is in Spanish Fork but violates the rent requirement. The user finds the rent in C5 to be a little high and might say:

U6: “next”

C6: “apartment city is Mapleton, rent is \$550”

The next most highly ranked item is spoken. Mapleton is a town that is close to Spanish Fork. This second choice sacrificed the city requirement in favor of rent closer to \$500.

In this ranked approach there is no explicit query object that the user must understand. The user’s utterances control the ranking. However, as a dialog proceeds the user’s desires may shift. The user may recognize that rent is higher than they expected and start focusing on other issues, such as parking facilities for their new car. The user may also give up on Spanish Fork and decide to get a job in Salt Lake thus changing everything. Our approach to this shift of interest is to have the importance of user utterances decay over time. More recent statements have precedence over earlier ones. Thus the user’s shift of focus is implied by the user’s current utterances.

It is this more fuzzy approach to queries that we have chosen for our list speech widget. The speaker’s expressed desires are used to compute a ranking on the items. We believe that this form of search dialog is more natural and closer to how one would speak with a real estate agent or automobile salesman.

Speech Feedback

There are many attributes of an item such as an apartment. These include security deposit, air conditioning, whether there is a pool, etc. Speaking all of these attributes to a user will create overload. Some attributes are more important to a particular person than others. Some are desirable to know but not as important in selecting an apartment.

In computer response C4 the number of bedrooms was mentioned and then omitted in C5 and C6. This is because the user had not expressed any interest in this attribute. It must be possible, however, for the user to find out about any of these attributes. In C5 and C6 the spoken order of the attributes have changed to reflect the user’s stated interests. The system must also track what the user is interested in so that object feedback is tailored to those interests.

QueryByCritique language

Based on the above discussion we can now define the spoken language supported by our list widget. There are four types of commands that the user can say.

- Attribute critique
- Attribute Inquiries
- List navigation
- Help

Attribute critique commands

The heart of the widget is in the attribute critique commands. These commands reflect two different ways in which people approach problems. In one approach the user comes to the system with definite attributes in mind such as the amount of rent they can afford and the number of bedrooms that they need. In other scenarios the user is fuzzier about their goals. In many situations the user is responding to the data and making evaluations of its appropriateness. We reflect these approaches in two styles of commands, which we term *value-specific* and *example-relative*. We believe these two classes are each suited for different search problems.

The value specific commands have the following forms:

- <attribute name> less than <value>
- <attribute name> greater than <value>
- <attribute name> is <value>

These commands are readily instantiated for virtually any application where the attributes have a sorted order that is clear to the user. Good examples are rent and number of bedrooms.

Some attributes, such as cities, do not have an interesting sorted order. Sorting them alphabetically may be algorithmically correct, but not at all helpful in our example dialogs. There are a broad class of attributes which are not ordered, but do have a distance metric. In the case of cities we can compare them on the number of miles they are apart. For such *distance-comparable* attributes only the “is” command is appropriate.

The example-relative commands implicitly use the attributes of the last data object presented by the computer.

- like <attribute name>
- don’t like <attribute name>
- like [this | <object type>]
- don’t like [this | <object type>]
- higher <attribute name>
- lower <attribute name>

The “like” and “don’t like,” commands either attract or repel objects with those attribute values. Liking or disliking an object is the same as commenting on all of the attributes heard about the last object presented. The higher and lower commands are shorthand for the greater

than and less than commands. If the apartment last spoken has rent of \$700, saying “lower rent” is the same as saying, “rent less than \$700.” The user is referencing the current object’s values.

An additional advantage of the example-relative commands is that the speech recognizer does not need to recognize attribute values. Only a limited command set plus the attribute names must be recognized. For example “don’t like color” can be more robustly recognized than “color is *colorName*” where the range of color names can be very large.

Navigation commands

These simply move up and down through the current list rankings. They are simply “Next” and “Previous.”

Attribute Inquiries

Most of the dialog fragments require the user to know about the available attributes for an object. In addition, only a very few attributes can be spoken quickly enough while searching. The inquiry commands are:

- What is <attribute name>?
- What are the attributes?
- Describe / More

The “Describe” command will speak the first few attributes of an object. The “More” command will speak additional attributes.

Help

Throughout all of our speech widgets, “What can I Say” along with “More” will describe all of the commands.

RELATED WORK

A practical example of using a spoken language interface to access a large amount of data is VPQ (Voice Post Query), a system developed by AT&T to provide spoken access to their corporate personnel database consisting of more than 120,000 entries [1]. VPQ uses natural language understanding, dialog control components, and dynamic constraint analysis to locate answers to the user’s exact query. Our widget takes a much more simplistic approach.

Our approach to locating information relevant to the user is similar to the system explained in [5], which produces a ranking of database objects in response to a query containing vague predicates. In the referenced system, the user makes binary relevance judgments about the retrieved objects, which the system remembers and uses to produce improved rankings. We apply the same idea to a spoken query interface in QueryByCritique in which the entire list of data is ranked based on similarities to the user’s spoken query. However, our approach is based on more than the user’s binary relevance judgments of retrieved objects. The user is able to critique each

attribute value and specify just how the system should adjust that value.

Intelligent information systems have been created to provide users with data that may not be exactly what was asked for in a query, but may in fact be relevant to what the user is looking for. *Cooperative answering* systems [4,6,7] use generalization, specialization, and relaxation of queries to capture related or neighboring information from a structured database. This would resolve the user’s vagueness problems but still requires a query to be expressed and managed and does not handle drift in user intentions.

Solutions to the search problem for speech interfaces have been developed for specific applications. The NJFun System [11] was developed to provide users with access to information on fun things to do in New Jersey. NJFun uses a reinforcement learning approach to automatically optimize the dialog policy and help the user find what he is looking for quickly. Our approach steers away from dialog optimization and concentrates primarily on allowing the user to explore the data. Our dialog consists of basic critique commands made by the user and immediate feedback of ranked items to the user.

Similar to reinforcement learning are *recommendation systems* [2,3,9], which can be used for any task that requires choice among a large set of predefined items. Adaptive Place Advisor [8] is an example of a recommendation system in which the user interacts with the system to narrow down choices by answering a sequence of questions aimed at removing alternatives, rather than simply ordering them. QueryByCritique grants the user full control over their comments on the data rather than imposing a sequence of questions upon the user. These approaches produce very crisp queries that are also not responsive to drift in the user’s intentions.

CRITIQUE QUERY ENGINE

Our list query engine was specifically designed to reflect the nature of the user dialog by controlling the ranking of items in the list based on the user’s utterances. An object, which we call an *utterance*, is used to model every critique command made by the user in a given session. Each utterance object contains the user’s spoken text, a list of attribute value distributions that define how to critique each of the mentioned attributes, and an importance weight. A history of recent utterances is maintained. List items are ranked based on their relationship to this utterance history.

Figure 2 shows a sample utterance history where each line in the table represents a single utterance made by the user. Each time the user utters a new critique command, an utterance object is added to the top of the history and the entire list of data items must be re-ranked according

to how well they evaluate against the utterance history. As utterances get older, they become less important to the ranking of list items.


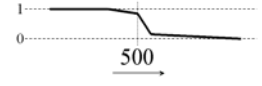
	Utterance	Distribution	Weight
1	"Bedrooms is 3"		1.0
2	"Rent is less than \$500"		0.95
...

Figure 2 - Sample utterance history

Attribute Value Distributions

Each utterance type defines a distribution over the set of possible attribute values. These attribute value distributions model the user's preference for a given attribute as specified in the utterance. These distributions are somewhat different for distance-comparable attribute types than they are for numeric attributes. In describing these distributions we consider separately the equality and inequality utterances.

Equality Utterances

In an equality utterance, the user speaks about a specific attribute value such as "bedrooms is one", "like rent", or "don't like city". For the "is" and "like" utterances the attribute value distribution is shown in Figure 3. The "don't like" distribution is shown in Figure 4.

Distributions range from 0.0 (inappropriate values) to 1.0 (most appropriate values). The distributions are centered on the value V , which is the attribute value mentioned by the user. Because we want a fuzzy rather than an exact reference to the value V , we use a piecewise linear distribution around V . The width W of the area of interest is defined as half of the standard deviation ($\sigma/2$) of the values for the given attribute.

Using these distributions we can define a function $utter(U,I)$, where U is an utterance from the history and I is a list item to be evaluated. In the case of these simple equality utterances, the $utter$ function returns the distribution value for the appropriate attribute value of I . In the case of utterances that reference an entire object, such as "don't like apartment," the $utter$ function is the average of the distributions for all attributes of the item I .

In the case of distance-comparable attributes such as cities, there is no ordering of the values. However, we can compute a distance between any two values for the attribute. For such attributes we can use the distributions shown in Figures 5 and 6. These are distributions on the distance between the value V and all other values. The

width W is $\sigma/2$ using the average of all distances between attribute values in place of the standard deviation.

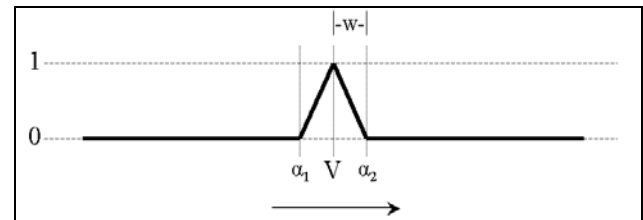


Figure 3 - Equality Distribution

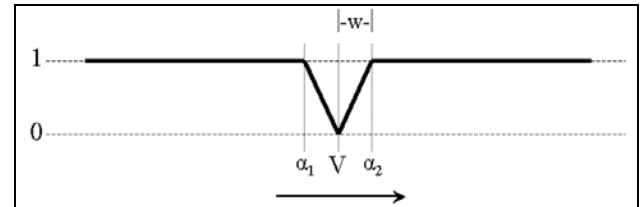


Figure 4 - "Don't Like" distribution

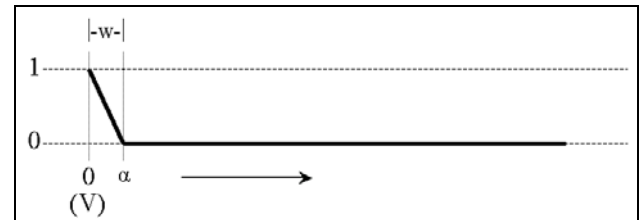


Figure 5 - Distance-comparable equality

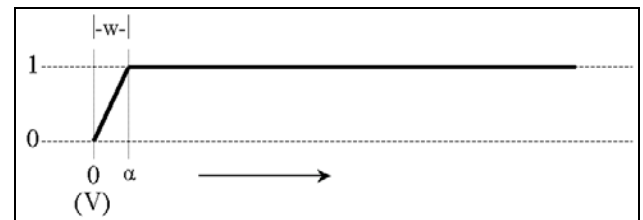


Figure 6 - Distance-comparable "Don't like"

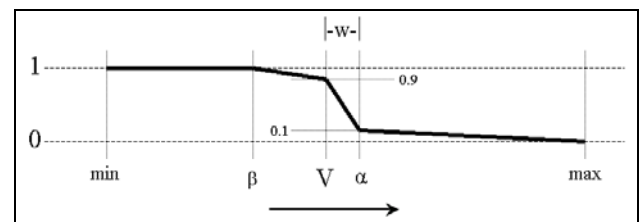


Figure 7 - "Less than" distribution

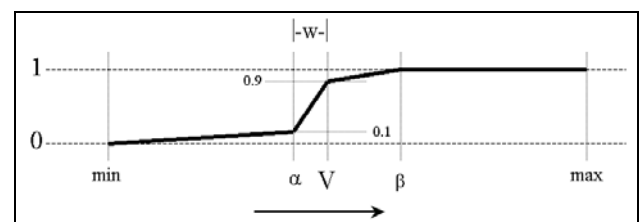


Figure 8 - "Greater than" distribution

Inequality Utterances

The inequality utterances include statements like “bedrooms greater than 1”, “lower rent” or “price between \$100,000 and \$500,000”. In designing these distributions, we must take into account that users are fuzzy in what they intend. Asking for rent less than \$500 does not mean that the user would refuse an otherwise great apartment that rented for \$510. The distributions for “less than” and “greater than” are shown in Figures 7 and 8. Remember that “lower” and “higher” are shorthand utterances for these same distributions.

As with the equality utterances, the distribution is focused on the value V , which is the value expressed by the user. Since these distributions are symmetric, we will only explain the distribution for “less than”.

In designing these distributions, there are two goals. The first is to account for the fact that user statements are only approximate. The user may accept a slightly higher rent than specified. This approximate behavior is accounted for by the α point in the distribution. As a secondary goal we have a desire to focus the user’s search more rapidly. In such a case we want to push the desired values away from the specified value to sample more broadly. For example, when a user specifies rent less than \$500, we have no idea how much less is desirable. For this reason we want to push towards values that are much lower rather than just a little lower. If the result is undesirable, the user can so specify using a subsequent “higher” or “greater than” utterance to push back the other way. This pushing is handled by the β portion of the distribution.

On the α side of the distribution the width $W = \sigma / 5$. This is much smaller than equality because we still want to strongly favor the user’s expressed wishes. Note that the distribution value at the α point is 0.1 rather than 0.0. The distribution slopes off to zero at the maximum (for less than). Subsequent utterances may provide stronger weight to other attributes. This slope will still tend to favor smaller values even if they are larger than what the user requested. Our goal is to provide the user with the best selection we can, given the data actually available in the list.

The β portion of the distribution is attempting to push towards more interesting choices for the user. We compute the β position of the “less than” distribution as $\beta = V - ((V - \min) * F)$, where F is our “push factor”. In the current implementation $F = 0.5$. This pushes interesting values some percentage of the distance between V and the minimum value. The β point is quite effective when the utterance history contains a “less than” and a “greater than” utterance for the same

attribute. The slopes to the β points of each utterance combine to form a peak in the total distribution close to half way between the expressed bounds.

The final inequality is the “between” utterance. This has the distribution shown in Figure 9. It has similar approximate properties to the “greater than” or “less than”. The width $W = \sigma / 5$.

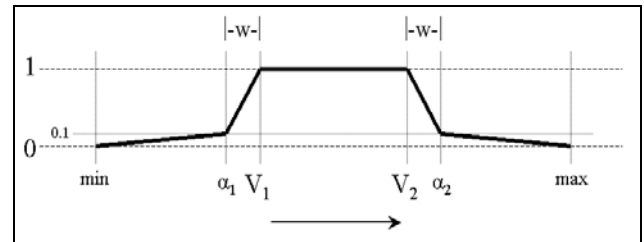


Figure 9 – “Between” distribution

Time Decay

As users review data, think about their goals, or just change their minds they tend to shift their intended search predicate. Frequently in conversation people do not announce such a shift, it is just inferred as time goes by. To accommodate this behavior we let utterances decay in importance over time. Each new utterance is given a weight of 1.0 and all previous utterances have their weight reduced by a decay factor. In our current implementation the decay factor is 5%. Setting the decay very high (for example 50%) causes prior statements to rapidly become irrelevant. Setting it too low (for example 1%) causes the query to be influenced by statements the user can no longer remember having said. Our current setting of 5% seems to work, but we have not explored this in depth.

Ranking computation

The utterance history, the utterance distributions, and the time decay weights all combine together to produce a ranking of each item in the list. This ranking of a data item is the weighted sum of evaluations of the data item against each utterance in the history. The relevance of a data item against an utterance is equal to the factor of the evaluation of the distribution function and the weight.

$$rank(I) = \sum_{U \in \text{utterances}} \text{utter}(U, I) * U.\text{weight}$$

Example

Using the data list in Figure 10, we take each item and evaluate it against every utterance in the history from Figure 11.

	City	Rent	Bedrooms	
1	Provo	\$200	1	1.0
2	Orem	\$500	4	0.9
3	Salt Lake	\$800	3	0.3

Figure 10 - sample data list

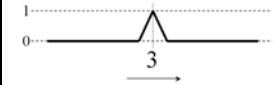
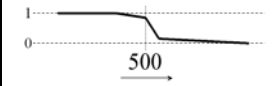
	Utterance	Distribution	Weight
1	“Bedrooms is 3”		1.0
2	“Rent is less than \$500”		0.95
...

Figure 11: Sample utterance history

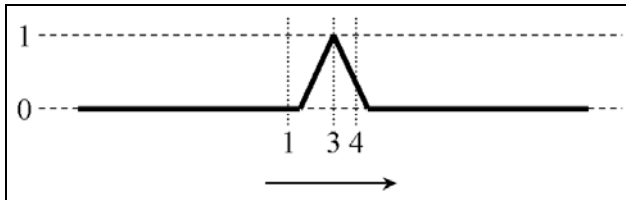


Figure 12: Utterance 1 distribution

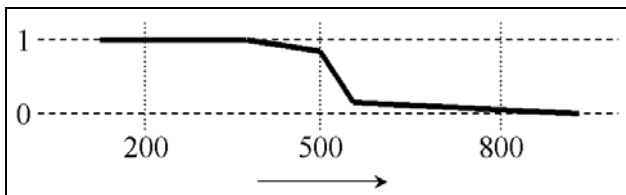


Figure 13: Utterance 2 distribution

Apartment 1 with bedrooms = 1 is evaluated against utterance 1 (see Figure 12), which returns a value of 0.0. The value is then multiplied by the importance weight of the utterance, which yields a relevance of $r1 = 0 * 1.0 = 0$. Apartment 1 with rent = \$200 is then evaluated against utterance 2 (see Figure 13), which yields a relevance of $r2 = 1.0 * 0.95 = 0.95$. The total relevance of apartment 1 is:

- $R = r1 + r2 = 0 + 0.95 = 0.95$

Apartment 2 with bedrooms = 4 is evaluated against utterance 1 (see Figure 12), which returns a value of 0.3. The value is then multiplied by the importance weight of the utterance, which yields a relevance of $r1 = 1.0 * 0.3 = 0.3$. Apartment 2 with rent = \$500 is then evaluated against utterance 2 (see Figure 13), which yields a relevance of $r2 = 0.9 * 0.95 = 0.855$. The total relevance of apartment 2 is:

- $R = r1 + r2 = 0.3 + 0.855 = 1.155$

The ranking for Apartment 3 is computed in a similar fashion. After the relevance of each item is recomputed the list is resorted based on relevance. The new ordering would now be as shown in Figure 14.

	City	Rent	Bedrooms	Relevance
2	Orem	\$500	4	1.155
3	Salt Lake	\$800	3	1.0285
1	Provo	\$200	1	0.95

Figure 14 - Reranked data list

Distance-comparable Attribute Metrics

Both numeric and distance-comparable values are supported under the current implementation of our list widget. The numeric attributes are handled quite simply using the distributions described previously. Non-numeric attributes are more problematic. Purely nominal attributes such as the manufacturer of an automobile or the owner of an apartment complex, where there is no distance metric between values, are handled using a simple metric. If two values are equal then their distance is zero otherwise the distance is huge. Using this metric, users can express their likes and dislikes, but little else.

String Attributes

Strings are an interesting case. They can have a lexicographical order, but that may not be meaningful for the problem. A user could use less than or greater than to focus in on a particular apartment complex name using the alphabetical order. Using alphabetic ordering in a speech system where the user cannot see the spelling is very difficult.

We think that strings are more effectively handled using some distance metric for similar sound or meaning. Various metrics such as minimal edit distance, longest common substring, longest common subsequence, and a weighted alphabetical distance can be used. Using the “equals” or “like/don’t like” utterances, the user can get string values that are close to what they spoke without having to say the word exactly right.

Semantically named attribute values

The most interesting class of attribute values is the names of objects, such as a city. If we make the name of a city be a surrogate for a particular geographic location, we can compute the Euclidian distance between cities and use that as a distance metric. We can compare two species of animals based on their distance from each other in a phylogenetic tree.

In studying several applications of our list widget we have encountered the problem of multiple definitions of distance. For example a comparison of two school districts can use their geographic distance. However, when people dislike a school district it is frequently for academic reasons. We may then use the difference between average standardized exam scores a distance metric. We now have two distance metrics that are equally valid for various classes of user. One would like to somehow use both and automatically discover the metric most appropriate for the situation. However, we do not yet have a general solution for this case.

SPOKEN FEEDBACK

At the heart of our list widget is speaking a description of the highest ranked list item to the user. The problem is that most useful data sets have many more attributes than can be spoken in a timely fashion. The user’s ability to

remember everything spoken decreases as the spoken feedback gets longer. The temporal nature of speech requires us to limit what is spoken to the user while at the same time allowing the user to control what and how much is spoken back. Certain attributes may be more important to a particular user than others; therefore we must ensure that these attributes are included in the feedback to the user.

To provide minimal feedback to the user that contains important attributes, we developed a simple algorithm based once again on the utterance history. Each attribute has a weight as to its importance in spoken feedback. The widget specification provides initial weights to seed the algorithm. Only those items with an importance value greater than a set threshold (we used 0.5) are spoken back to the user. (It should be noted that this feedback importance value is not the same as the weight assigned to each utterance.) Each time the user includes an attribute in an utterance, that attribute is given an importance value of 1.0. This is the case not only with critique commands but also commands that request attribute values. Each utterance thereafter that doesn't include that attribute causes the importance of that attribute to decay by 10%. Attributes are spoken back in their order of importance. This way the user always hears first what he has most recently spoken about.

To illustrate how this works, consider the following user session with an importance threshold of 0.8 (0.8 is high, but serves well to limit the example):

C7: "Car make is Honda, year is 1999, price is \$11,000"

Make	Model	Year	Price	Mileage
1.0	0.0	1.0	1.0	0.0

Make, year, and price are labeled in the data as default important attributes and are given importance values of 1.0.

U8: "What is mileage?"
C8: "mileage is 52,000"

Make	Model	Year	Price	Mileage
0.9	0.0	0.9	0.9	1.0

Mileage importance is set to 1.0, while all the other attributes decay by 10%. Mileage will be spoken first the next time the computer give feedback.

U9: "Next"
C9: "Car mileage is 53,361, make is Volkswagen, year is 1997, price is \$15,750"
U10: "What is model?"
C10: "Model is Jetta GLS"

Make	Model	Year	Price	Mileage
0.81	1.0	0.81	0.81	0.9

Model importance is set to 1.0, while all the other attributes decay by 10%.

U11: "Like model"
C11: "Car model is Jetta GLS, mileage is 53,361"

Make	Model	Year	Price	Mileage
0.729	1.0	0.729	0.729	0.81

The last feedback from the system only included model and mileage since their importance values are greater than the threshold we initially set of 0.8.

EVALUATION

We have performed a rough usability evaluation on our current list widget implementation. The specific questions that we wanted to answer with this evaluation was whether beginning users could learn the technique rapidly and use it effectively on lists of data that are normally unreasonable using speech. This was not a controlled experiment to compare competing approaches.

Using the WWW we extracted two example lists. The first is a list of 100 automobiles for sale. Each automobile has 11 attributes, including the make of the auto, price, number of miles, color, etc. Our second list is 122 student apartments offered for rent each with 13 attributes, including the rent, deposit required, distance from campus, number of bedrooms, etc.

For our usability test we invited in two groups of 5. The testers were university students from non-technical majors that had no prior experience with spoken language interfaces. We wanted to know how rapidly someone with limited instruction could accomplish information tasks using our spoken list widget.

We used the IBM ViaVoice recognition engine. Each user went through the speaker-dependent recognizer training before these tests to reduce recognizer failures as much as possible. Our work is focused on dialog design, not recognizer performance.

Our primary measure was the amount of time that it took to complete each task. As a benchmark we used the total time required to speak the entire list. It takes an average of 3 seconds to speak each attribute. The total speaking

time would be about 80 minutes for the entire apartment list and 55 minutes for the cars. Even if listening to the whole list were an effective technique, the time is prohibitive.

Group I was given the following series of tasks

- 3 tasks from the apartment list with only the value-specific commands (less, greater, equals)
- 3 tasks from the car list with only example-relative commands (like, don't like)
- 2 open-ended tasks with the apartment list using any commands they want
- 2 open-ended tasks with the car list using any commands

Group II was given the same tasks with the order of the lists reversed.

This test was designed to provide as much usability information as possible rather than for making controlled comparisons. Before each test, the users were given a one-page brief list of all of the available commands for the list widget.

Completion rates

Each tester performed 10 tasks for a total of 100 tasks. Out of those 100 tasks, 9 tasks were not completed within 10 minutes. Group I's use of example-relative commands on the car list accounted for 5 of the 9 failures. These 5 failures are all traceable to problems with nominal attributes such as model or color that do not have a distance metric. If a user is looking for a black vehicle, their only option was to say "don't like color" for every color until black is finally spoken. It is quite clear that for nominal attributes, the users must be able to specifically request a value.

Out of the 10 people tested, 6 of them completed all 10 tasks in less than 5 minutes. Of the remaining 4 people 1 failed to complete 3 of the tasks in less than 5 minutes and the remaining 3 people failed to complete 2 tasks. All testers learned both sets of commands within the first few tasks and were reasonably effective after that. Within the inherent timing constraints of speech, our spoken list widget performed quite well on large amounts of data with inexperienced users.

Task	Apartments		Cars	
	Ave seconds	Possible hits	Ave seconds	Possible hits
1	470	7%	241	1%
2	205	2%	106	2%
3	258	1%	61	3%

Table 1: Value-specific task times

Value-specific results

Both groups started with only the value-specific commands. Each of the three tasks asked about three attributes of the objects being searched. The performance times are as shown in Table 1.

Testers that used value-specific commands on the car list performed quite well and learned quickly. With the exception of one person on their first task, all users completed these tasks in less than 200 seconds (3.3 minutes). The "possible hits" data is the percentage of the list items that could satisfy the task criteria. Performance times are driven more by experience with the dialog than by the hit rate.

Example-relative commands

Both groups used the example-relative commands on their second set of tasks. As discussed earlier, group I's use of like/don't like commands on the nominal attributes of cars performed very poorly. The high rate of timeouts makes the other data irrelevant.

On the other hand group II performed quite well on the apartment data. These times are better than those for the value-specific commands.

Task	Apartments		Cars	
	Ave seconds	Possible hits	Ave seconds	Possible hits
1	208	7%	473	1%
2	159	2%	346	2%
3	200	1%	315	3%

Table 2: Example-relative task times

The data does not provide any clear evidence whether the value-specific or example-relative commands performed better.

Open-ended questions

Our last sets of tasks were open-ended. We wanted to see how users would perform based on their own desires and interests. We chose apartments and cars because students have an inherent interest in these items and their own personal biases on these subjects.

For the car list we posed two tasks. The first was to test the user's own desires. Each user was given a sheet with a list of the attributes of a car. They were asked to fill out the sheet with their desired car. The testers were then asked to use the car list to locate a car they would like to buy. They were then given an artificial scenario and asked to find a car that fit that scenario. A similar strategy was used with the apartment list.



	Apartments	cars
self -data	175	363
artificial	140	161

Table 3: Open-ended tasks – completion time

Testers completed their tasks in a timely manner. From the video tapes we did find that half of the testers changed their minds about what they wanted after working with the data. We are not sure if this is a realistic shift or tester fatigue. It was clear that the utterance history decay algorithm readily handled shifts of focus.

Command usage

A review of the videotapes showed the following distribution of command utterances by users.

Critiques	60%
Value Queries	30%
List Navigation	6%
Undo	4%

Table 4: Command usage

Most of the user interaction involved actual critique statements about what the user wanted. The second most popular were value queries for information about attributes not spoken in the initial presentation of a list item. The ability to scroll through the top choices in the ranked list was rarely used. After our preliminary tests, we added an Undo command to allow users to recover from recognition errors. This command was infrequently used. Recognition errors were not a large problem, and in many cases continued use of the critique commands overcame the problem naturally.

SUMMARY

We have described a spoken dialog system for accessing long lists of attributed objects. The system works by allowing users to comment on objects and their attribute values rather than formulating a query. The fuzzy nature of the query model facilitates location of objects when the user has imprecise goals. The use of a time-decay on utterances allows users to shift their focus without explicitly instructing the system.

The spoken language list widget has demonstrated its ability to access long lists of attributed data items. Users learned the technique rapidly. The example-relative commands are ineffective when used with attributes that have no distance metric to compare values.

REFERENCES

1. Buntschuh, B., Kamm, C., Di Fabrizio, G., Abella, A., Mohri, M., Narayanan, S., Zeljkovic, I., Sharp, R. D., Wright, J.H., Marcus, S., Shaffer, J., Duncan,

R. and Wilpon, J.G., "VPO: a spoken language interface to large scale directory information", Proc. ICSLP, Sydney, 1998.

2. Burke, R., Hammond, K., and Young, B. 'Knowledge-based navigation of complex information spaces', In Proceedings of the 13th National Conference on Artificial Intelligence AAAI96, pp. 462-468 (1996).
3. Burke R., 'The Wasabi Personal Shopper: A Case-Based Recommender System', in: Proceedings of the 16th National Conference on Artificial Intelligence AAAI99 (1999).
4. Chu, W., Yang, H., Chiang, K., Minock, M., Chow, G., and Larson, C. "CoBase: A scalable and extensible cooperative information system," Journal of Intelligent Information Systems, vol. 6, pp. 223-259 (May 1996).
5. Fuhr, Norbert, "A Probabilistic Framework for Vague Queries and Imprecise Information in Databases," in Proceedings of the 16th International Conference on Very Large Databases, pp. 696-707, (August 1990).
6. Gaasterland, T., Godfrey, P., Minker, J., "An Overview of Cooperative Answering," Journal of Intelligent Information Systems, vol. 1, no. 2, pp. 123-157, (1992).
7. Gaasterland, T., Godfrey, P., Minker, J., "Relaxation as a Platform of Cooperative Answering," Journal of Intelligent Information Systems, vol. 1, no. 3, pp. 293-321, (1992).
8. Göker, M. H., Thompson, C. A., "Personalized Conversational Case-Based Recommendation," in Advances in Case-Based Reasoning. Proceedings of the 5th European Workshop on Case-Based Reasoning, EWCBR 2000, Trento, Italy. LNAI (1998).
9. Resnick P., Varian H. (eds), 'Recommender Systems', Communications of the ACM, Vol. 40, No. 3, March (1997).
10. Rosenfeld, R., Olsen, D., and Rudnicky, A. 'Universal Speech Interfaces' interactions (Oct 2001).
11. Singh, S., Litman, D., Kearns, M., Walker, M., "Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System", Journal of Artificial Intelligence Research, vol. 16, pp. 105-133, (2002).